A Memory System for Robot Cognitive Architectures

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

DISSERTATION

von

Fabian Tërnava

ehem. Peller-Konrad

Tag der mündlichen Prüfung:	04. 06. 2025
Erster Referent:	Prof. Dr.-Ing. Tamim Asfour
Zweiter Referent:	Prof. Dr. Michael Beetz

“ *Those who cannot remember the past are condemned to repeat it*

George Santayana ”

– Wisdom that unfortunately never loses its relevance

Acknowledgments

Completing this PhD has been an incredible journey, and I am deeply grateful to the many people who have supported me along the way.

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Tamim Asfour, for his invaluable guidance, patience, and encouragement throughout my research. Working with and programming robots was always a dream, and this dream only became a reality thanks to Tamim's tireless efforts to promote this topic politically and scientifically. Also, his insights and expertise have been instrumental in shaping both this dissertation and my growth as a researcher.

A special thanks to my colleagues and friends at H^2T, whose collaboration, insightful conversations, and shared experiences made this journey both intellectually stimulating and enjoyable. I would especially acknowledge Joana Plewnia and Leonard Bärmann, whose great support during their time as students – and now as fellow scientists – has incredibly advanced my research. Our joint work has grown into a lasting friendship. I am especially grateful for all the friendships formed during countless hours of debugging code, crashing robots, writing papers, and exchanging ideas with all of my colleagues.

I am also indebted to my family, whose unwavering support, patience, and belief in me have been a continuous source of strength. Their encouragement has been a constant source of motivation. In particular, I want to thank my wife Claudia, whose devotion and tireless support – especially during the final phase – allowed me to fully dedicate myself to this work.

To everyone who has been part of this journey – thank you.

Abstract

The development of memory systems for robot cognitive architectures represents a crucial step toward creating intelligent, autonomous machines capable of learning from experience, adapting to new environments, and making informed decisions. Unlike traditional robotic control systems, which rely on rigid programming and predefined rules, a memory-centric cognitive architecture allows for the seamless integration of multi-modal information, enabling robots to process, store, and retrieve knowledge in a structured and context-aware manner. This research introduces a novel memory system for a robot cognitive architecture that acts as a mediator between high-level symbolic cognitive processes, such as reasoning and planning, and low-level sensorimotor functions, including perception and action execution. By identifying the limitations of existing memory frameworks, this work proposes an architecture that supports structured episodic memory, multi-modal data representation, and efficient information retrieval while incorporating adaptive forgetting mechanisms to manage memory constraints dynamically.

The memory system is implemented within the ArmarX robotic software framework, leveraging a hierarchical design that integrates working memory for short-term processing, long-term memory for knowledge retention, and episodic memory for the storage of past experiences. A key challenge in cognitive robotics is bridging the gap between symbolic and sub-symbolic representations, ensuring that robots can interpret, generalize, and apply past experiences to new situations. To overcome this, the system employs a flexible knowledge representation model, enabling efficient introspection and structured data storage. Memory consolidation techniques, inspired by cognitive psychology, ensure that relevant information is preserved while redundant or outdated data is gradually forgotten through frequency-based and relevance-driven filtering processes. Furthermore, the proposed system enables robots to build associations, e. g., between sensory perceptions and motor actions, allowing for more refined decision-making based on prior interactions and accumulated knowledge.

A critical component of this research is the evaluation of memory performance across various robotic tasks, demonstrating the system's ability to enhance robots' cognitive capabilities in real-world scenarios. Experimental results show that robots equipped with the memory system exhibit improved task efficiency, faster adaptation to changing conditions, and more effective interaction with humans and dynamic environments. The structured episodic memory model allows robots to recall and verbalize past experiences, improving transparency and explainability in decision-making. The integration of experience-based learning enables robots to anticipate the consequences or cost of actions, refine task execution strategies, and optimize planning processes over time. Additionally, the system's ability to process multi-modal sensory inputs, including vision, auditory signals, and proprioceptive data, facilitates richer knowledge representations.

By advancing the role of memory in cognitive robotic architectures, this research contributes to the development of more intelligent, autonomous systems capable of long-term learning and adaptive reasoning. The findings highlight the importance of memory-driven cognition in bridging the gap between traditional AI models and embodied intelligence. Future research directions include enhancing memory consolidation mechanisms through biologically inspired sleep-based reprocessing models, exploring large-scale distributed memory systems for multi-agent learning, and integrating advanced deep learning techniques to refine knowledge abstraction and generalization. The proposed memory framework serves as a foundation for further exploration in cognitive robotics, offering a scalable and extensible approach to enabling robots to understand, learn, and interact with their surroundings in a more human-like manner.

Deutsche Zusammenfassung

Die Entwicklung von Gedächtnissystemen für kognitive Roboterarchitekturen stellt einen entscheidenden Schritt auf dem Weg zur Schaffung intelligenter, autonomer Maschinen dar, die aus Erfahrungen lernen, sich an neue Umgebungen anpassen und fundierte Entscheidungen treffen können. Im Gegensatz zu traditionellen Robotersteuerungssystemen, die auf starrer Programmierung und vordefinierten Regeln basieren, ermöglicht eine gedächtniszentrierte kognitive Architektur die nahtlose Integration multimodaler Informationen. Dadurch können Roboter Wissen in strukturierter und kontextbewusster Weise verarbeiten, speichern und abrufen. Diese Arbeit stellt ein neuartiges Gedächtnissystem für robotische kognitive Architekturen vor, das als Vermittler zwischen high-level symbolischen kognitiven Prozessen – wie Schlussfolgern und Planen – und low-level sensomotorischen Funktionen – wie Wahrnehmung und Handlungsausführung – agiert. Durch die Identifikation der Einschränkungen bestehender Gedächtnisframeworks schlägt diese Arbeit eine Architektur vor, die strukturiertes episodisches Gedächtnis, multimodale Datenrepräsentation und effiziente Informationsabfrage unterstützt. Gleichzeitig werden adaptive Vergessensmechanismen integriert, um Speicherbeschränkungen dynamisch zu verwalten.

Das Gedächtnissystem ist im Rahmen des robotische Software-Frameworks ArmarX implementiert und nutzt ein hierarchisches Design, welches das Arbeitsgedächtnis für kurzfristige Verarbeitung, das Langzeitgedächtnis für Wissensspeicherung und das episodische Gedächtnis für die Speicherung vergangener Erfahrungen integriert. Eine zentrale Herausforderung in der kognitiven Robotik besteht darin, die Lücke zwischen symbolischen und subsymbolischen Repräsentationen zu schließen, damit Roboter vergangene Erfahrungen interpretieren, verallgemeinern und auf neue Situationen anwenden können. Um dies zu erreichen, setzt das System ein flexibles Wissensrepräsentationsmodell ein, das eine effiziente Introspektion und strukturierte Datenspeicherung ermöglicht. Gedächtniskonsolidierungstechniken, die von der kognitiven Psychologie inspiriert sind, sorgen dafür, dass relevante Informationen

erhalten bleiben, während redundante oder veraltete Daten durch frequenzbasierte und relevanzgesteuerte Filterprozesse schrittweise vergessen werden. Darüber hinaus erlaubt das vorgeschlagene System den Robotern, Assoziationen zu bilden, etwa zwischen sensorischen Wahrnehmungen und motorischen Aktionen, wodurch eine verfeinerte Entscheidungsfindung auf Basis vorheriger Interaktionen und angesammelten Wissens möglich wird.

Ein zentraler Bestandteil dieser Forschung ist die Evaluierung der Gedächtnisleistung in verschiedenen robotischen Aufgaben, um die Fähigkeit des Systems zur Verbesserung kognitiver Fähigkeiten in realen Szenarien zu demonstrieren. Experimentelle Ergebnisse zeigen, dass Roboter mit dem Gedächtnissystem eine höhere Aufgabeneffizienz, eine schnellere Anpassung an sich ändernde Bedingungen und eine effektivere Interaktion mit Menschen und dynamischen Umgebungen aufweisen. Das strukturierte episodische Gedächtnismodell ermöglicht es Robotern, vergangene Erfahrungen abzurufen und zu verbalisieren, wodurch die Transparenz und Erklärbarkeit von Entscheidungen verbessert wird. Die Integration erfahrungsbasierter Lernprozesse erlaubt es den Robotern, die Konsequenzen oder Kosten von Handlungen vorherzusehen, Strategien zur Aufgabenbewältigung zu verfeinern und Planungsprozesse im Laufe der Zeit zu optimieren. Zusätzlich ermöglicht die Fähigkeit des Systems, multimodale sensorische Eingaben – einschließlich visueller, auditiver und propriozeptiver Daten – zu verarbeiten, eine reichhaltigere Wissensrepräsentation.

Durch die Weiterentwicklung der Rolle des Gedächtnisses in kognitiven Roboterarchitekturen leistet diese Forschung einen Beitrag zur Entwicklung intelligenterer, autonomer Systeme, die langfristiges Lernen und adaptives Denken ermöglichen. Die Ergebnisse unterstreichen die Bedeutung einer gedächtnisgesteuerten Kognition als Brücke zwischen traditionellen KI-Modellen und verkörperter Intelligenz. Zukünftige Forschungsvorhaben umfassen die Verbesserung der Gedächtniskonsolidierungsmechanismen durch biologisch inspirierte, schlafbasierte Reprozessierungsmodelle, die Erforschung großskaliger verteilter Gedächtnissysteme für Multi-Agenten-Lernen und die Integration fortschrittlicher Deep-Learning-Techniken zur Verfeinerung der Wissensabstraktion und -verallgemeinerung. Das vorgeschlagene Gedächtnisframework dient als Grundlage für weitere Erkundungen in der kognitiven Robotik und bietet einen skalierbaren und erweiterbaren Ansatz, um Robotern ein menschenähnlicheres Verstehen, Lernen und Interagieren mit ihrer Umgebung zu ermöglichen.

Contents

CHAPTER 1

Introduction

In recent decades, robots have become increasingly essential in a wide array of industries, ranging from manufacturing and healthcare to education and space exploration. Their ability to precisely perform repetitive tasks, operate in hazardous environments, and augment human capabilities has transformed the way we approach challenges in these fields. As robots continue to evolve, their roles are expanding beyond the traditional confines of mechanical automation into domains that require more complex decision-making, adaptability, and interaction with dynamic environments and humans. The rise of cognitive robotics – a field that aims to imbue robots with human-like cognitive abilities – marks a critical step towards creating machines with intelligent behavior.

But what exactly does *cognition* mean? Vernon (2014a,b) define cognition as

> “ *the process by which an autonomous system perceives its environment, learns from experience, anticipates the outcome of events, acts to pursue goals, and adapts to changing circumstances.* ”

This definition highlights three essential components for cognitive robotic systems:

1. *perception* components that allow the system to perceive its environment,
2. *processing* components that enable learning from experience, predicting the outcomes of actions, and adapting to evolving conditions, and
3. *action execution* components that enable goal-directed actions.

A robot’s cognitive architecture must incorporate a repository for information gathered through perception and action execution, enabling the processing components to access, analyze, and ultimately store results – this repository is referred to as memory and is central to the architecture.

Human memory is one of the most remarkable and complex systems in nature, capable of processing vast amounts of sensory data in a highly distributed manner. Humanoid robots encounter comparable challenges, making the development of memory-based cognitive control architectures a critical area of research to equip these robots with intelligent behaviors and enable smooth, human-like interactions with their environment. To accomplish tasks such as object recognition, task planning and execution, navigation, motion control, learning, and interaction, various processes are required to filter, interpret, and utilize the memorized data effectively.

Thus, we argue that a *memory system* is a fundamental element in any cognitive architecture. Such a memory system should not only serve to integrate different system components and store multi-modal information at various levels of abstraction but also provide mechanisms for deriving semantics from sensorimotor data, parameterizing symbolic plans for context-specific execution, and predicting the effects of actions.

We propose our first hypothesis for a memory system for a robot cognitive architecture:

Hypothesis 1

A memory system in a cognitive robotic control architecture acts as a mediator between (1) high-level abilities, typically represented symbolically, such as language comprehension, scene understanding, planning, plan execution monitoring, and reasoning, and (2) low-level abilities, such as sensor data processing and sensorimotor control.

This implies that the memory system must be capable of handling vast amounts of data – whether the data is symbolic (e. g., plans, words, relationships) or sub-symbolic (e. g., images, joint configurations, forces). It must bridge the gap between sub-symbolic (sensorimotor) and symbolic (semantic) representations, addressing the *signal-to-symbol gap* by, for example, learning representations of perception-action dependencies in the form of object-action complexes (OACs), as suggested by Krüger et al. (2011).

From this observation arises our second hypothesis:

Hypothesis 2

Representations are key.

The capability to store and retrieve multi-modal information, the efficiency of these processes, and the ability to learn from such data all depend on a meaningful and efficient multi-modal representation of knowledge. This representation must be sufficiently specific to distinguish between symbolic and sub-symbolic information, while also supporting generalization across both sensorimotor and symbolic levels. Furthermore, it must facilitate *knowledge associa-*

tions (Wood et al., 2012), as the system requires an understanding of how perception and action are interrelated and which sensory experiences typically co-occur. This multi-modal framework is essential for integrating information generated by various cognitive processes within the architecture (Paulius and Sun, 2019).

Finally:

Hypothesis 3

Knowing when and how knowledge changes is just as important as the knowledge itself.

Episodic knowledge – the understanding derived from specific experiences – shapes learning by linking past events to new contexts. Unlike abstract knowledge, it retains situational details, fostering adaptability and informed decision-making. By tracking how knowledge evolves, individuals recognize patterns, refine assumptions, and apply lessons effectively. This, in combination with introspective representations, supports metacognition, enabling reflection on past successes and failures to enhance problem-solving. Understanding its role has broad implications for learning, artificial intelligence, and development in the architecture. Learning from experience through episodic knowledge is key to intellectual growth and adaptation in a rapidly changing world which is why a framework that aims to combine reactive and deliberative components requires this capability (Paulius and Sun, 2019).

The three hypotheses serve as the conceptual backbone that guides the entire research and development process presented in this thesis. However, one of the key limitations hindering the realization of truly autonomous and intelligent robots lies in the development of memory-centric cognitive control architectures that can effectively manage the vast amounts of sensory data generated by robots and enable them to learn, reason, and act in complex environments. How can we develop robotic systems with advanced cognitive and sensorimotor capabilities that rival those of humans, particularly in terms of learning and development? One of the most significant limitations of classical robotics is the reliance on predefined programming and rigid control systems that lack the flexibility to adapt to new situations (Cushing et al., 2007). While advances in artificial intelligence and machine learning have enabled robots to perform tasks like object recognition, motion planning, and basic interaction, these processes often occur in isolation. What is missing is an overarching cognitive framework – connecting core cognitive components via a centralized memory – that allows robots to contextualize their actions within a larger goal, learn from experiences, and adjust to novel circumstances. Such an architecture can further provide insights into cognitive functions that are not merely theoretical but grounded in practical applications, since robots embody the principles of embodied cognition, emphasizing the importance of the brain-body-environment coupling in understanding cognitive processes (Hoffmann and Pfeifer, 2018).

1.1. Problem Statement

We argued that a comprehensive memory system is central to overcoming the challenges faced by cognitive robots. Memory systems in robotics are not simply data repositories; they are integral to the functioning of a cognitive control architecture. In humans, memory allows for the storage and retrieval of diverse types of information, from sensory experiences to abstract reasoning, enabling complex thought processes, decision-making, and adaptive behaviors. Similarly, robots require a memory system that can store and process both symbolic data (e. g., language, rules, plans) and sub-symbolic data (e. g., sensory inputs, motor commands), bridging the gap between perception and action.

For a robot to exhibit intelligent behavior, its memory must support several critical functions. It must allow the robot to store and retrieve information from past interactions to guide future actions. For example, if a robot encounters an obstacle while navigating a room, it should be able to recall this experience and modify its path to avoid similar obstacles in the future. Further, the memory system must integrate multi-modal data – combining visual, auditory, and tactile inputs with internal models of the world – so the robot can form a cohesive understanding of its environment. Also, memory must enable the robot to learn from its experiences and generalize that learning to new contexts, a process akin to human cognitive development. Of course, all of the aforementioned processes should be efficient.

To address the memory-related challenges in cognitive robotics, several conceptual key requirements must be met. First and most important, the memory system needs to be *active*, meaning it should not simply store data but actively process and retrieve relevant information as needed by the robot. Second, the memory should be *episodic*, organizing experiences into temporally structured episodes, allowing the robot to recall and learn from specific past events. Third, the memory must be *multi-modal*, capable of handling data from various sources, such as visual, auditory, and tactile inputs, in a unified manner. Fourth, the system should be *associative*, enabling the robot to link different pieces of information, such as associating an object with a particular action or sensory input. Finally, the memory should be *introspective*, allowing the robot to evaluate its own knowledge and adapt its behavior based on this self-assessment.

From a technical perspective, the memory system must be *distributed*, *access efficient*, and *long-term*, ensuring that it can scale with the growing complexity of cognitive tasks without compromising performance. By integrating these characteristics, a comprehensive memory system can empower cognitive robots to perform more sophisticated tasks, learn autonomously, and interact with their environment in a more human-like manner.

We formalize our main objective as follows:

Main Objective

How can we build a flexible and extensible memory system in a robot cognitive architecture that is *active*, *episodic*, *multi-modal*, *associative* and *introspective* while being highly *distributed*, *access efficient* and supports *long-term* storage, even for a lifetime of experiences?

In recent years, foundation models, particularly large-scale models based on deep learning, have gained significant attention due to their impressive capabilities in tasks like language generation, image recognition, and translation. These models, which include systems such as GPT (OpenAI, 2025) and BERT (Devlin et al., 2019), are trained on massive amounts of data and can generate responses that appear highly intelligent and contextually appropriate. Despite these impressive abilities, foundation models have often been criticized as "parrot systems" (Beetz, 2024) – mimicking cognition rather than truly embodying it – because they generate outputs based on patterns they have observed in the training data, without understanding or reasoning about the content. A famous joke in the AI community describes a foundation model that enters a bar. The bartender asks, "*What can I get you?*" The model replies, "*I don't know. Just give me the average of all the drinks you've ever served.*", highlighting the model's lack of true understanding, reasoning or individual decision-making capabilities. These models are trained to predict the next word in a sentence or recognize patterns in images, but they do not possess an internal model of themselves or of the world. As a result, they lack the ability to engage in true cognitive processes such as reasoning, decision-making, and learning from experience in a dynamic environment. While these models can produce convincing outputs, their behavior is reactive and limited to the specific tasks they were trained on. Robots that are controlled end-to-end by foundation models are far away from achieving human-like cognitive abilities as we know them from science fiction, in which robots are able to think, learn and adapt to new situations.

In contrast, we believe that cognitive robots should aim to emulate human-like cognition by integrating perception, reasoning, learning, and action components within a unified control architecture, that (1) contributes to understanding aspects of human intelligence, including elements related to prominent theories, such as *theory of mind* (Newell, 1990), (2) eases the development of cognitive components, as the architecture is modular and can be extended with new components, and (3) supports introspection and explainability of the system and that memory, and especially the collection of personalized experiences, i. e., Episodic Memory (EM), is key in their architecture.

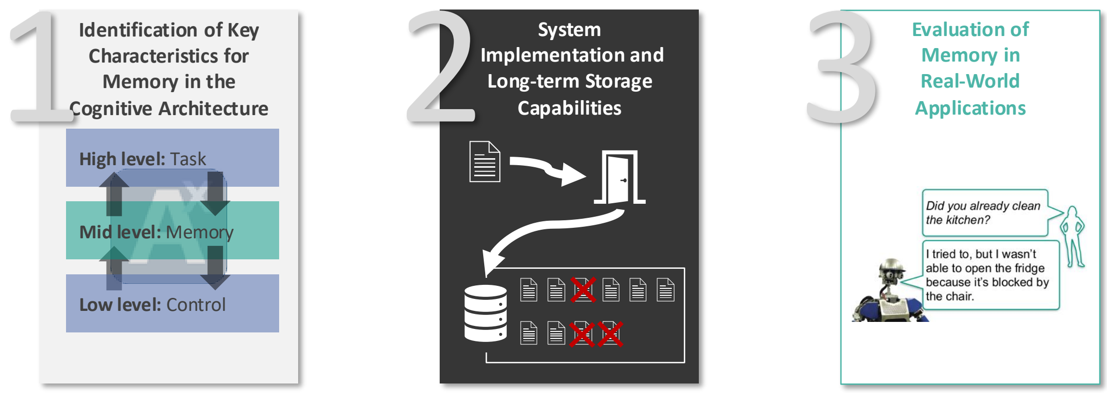

1
Identification of Key Characteristics for Memory in the Cognitive Architecture
High level: Task
Mid level: Memory
Low level: Control
2
System Implementation and Long-term Storage Capabilities
3
Evaluation of Memory in Real-World Applications
Did you already clean the kitchen?
I tried to, but I wasn't able to open the fridge because it's blocked by the chair.

Figure 1.1.: This thesis is structured into three major contributions: (1) Identification of key characteristics for a memory system for cognitive architectures and its design. (2) Implementation of a memory system for a cognitive architecture in ArmarX. Both contributions are covered in Chapter 4. (3) Evaluation of Efficiency and Applicability using evaluation and example scenarios. This is then covered in Chapter 5, 6 and 7.

1.2. Contributions

The goal of this thesis is to address limitations of existing memory systems in cognitive robotics and propose a novel memory-centric cognitive architecture that meets the complex requirements of such systems. By building on insights from cognitive science, artificial intelligence, and robotics, we aim to develop an efficient memory architecture that enables robots to learn, reason, and adapt in real-world environments. Specifically, this thesis will explore how memory can mediate between arbitrary high-level cognitive functions – such as planning, reasoning, and language processing – and low-level sensorimotor control, providing a unified framework for intelligent behavior.

In summary, we present a detailed technical overview of the memory system's implementation, which meets the demands of complex robotic systems, such as humanoid robots, and provide a comprehensive evaluation of its efficiency and applicability. It is important to note that this work does not describe a complete cognitive architecture but rather focuses on the memory system component within such architectures.

In the following, we will explain the three main contributions of this thesis, as shown in Figure 1.1.

1.2.1. Identification of Key Characteristics

At the conceptual level, we identify and motivate essential characteristics of a memory system and its knowledge representation in the cognitive control architecture of a humanoid robot, which support our three hypotheses. As argued, these characteristics are embodied in five key requirements: *active*, *episodic*, *multi-modal*, *associative*, and *introspective*. Additionally, we

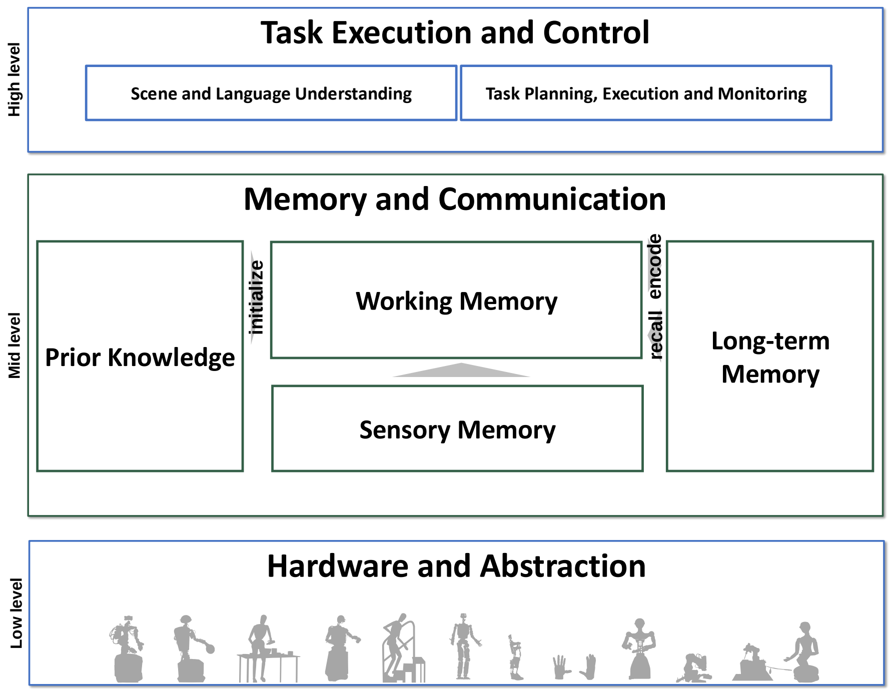


Figure 1.2.: Embedding of the implemented memory system within the cognitive architecture of the robot software framework ArmarX. Located in the middle between high-level capabilities and low-level control, the memory consists of Working Memory (WM), Long-term Memory (LTM) and Prior Knowledge (PK). All communication passes through the memory, making it an essential mediator between the two layers. A unified knowledge representation framework definitely defines memory functions. ©Elsevier (Peller-Konrad et al., 2023), modified

highlight the need of technical requirements: *distributed*, *efficient*, and *long-term*. We argue that memory is not merely a passive storage device within the architecture but an active element capable of processing both symbolic and sub-symbolic multi-modal data. The ability to inspect stored information and adapt behaviors based on that information is crucial for supporting this active role. Furthermore, we argue that data should be managed in an episodic-like manner, irrespective of whether it pertains to declarative or non-declarative knowledge. Finally, we explain how forming associations between shared knowledge within the system and extracting meaningful information from data enhance the efficiency of data processing, learning, and externalization. The technical requirements, meanwhile, focus on optimizing the system's overall performance and improving scalability. We argue that memory-driven cognitive control architectures with these core characteristics are essential for the development of humanoid robots with advanced cognitive abilities.

1.2.2. Implementation of Memory System

The aforementioned principles have guided the implementation of a highly extensible memory system with these core features in our robot software framework ArmarX, depicted in

Section 1.2.2. Drawing inspiration from cognitive science, we designed the memory system to structure and optimize data flow in working memory and long-term memory, and simplify the development and integration of new core cognitive components. In particular, we will focus on storing large amounts (i. e., days, weeks or a lifetime) of data in long-term memory. In order to remain as efficient as possible in this particular case, the memory must have the ability to forget irrelevant or redundant data and to generalize knowledge. But memory should not only be used to remember what happened in the past. Memory is also crucial for predicting the future – a capability that also requires the data to be introspectable.

In our LTM we propose different filtering mechanisms as well as a Deep Episodic Memory (deep EM) to generalize and predict knowledge using deep learning. Forgetting methods can be used in two places in this system. Online methods are used to filter data from working memory to long-term memory. Due to the high throughput, these methods must be real-time capable. Offline methods, on the other hand, are used to filter or abstract data from long-term memory when the robot is not in operation.

The implementation also introduces a novel approach to representing arbitrary information in memory, allowing real-time inspection of data through variant-based data types while supporting strictly typed data structures within the solely strictly typed language C++ for its performance benefits. This data representation, and thus the memory system itself, has been used to support multiple target programming languages and multiple middleware systems.

1.2.3. Evaluation of Efficiency and Applicability of Memory System

Finally, we evaluate the efficiency of the memory system. First, the memory is evaluated quantitatively. To do this, we compare transmission speed and response times with direct connections (i. e., without memory) and central databases. In addition, we evaluate different forgetting methods. To this end, we investigate the extent to which the data in the memory can be reduced (filtered or abstracted) in order to solve an object-recognition task without any loss of performance. In addition, the ability of deep episodic memory to reconstruct and predict knowledge on different data sets was investigated.

Finally, we will analyze the applicability of memory in two exemplary tasks – verbalization and experience-based symbolic planning – and outline other components, that benefit from our proposed architecture, such as grasping and manipulation.

1.3. Structure of the Thesis

Chapter 2 provides a comprehensive review of related research on memory from cognitive psychology, neuroscience, and computational perspectives. Within the domain of cognitive psychology, the historical development of memory classification is introduced, outlining the different types of memory and their theoretical foundations. Additionally, case studies of prominent patients who have significantly contributed to our understanding of memory function and classification are examined. Empirical findings from neuroscience support these classifications, highlighting the brain regions that play a crucial role in memory formation and retrieval. The discussion then extends to artificial cognitive architectures and artificial memory systems, which are categorized into three primary paradigms: Cognitivist Architectures, Emergent Architectures, and Hybrid Architectures. Particular emphasis is placed on the mechanisms of knowledge representation and processing within these systems. Furthermore, we discuss forgetting as a fundamental function of memory. Finally, common errors in the modeling of cognition and knowledge representation within artificial cognitive architectures are examined.

Chapter 3 illustrates the importance of memory in robotic cognitive architectures through several example scenarios. A typical fetch-and-carry task is analyzed from multiple perspectives, demonstrating that (1) memory must accommodate and manage diverse modalities, necessitating a highly flexible memory structure; (2) episodic memory plays a fundamental role in task execution; (3) the ability to forget and abstract knowledge is essential for efficient memory management; (4) memory is integral to a broad spectrum of cognitive functions.

Chapter 4 presents the design and implementation of a memory system for cognitive robotic architectures. Initially, the conceptual and technical requirements that a general memory system must fulfill within a cognitive architecture are identified. Subsequently, the implementation of the memory system within the ArmarX robot software framework is described in detail. The discussion encompasses the employed knowledge representation, the data exchange layer, and the mechanisms for volatile and persistent data storage. Additionally, strategies for the efficient management and selective deletion (forgetting) of complete datasets or partial data are examined.

Chapter 5 outlines the methodology and results of the memory system evaluation. Three distinct evaluation scenarios are considered: (1) Assessment of working memory, focusing on the availability and real-time performance of the system's components; (2) Evaluation of long-term memory, examining its capacity for data compression and detection of redundant information; (3) Analysis of various forgetting mechanisms and their parameterizations in the context of a specific task.

Chapter 6 examines the application of long-term memory in the verbalization of robot experiences. A brief overview of related work in the field of verbalization is provided. Given the data-driven approach adopted in this study, the dataset and learning method are described in detail, establishing links to the preceding chapters on memory and memory implementation. Finally, the results of the verbalization process are presented and analyzed.

Chapter 7 describes how a robot can use knowledge about past action executions to improve its decision making for time-critical tasks in the future. Related topics are introduced first. Then, as an example of a task that requires learning from experience, the problem of planning with variable cost-risk functions is discussed in detail and finally, methods are presented and analyzed to estimate and continuously improve these functions from experience.

Chapter 8 concludes the work and discusses possible extensions.

CHAPTER 2

State of the Art

Memory can be viewed from different perspectives. The cognitive psychological view of memory focuses on understanding how information is encoded, stored, and retrieved within the **mind** (Craik et al., 1996; Jaime et al., 2012). In this view, memory is often conceptualized as a set of processes that operate on sensory inputs to form representations of experiences, which can later be accessed consciously or unconsciously. Cognitive psychologists study various stages of memory, conceptualizing it as a dynamic and structured system that operates like information processing. This view is often informed by models and theories that describe different types of memory systems, their interactions, and the mechanisms underlying memory function. Findings from clinical observations and therapeutic processes about patients with amnesia, dementia or trauma and brain injuries contribute to the understanding of memory within cognitive psychology.

The neuroscientific view on memory examines how memory processes are supported by neural structures and biological mechanisms within the **brain** (Thompson and Kim, 1996). This perspective seeks to understand how neurons, brain regions, and biochemical processes contribute to the encoding, storage, and retrieval of memories. In this view, memory is not seen solely as a cognitive function but as a result of physical changes within the brain, shaped by patterns of neural activity and the connections among neurons and brain regions.

The computational view on memory examines memory as an information-processing system, inspired by computer science and artificial intelligence principles. In this view, memory is conceptualized as a system of data storage, manipulation, and retrieval, analogous to how a computer processes, stores, and accesses information. Researchers in this field aim to understand memory by developing computational models and simulations, which allow for testing theories on how memory might work within artificial and biological systems.

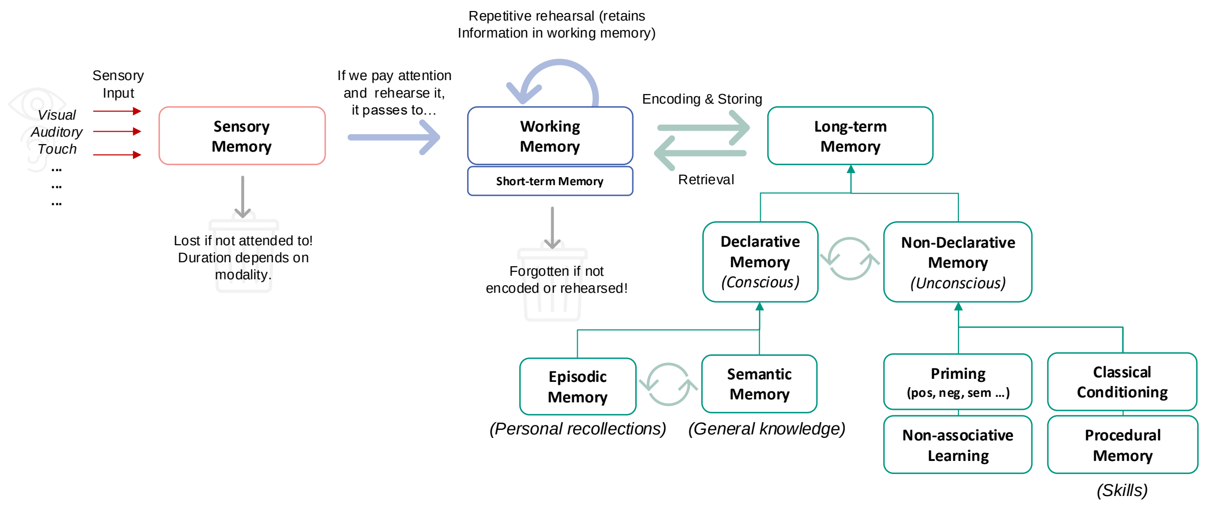

Repetitive rehearsal (retains Information in working memory)
Sensory Input
Visual
Auditory
Touch
...
Sensory Memory
Lost if not attended to! Duration depends on modality.
If we pay attention and rehearse it, it passes to…
Working Memory
Short-term Memory
Forgotten if not encoded or rehearsed!
Encoding & Storing
Retrieval
Long-term Memory
Declarative Memory (Conscious)
Non-Declarative Memory (Unconscious)
Episodic Memory
Semantic Memory
(Personal recollections)
(General knowledge)
Priming (pos, neg, sem ...)
Classical Conditioning
Non-associative Learning
Procedural Memory
(Skills)

Figure 2.1.: Overview of the memory taxonomy, illustrating the division of memory into Sensory Memory (red), WM (blue), and LTM (green). LTM is further categorized into Procedural Mmemory (PM), EM, and Semantic Memory (SM), highlighting the hierarchical structure of cognitive memory systems. There is constant transfer between episodic and semnatic memory as well as decalrative and non-declarative memory. ©Elsevier (Peller-Konrad et al., 2023), modified

This chapter provides an exploration of memory through multiple perspectives, delving into the cognitive psychological, neuroscientific, and computational views. Additionally, it examines specific aspects of memory, including episodic memory, long-term storage mechanisms, and common misconceptions about memory function in artificial cognitive systems.

2.1. Cognitive Psychological View on Memory

Cognitive psychology explores memory as a structured and dynamic system, encompassing processes like encoding, storage, and retrieval. By understanding how these processes operate, researchers have developed influential theories and models that offer insights into the nature of memory and its functions.

This section delves into key aspects of the cognitive psychological view of memory. It begins by examining the invaluable contributions of patients with memory impairments, whose experiences have shaped our understanding of memory systems and their neural underpinnings. Some findings also contribute to the neuroscientific view of memory as described in 2.2. It then explores foundational theories such as Atkinson and Shiffrin's Multi-Store Model, which outlines the interaction between sensory, short-term, and LTM stores, and Baddeley and Hitch's Working Memory Model, which provides a nuanced view of short-term memory processes. Finally, the section highlights the importance of episodic memory, the system that allows individuals to recall personal experiences and navigate their autobiographical past. Figure 2.1 illustrates a nowadays widely accepted taxonomy of memory types, categorizing them into distinct systems such as sensory memory, short-term memory, and long-term memory, with further subdivisions like episodic, semantic, and procedural memory.

2.1.1. How Patients Shape the Understanding on Memory

The cognitive psychological view of memory is significantly informed by the study of patients with amnesia, dementia, and brain injuries or by psychological studies. These elements provide valuable insights into the functioning of memory systems and the underlying neural mechanisms, enhancing our understanding of how memory operates in both healthy and impaired states.

Famous patients, such as Henry Molaison (usually referred as H.M.) and Clive Wearing (usually referred as C.W.) (McComas, 2022), have been pivotal in shaping the understanding of memory. H.M. underwent a bilateral medial temporal lobectomy to alleviate severe epilepsy, which resulted in profound anterograde amnesia – an inability to form new explicit memories while retaining older memories and intact procedural memory (Hassabis et al., 2007).

His case demonstrated the distinction between different types of LTM, particularly the separation of declarative (explicit) and procedural (implicit) memory systems. The insights gained from H.M. and similar cases have underscored the role of the Hippocampus (HPC) in the consolidation of new memories and the importance of specific brain regions in different memory processes (Hassabis et al., 2007).

C.W. is known for his profound amnesia resulting from a viral infection that led to damage in the medial temporal lobe, specifically affecting the HPC and surrounding structures (Chu et al., 2007). Similarly, this case has been instrumental in understanding the distinction between different types of memory. Amnesia, particularly in its various forms, offers critical insights into the cognitive processes underlying memory.

Patients usually participate in psychological tests, such as the Wechsler Memory Scale and the Rey-Osterrieth Complex Figure Test, which commonly used to assess various aspects of memory function (Al-Qazzaz et al., 2014). These tests help differentiate between types of memory impairments, such as those seen in dementia versus those resulting from brain injuries. E. g., such tests have been used extensively to assess C.W.'s memory capabilities, revealing significant deficits in his ability to recall new information. For instance, tests measuring verbal and visual memory have shown that while C.W. can perform tasks that rely on previously learned skills, he struggles with tasks that require the recall of new information (McComas, 2022).

Dementia, characterized by progressive memory loss and cognitive decline, further illustrates the complexities of memory. Different types of dementia, such as Alzheimer's disease and frontotemporal dementia, exhibit distinct patterns of memory impairment. Alzheimer's disease is often associated with early deficits in episodic memory, while frontotemporal dementia may impact executive functions and social cognition more prominently (Thomas-Antérion et al., 2000).

Brain injuries, such as those resulting from strokes or traumatic brain injuries, can lead to various memory impairments, depending on the location and extent of the damage. For instance, individuals who have experienced a stroke may exhibit deficits in both short-term and LTM, highlighting the importance of specific brain regions in memory processing (Al-Qazzaz et al., 2014). The study of memory recovery following brain injuries also informs cognitive rehabilitation strategies, emphasizing the potential for neuroplasticity and recovery of function (Deal et al., 2016).

Finally, traumatic events contribute to our understanding of memory by influencing how memories are formed, stored, and retrieved. One of the key aspects of trauma's influence on memory is the phenomenon of intrusive memories. Bonsall and Holmes (2023) describe how traumatic memories often manifest as involuntary and distressing recollections that intrude upon an individual's thoughts. These memories are characterized by their emotional intensity and persistence, which can lead to significant distress and impairment in daily functioning. Further, traumatic memories often differ from non-traumatic memories in terms of their vividness and emotional intensity. Porter and Birt (2001) found that traumatic memories are associated with a richness of detail and heightened emotional information compared to positive emotional memories. The oscillation between the integration of trauma-related information and psychological defenses can result in flashbacks and intrusive memories, highlighting the complex interplay between cognitive processes and emotional responses to trauma. Bisby et al. (2020) emphasize that traumatic events do not produce a singular "trauma memory", but rather a series of fragmented memories that correspond to different aspects of the traumatic experience. This disjointedness can lead to difficulties in forming coherent autobiographical narratives, which is often observed in individuals with post-traumatic stress disorder (PTSD). Radulovic (2017) discusses how traumatic memories may be encoded within a broader socio-cognitive-affective circuit, suggesting that these memories are influenced by various cognitive and emotional factors. This perspective aligns with the notion that trauma can lead to alterations in brain function, particularly in areas related to memory processing, such as the HPC and the limbic system (Levine, 2015).

2.1.2. Atkinson and Shiffrin's Multi-Store Model

One of the central models in cognitive psychology is the multi-store model of memory (Atkinson and Shiffrin, 1968), which divides memory into different stages: *sensory memory*, *short-term memory*, and *long-term memory*. This model emphasizes the sequential flow of information through these stores, highlighting the processes of encoding, storage, and retrieval.

Sensory memory is the first stage in the model, where sensory information is briefly held in its raw form. It acts as a buffer for stimuli received through the senses, retaining impressions for a very short duration – typically less than a second for visual stimuli (iconic memory) and a few seconds for auditory stimuli (echoic memory) (Patel et al., 2019). Following sensory memory, information that is attended to enters short-term memory. This stage is characterized by its limited capacity, often cited as being able to hold approximately seven "items" (or chunks, plus or minus two) at any given time (Plancher and Barrouillet, 2019). Short-term memory serves not only as a temporary storage system but also as a workspace for cognitive tasks, where information can be manipulated and rehearsed. The model posits that through processes such as rehearsal, information can be encoded into LTM, which is the final stage of the model (Unrug et al., 1997). LTM is capable of storing vast amounts of information for extended periods, ranging from days to a lifetime, and is further divided into explicit (declarative) and implicit (non-declarative) memory systems (Squire, 2004).

Despite its influential role in memory research, the Atkinson and Shiffrin model has faced criticism. Critics argue that the model oversimplifies the complexities of memory processes by depicting a linear flow of information and suggesting a single short-term store responsible for both maintenance and processing tasks (Plancher and Barrouillet, 2019). Recent research has proposed more nuanced perspectives, such as the working memory model, which distinguishes between different types of short-term storage and processing systems (Unrug et al., 1997). Furthermore, the model does not account for the influence of contextual and social factors on memory, which have been shown to play significant roles in how information is remembered and retrieved (Cheng and Schwing, 2022).

2.1.3. Baddeley and Hitch's Working Memory Model

The Baddeley and Hitch Working Memory Model (Baddeley and Hitch, 1974) significantly advanced the understanding of short-term memory by proposing a more complex structure than the traditional view of a single storage system. This model delineates working memory into multiple components, specifically a central executive and two slave systems: the phonological loop and the visuospatial sketchpad. In later iterations, an additional component known as the episodic buffer was introduced, further enriching the model's explanatory power regarding cognitive processes (Logie, 2014; Masoura et al., 2020).

The central executive serves as the control system that oversees and coordinates the activities of the slave systems. It is responsible for allocating attention, managing cognitive tasks, and integrating information from various sources (Hegarty et al., 2000; Moscovitch, 1994). This component can process information from different sensory modalities, thereby allowing for flexible cognitive functioning and is particularly crucial in tasks that require the manipulation of information, such as problem-solving and reasoning, as it directs the flow of information and prioritizes tasks based on cognitive demands (Harasimczuk, 2024; Linden et al., 1994).

The phonological loop is one of the slave systems dedicated to verbal and auditory information. It consists of two subcomponents: the phonological store, which temporarily holds auditory information, and the articulatory rehearsal process, which allows for the maintenance of this information through subvocal repetition (Jarrold et al., 2000; Morris and Jones, 1990). Thus, this system is essential for language comprehension and verbal learning, as it enables individuals to retain and manipulate verbal information over short periods.

The visuospatial sketchpad, the second slave system, is responsible for processing and storing visual and spatial information. It allows individuals to visualize and manipulate images and spatial relationships, which is crucial for tasks such as navigation and understanding diagrams (Logie, 2014; Masoura et al., 2020).

Later, the episodic buffer got introduced as a fourth component of the working memory model. This addition was made to address the limitations of the original model in integrating information from the phonological loop and visuospatial sketchpad with LTM. The episodic buffer acts as a temporary storage system that allows for the integration of information from various sources, facilitating the creation of coherent episodes or experiences (Harasimczuk, 2024; Masoura et al., 2020). This component is particularly important in tasks that require the integration of multimodal (here verbal and visual) information.

However, the model is not without criticism; some researchers argue that the model's components, especially the central executive and its subsystems, are too vague to mimick human cognition and that more specific mechanisms should be identified (Duff, 2000; Hegarty et al., 2000).

2.1.4. Tulving's Model of Declarative Long-Term Memory

Tulving (1972) proposed that LTM can be categorized into different systems, each serving distinct functions and characterized by unique properties. His model emphasizes the functional and neural dissociation between episodic and semantic memory, which has been foundational in understanding how humans process and retrieve information.

Episodic memory (EM) refers to the ability to recall specific events or experiences from one's past, including contextual details of "what", "where", and "when" events occurred (Ergorul and Eichenbaum, 2004). EM is essential for personal identity and continuity and plays a significant role for forming a coherent narrative of one's life experiences. In contrast, semantic memory encompasses general knowledge about the world, facts, and concepts that are not tied to specific experiences (Grande et al., 2021; Renoult and Rugg, 2020). This distinction highlights that while EM is tied to personal experience, semantic memory is more about factual information devoid of personal context.

Tulving's serial parallel independent model (Tulving, 1984) further elaborates on these distinctions by suggesting that the encoding of information is a serial process where perceptual information first feeds into the semantic memory system. This system then transmits relevant information to EM, indicating a hierarchical relationship between them. This model has been supported by neuropsychological evidence showing that damage to specific brain areas can selectively impair one type of memory while leaving the other intact, thereby reinforcing the idea of their functional independence (Renoult and Rugg, 2020). Further, studies have shown that both episodic and semantic memories benefit from different stages of sleep, which supports the notion that these memory systems operate independently yet interactively (Rauchs et al., 2005).

Research proved that EM is not merely a passive storage system but an active process that involves the integration of various cognitive functions. For instance, the ability to recall specific events is influenced by emotional states, as emotional content can enhance memory retention and retrieval (Auguste et al., 2023; Saive et al., 2014). Sensory modalities, particularly olfaction, have a great influence on EM retrieval. E. g., odors can serve as powerful cues for recalling autobiographical memories, suggesting that sensory experiences are intricately linked to the EM system (Grabbe, 2010). This relationship emphasizes the multimodal nature of memory, where different sensory inputs can enhance the richness and vividness of recalled experiences.

2.1.5. Non-Declarative Long-Term Memory Model

Non-declarative memory encompasses all information that is not consciously accessible, enabling learning and performance without the need for conscious awareness. This form of memory can, according to Squire (1986), be further subdivided into priming, classical conditioning, non-associative learning, and procedural memory.

Priming refers to the process by which exposure to a stimulus enhances the ability to retrieve information from long-term memory when presented with a related stimulus. This effect arises from the associative nature of memory, allowing connections to be formed regardless of the representational format. Priming can be further categorized into positive priming (facilitating response), negative priming (inhibiting response), semantic priming (connections based on meaning), perceptual priming (based on sensory features), and conceptual priming (involving conceptual information).

Classical conditioning involves forming associations between two stimuli, leading to a learned response. A well-known example is Pavlov's dog experiment (Coon and Mitterer, 2018), in which the ringing of a dinner bell (neutral stimulus) was paired repeatedly with the presentation of food (unconditioned stimulus), ultimately causing the dog to salivate (conditioned response) at the sound of the bell alone. This associative learning process demonstrates how neutral stimuli can acquire significance through repeated pairings.

Non-associative learning represents the simplest form of learning, where behavioral responses change over time in reaction to a single stimulus, without the need for stimulus association (Ioannou and Anastassiou-Hadjicharalambous, 2018). It encompasses two primary mechanisms: (1) Habituation, where repeated exposure to a stimulus gradually reduces responsiveness. The extent of habituation depends on factors such as the frequency, intensity, and duration of the stimulus, as well as the agent's familiarity with it. (2) Sensitization, in contrast, enhances responsiveness to a stimulus following a strong or noxious event. Unlike habituation, sensitization does not require repeated exposure; a single intense stimulus can elicit a heightened response. For example, in addiction relapse, even a brief encounter with drug-related cues can trigger strong cravings and compulsive behaviors. Procedural memory is responsible for the acquisition, storage, and execution of skills and habits, often without

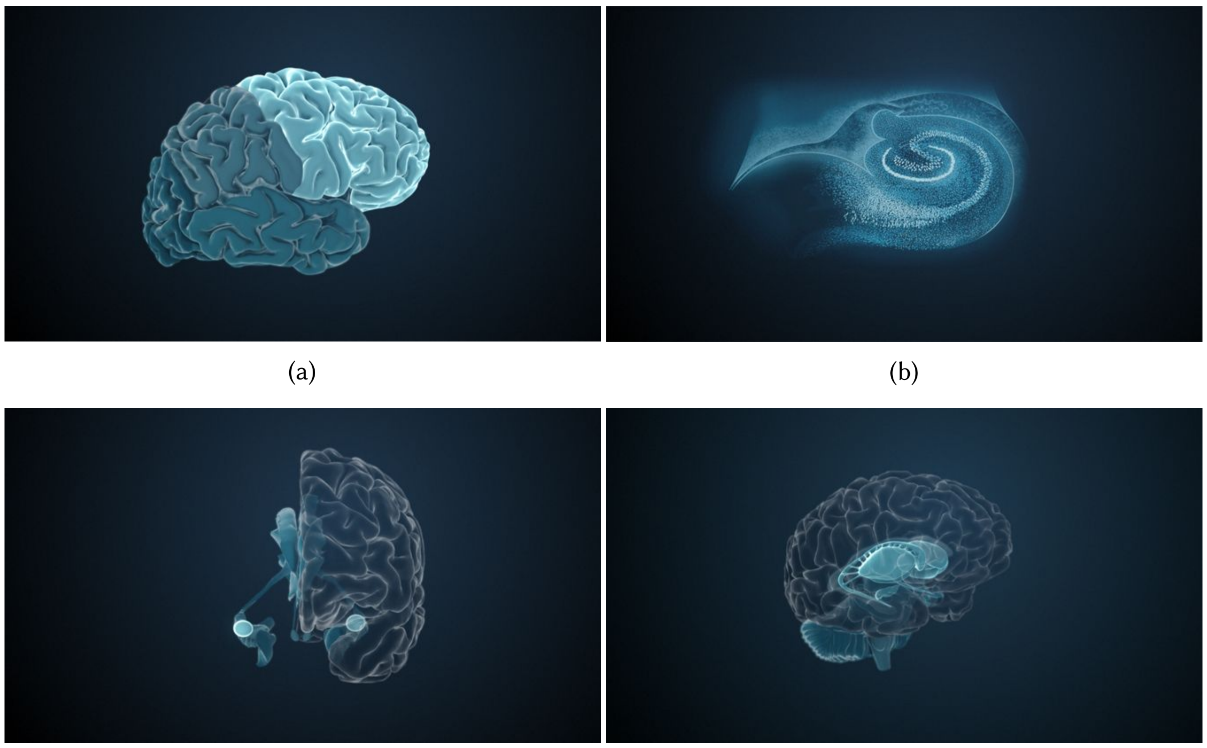

(a) (b) (c) (d)

Figure 2.2.: Key brain regions involved in memory processing. Figure 2.2(a) depicts the frontal lobe in which the Prefrontal Cortex (PFC) is located. The PFC is essential for working memory, decision-making, and cognitive control. Figure 2.2(b) illustrates the HPC. It plays a crucial role in forming, organizing, and retrieving new memories. Figure 2.2(c) shows the Amygdala (AMY), which is involved in emotional memory, particularly in processing fear and strong emotions. Figure 2.2(d) pictures the Basal Ganglia (BG), which contribute to procedural memory and habit formation, helping to automate repetitive tasks. ©www.dasgehirn.info (NWG, 2025)

conscious awareness. It involves a gradual process of practice and refinement, engaging multiple brain regions, including the basal ganglia and cerebellum. Despite its importance for skilled behaviors, the mechanisms by which procedural memories are encoded and retrieved remain incompletely understood.

2.2. Neuroscientific View on Memory

Memory is not only a cognitive phenomenon but also a biological process rooted in the intricate workings of the brain. The neuroscientific anatomy of memory in the brain is a complex interplay of various structures and processes that facilitate the encoding, storage, and retrieval of information. Understanding these mechanisms not only sheds light on the fundamental nature of memory but also has implications for addressing memory-related disorders and enhancing cognitive functions. This section explores the neuroscientific foundations of memory by examining its distinct stages and systems. It begins with sensory memory, the brief and immediate retention of perceptual inputs that forms the gateway to all further memory processing. The discussion then shifts to WM, the brain's dynamic workspace for holding and manipulating information, supported by regions like the Prefrontal Cortex (PFC). Finally, it delves into LTM, the vast and enduring store of knowledge and experiences, highlighting the roles of the HPC, neocortex, and other structures in consolidating and retrieving memories. Figure 2.2 depicts the important regions of human memory which interplay during the encoding, storage, and retrieval of information.

2.2.1. Processing Perceptions and Capturing the Immediate

Human sensory memory is a critical component of cognitive processing, serving as a brief repository for sensory information that allows for immediate recall and decision-making. Neuroscientific research has enlightened various aspects of sensory memory, particularly its neural underpinnings, the role of different brain regions, and the mechanisms involved in encoding and retrieval. Sensory memory is characterized by its fleeting nature, typically lasting only a few seconds. This transient storage allows individuals to hold onto sensory information long enough to process it and make decisions based on that information. Ede and Nobre (2022) emphasize that decision-making often involves sampling from both current sensory input and memory, highlighting the interplay between sensory memory and cognitive functions. This interplay is further supported by findings from Draschkow et al. (2021), who note that in naturalistic settings, individuals can choose to either revisit objects of interest or rely on their memory, indicating that sensory memory is integral to behavioral decisions.

The neural basis of sensory memory is distributed across several brain regions. For instance, the *lateral occipital cortex* is crucial for short-term visual memory, as it supports the identification and categorization of visual stimuli (Jaime et al., 2014). The hippocampal and parahippocampal regions are also vital for retrieving contextual information, which is essential for LTM formation. Moreover, studies have shown that primary sensory cortices are involved in working memory tasks, suggesting that these areas not only process sensory information but also play a role in memory encoding and retrieval (Lemus et al., 2009).

The integration of sensory memory with higher cognitive functions is evident in the way sensory representations are processed in the brain. For example, Slotnick (2004) highlights that visual memory and perception share common neural substrates, suggesting that the same brain regions are activated during both processes. This overlap indicates that sensory memory is not isolated but – in some sense next to WM – interacts with other cognitive functions, facilitating a more comprehensive understanding of sensory experiences.

2.2.2. The Brain's Dynamic Workspace

WM is a crucial cognitive system that enables the temporary storage and manipulation of information necessary for various complex tasks, such as reasoning, decision-making, and learning. Neuroscientific research has made significant strides in understanding the neural mechanisms underlying WM, revealing the roles of specific brain regions, neuronal activity patterns, and synaptic processes.

One of the primary regions involved in WM is the PFC, which is essential for maintaining and manipulating information over short periods. Persistent neural activity in the PFC has been identified as a key mechanism supporting WM. This activity allows neurons to remain active even after the initial stimulus has disappeared, thereby providing a neural correlate for the retention of information (Riley and Constantinidis, 2016). Studies have shown that the strength of functional connectivity between the PFC and other regions, such as the Mediodorsal Thalamus (MD), correlates with memory load during WM tasks, indicating a cooperative involvement of these areas (Oyama et al., 2021).

In addition to the PFC, the HPC, Basal Ganglia (BG), and Thalamus (THAL) also play roles in the encoding and retrieval of information, suggesting a network of interconnected areas that support WM functions (Funahashi, 2013; Manohar et al., 2019). For instance, the *thalamic mediodorsal nucleus* has been shown to participate in spatial WM processes. Moreover, the Medial Temporal Lobe (MTL) has been implicated in managing the workload of WM, with neuronal firing patterns reflecting the complexity of the information being processed (Boran et al., 2020).

The concept of limited WM capacity has also been explored from an evolutionary perspective. Ma et al. (2021) suggest that the constraints of human WM may have evolved to optimize cooperation among individuals, as managing limited information can enhance social interactions and reciprocal behaviors. The coding of information in WM is not solely based on the firing rates of individual neurons but also involves the collective activity of neuronal populations. This population coding allows for the representation of various stimulus attributes, which can be dynamically adjusted based on the demands of the task (Barak et al., 2010).

Furthermore, the dynamic nature of Working Memory (WM) is evident in how it adapts to different contexts and tasks. Recent studies have shown that WM can exist in both focused and unfocused states, where neuronal firing patterns can either sustain activity or remain silent while still retaining information (Wang et al., 2023). This flexibility allows the brain to manage resources efficiently, adapting to the varying demands of cognitive tasks.

2.2.3. Enduring Storage of Memories

Human long-term memory is a complex and multifaceted cognitive function that allows individuals to store and retrieve information over extended periods, ranging from days to decades. The neuroscientific understanding of LTM encompasses various aspects, including the roles of specific brain regions, the molecular mechanisms involved in memory consolidation, and the interplay between different types of memory systems.

The HPC is widely recognized as a critical structure for the formation and consolidation of long-term memories. It is particularly important for declarative memory, which involves the conscious recollection of facts and events (Squire, 2017). Research indicates that the HPC temporarily stores *memory traces* – the physical and biochemical changes in the brain that represent the storage of information – before they are gradually transferred to cortical areas for long-term storage, a process known as systems consolidation (Mizuno et al., 2012). This transfer is essential for the stability of memories, as it allows for the integration of new information with existing knowledge. Studies have shown that the integrity of the HPC and its connections with the PFC is vital for effective LTM performance, especially when the information is complex or abstract (Cohen, 2011).

Crucial for LTM consolidation are specific molecular mechanisms, such as Brain-derived Neurotrophic Factor (BDNF), which facilitates synaptic plasticity, a fundamental process that underlies learning and memory (Bekinschtein et al., 2007). For instance, the activation of BDNF signaling pathways has been shown to enhance long-term potentiation, a process that strengthens synaptic connections (Wittmann et al., 2005). Furthermore, the expression of specific proteins, such as Cyclic AMP Response Element-binding Protein (CREB), is necessary for the consolidation of long-term memories, as demonstrated by studies showing that CREB knockdown impairs memory retention (Brightwell et al., 2005, 2007).

Episodic memory is characterized by the ability to recall specific events, experiences, and their contextual details, including time and place. The HPC is central to the encoding and retrieval of episodic memories. It plays a crucial role in linking various aspects of an experience, such as the who, what, where, and when, into a coherent memory trace (Sols et al., 2017). Research has shown that the HPC exhibits rapid neural reinstatement of prior events following event boundaries, which enhances the representation of these experiences in LTM (Sols et al., 2017). This reinstatement process is thought to facilitate the integration of new information with existing memory networks, a critical aspect of neuroplasticity – the process of synaptic and structural changes in the brain. Such neuroplastic changes associated with EM are not static; they evolve with experience and learning. For instance, the development of EM during childhood is linked to changes in the HPC and its connections to the PFC, which matures over time (Ghetti and Bunge, 2012). This developmental trajectory suggests that neuroplasticity is a lifelong process, with the brain continuously adapting to new experiences and information.

The role of sleep in LTM consolidation is another important aspect of LTM. Research has shown that sleep facilitates the transformation of newly acquired memories into stable long-term representations. During sleep, particularly during slow-wave sleep, the brain replays and reorganizes memory traces, which enhances their integration into existing knowledge networks (Gais et al., 2007; Takashima et al., 2006). This process is thought to involve the HPC and neocortex, with the HPC playing a crucial role in the initial encoding and the neocortex being involved in the long-term storage of memories.

Finally, the interaction between emotional states and LTM formation is significant. The dopaminergic system, particularly the activity of midbrain dopaminergic neurons and their interaction with the amygdala, has been implicated in enhancing memory formation, especially for emotionally charged experiences (Wittmann et al., 2005). This motivates that emotional arousal can modulate the strength and persistence of long-term memories, making them more vivid and accessible.

2.3. Computational View on Memory

The computational view of memory explores how memory processes can be modeled and implemented in artificial systems, offering insights into both human cognition and machine intelligence. The primary objective of such systems is to exhibit intelligent behavior akin to human cognition (Lieto et al., 2018).

This section examines how memory is conceptualized within such systems, named *[artificial] cognitive architectures*. By simulating memory as an information-processing system, computational models bridge neuroscience, cognitive psychology, and artificial intelligence. This enables researchers to study memory in dynamic and scalable ways.

Cognitive architectures are characterized by their ability to integrate various core cognitive processes into a cohesive framework. These architectures often draw inspiration from cognitive psychology and neuroscience, reflecting the intricate nature of human cognition and the need for robust models that can replicate these processes in machines. Kotseruba and Tsotsos (2018) shows, that almost all cognitive architectures of the last 40 years make use of some sort of memory. This insight can be seen as proof that memory plays a crucial role in cognitive architectures, particularly in how these systems emulate human-like cognitive processes. The authors additionally state that episodic memory remains underrepresented in computational models of cognition, however, the integration of EM into cognitive architectures can enhance their ability to process and recall experiences, thereby improving their overall functionality. Additionally, non-declarative memory is often only explicitly implemented as procedural memory, with a focus on replicating learned behaviors and skills. However, some research explore how artificial systems can incorporate conditioning mechanisms to enable adaptive, socially intelligent behaviors (Novianto et al., 2014).

The first subsection investigates traditional rule-based systems that model memory as structured and symbolic, inspired by early cognitive theories, such as the architecture Connectionist Learning with Adaptive Rule Induction On-line (CLARION). Section 2.3.2 focuses on adaptive, network-based systems where memory arises from self-organization and learning. Finally, we explore integrative models that combine symbolic reasoning and emergent learning to reflect the complexity of human memory, as done in State, Operator Apply Result (Soar). Together, these perspectives highlight how computational approaches advance our understanding of memory, and EM specifically with its applications in artificial intelligence.

2.3.1. Cognitivist Cognitive Architectures

Cognitivist cognitive architectures rely on symbolic representations and rule-based processing to model cognitive functions. This method of representing information is both natural and intuitive, often yielding high performance by abstracting away irrelevant data. Symbols are particularly well-suited for expressing descriptive problem statements and actions, as they can be easily augmented with probabilities and beliefs. Consequently, symbolic representations are commonly employed for complex high-level cognitive tasks such as planning, reasoning, and language comprehension. At a semantic level, they facilitate learning from experience. Cognitivist agents operating in real-world environments must employ rules to derive symbols from sub-symbolic sensory input, thereby addressing the *signal-to-symbol gap* (Krüger et al., 2011).

Cognitivist architectures typically distinguish between different types of memory, including working memory, LTM, and specific forms such as semantic and episodic memory. WM in cognitivist cognitive architectures is essential for temporarily holding and manipulating information necessary for ongoing tasks.

E. g., Executive-Process/Interactive Control (EPIC) (Kieras and Meyer, 1997) implements this paradigm. A key objective of EPIC is to model cognitive executive processes, with an emphasis on the precise timing of human perceptual, cognitive, and motor activities in multitasking scenarios. To achieve this, EPIC relies on *Sensory* and *Perceptual Processors* to transform sensory inputs into symbolically coded representations of changes in sensory properties. These processors handle visual, auditory, and tactile inputs. EPIC's working memory consists of modality-specific items and perceptually unrelated information, such as goals and actions, and is updated periodically. Long-term knowledge is represented through production rules.

Related to EPIC, ICARUS (Langley and Choi, 2006) is an architecture that focuses on problem-solving. In each recognize-act cycle, perceptual information is converted into short-term beliefs using categorization and inference and stored in the WM. Such beliefs are used to find an action sequence from the current state to a goal state. Again, production rules to abstract from sensory input to beliefs are stored in the LTM, along with current goals and known actions. While EPIC has a one-way connection from LTM to WM, and thus is not able to learn new production rules, ICARUS is able to learn the connection between skill execution and goal achievements in the form of new skills. Further, it is able to update skill constraints from failure executions.

In architectures like Soar (Laird et al., 1987), WM is implemented as a dynamic structure that allows for the active processing of information, enabling the system to make decisions based on current goals and available data. LTM, which includes both semantic and episodic memory, is vital for the retention of knowledge over extended periods. Semantic memory refers to the storage of factual information and concepts, while EM involves the recollection of personal experiences and specific events. In architectures such as CLARION (Sun, 2007), these memory types are integrated to support learning and adaptation, allowing the system to generalize knowledge from past experiences to new situations. For instance, EM can be utilized to recall specific events that inform future decisions, while semantic memory provides a broader context for understanding and interpreting new information.

In summary, the realization of these memory types varies across different architectures. Some systems utilize snapshots of experiences, storing specific instances that can be recalled when needed. This approach is evident in architectures that implement episodic-like memory, where the system retains detailed records of past interactions to inform future actions (Kasap and Magnenat-Thalmann, 2010; Stachowicz and Kruijff, 2012). Other systems may adopt a more generalized approach, abstracting knowledge from multiple experiences to form broader concepts and rules that guide behavior (Liu et al., 2017). Since cognitive architectures usually evolve, as they are complex software systems that sometimes are under development for decades, some have been reconsidered or have altered their methods of storing knowledge over time. As an example, early versions of Soar follow a snapshot-based approach without generalization of knowledge and learning techniques. By that time, Soar got classified as cognitivist architecture. Newer versions of Soar (Laird, 2019, 2022b) implement EM as well as generalization, abstraction and learning, which is why it is nowadays classified as a hybrid architecture.

2.3.2. Emergent Cognitive Architectures

Symbolic representations may also have restrictions. Cognitivist architectures that have a fixed and predefined set of production rules to generate symbols from perception are less flexible and robust against a constantly changing environment (Kotseruba and Tsotsos, 2018) as new production rules are not learned during operation. Unlike traditional cognitive architectures that often rely on predefined rules and structures, emergent architectures focus on how cognitive processes can arise from the interactions of simpler elements within a system, akin to how complex behaviors emerge in biological systems (Vernon et al., 2010; Ye et al., 2018) – typically by exploiting highly parallel models based on connectionism. Emergent architectures utilize self-organizing systems as a foundational concept, emphasizing the importance of learning from experience. The concept of *embodiment*, which posits that cognitive processes are deeply rooted in the physical interactions of an agent with its environment, is considered fundamental for emergent learning (Vernon et al., 2010). This approach contrasts with traditional architectures that may rely on static representations of knowledge and behavior.

Memory in emergent systems usually follows their self-organizing philosophy to arise naturally from interactions within neural or agent-based networks, such as artificial neural networks (McCulloch and Pitts, 1943), spiking neural networks (Malcolm and Casco-Rodriguez, 2023), or self-organizing maps (Kaski, 2010). The focus on learning and development of these architectures naturally prioritizes a strong focus on EM and WM. Unfortunately, implementations of such systems often lack transparency due to the underlying learning process. Utilizing prior or explicit semantic knowledge is not intended by the architecture.

The Self-Aware Self-Effecting (SASE) (Weng, 2002) emergent cognitive architecture is founded on the principle that intelligence arises from the interactions and connections among simple, low-level elements rather than from a centralized, pre-programmed control system. In SASE, the fundamental units of intelligence are simple, self-aware agents capable of sensing and acting within their environment while forming complex networks of interactions and relationships. Through collective operation, these agents generate complex and adaptive behaviors without relying on centralized control. Emergence in SASE results from the integration of self-awareness, self-effectuation, and interaction. Self-awareness enables an agent to perceive and represent its own state, allowing it to adapt its behavior accordingly. Self-effectuation empowers an agent to act upon its own state, directly influencing its behavior. Interaction facilitates the exchange of information and coordination among agents, fostering emergent phenomena such as cooperative and competitive behavior, as well as the development of higher-level structures and patterns.

Rothfuss et al. (2018) introduces the *Deep Episodic Memory*, a system where knowledge is stored in distributed units in a convolutional long short-term memory (Hochreiter and Schmidhuber, 1997) with each memory encoded as a pattern of activation. Thus, memory formation relies on learning algorithms, such as backpropagation (Kelley, 1960), Hebbian learning (Hebb, 2005), or reinforcement learning (Sutton and Barto, 2018). Specifically, the authors employ a Variational Auto-encoder (VAE) (Kingma and Welling, 2013) in order to convert visual experiences into a latent representation that facilitates encoding, recalling, and predicting sensorimotor experience using unsupervised learning. Through the utilization of multiple decoders, the learned representation can be reconstructed and predicted. Such a learned representation is beneficial as it generalizes and compresses knowledge.

Some emergent architectures develop hierarchical layers of memory, where lower layers capture specific features or timespans and higher layers represent more abstract concepts or contextual knowledge (Bärmann et al., 2024b). Cowell et al. (2019) emphasize the importance of a continuous hierarchy of memory representations, suggesting that different types of memory interact across various timescales, ultimately leading to a more integrated cognitive function.

Just as symbol processing is applied to all cognitive processes in cognitivist architectures, the self-organising approach is not limited to memory in emergent systems. For instance, the attention-gating mechanism described by Elshaw et al. (2010) allows for selective processing of auditory signals, which is vital for maintaining relevant information in WM. This mechanism is complemented by recurrent self-organizing maps that facilitate the development of emergent representations, demonstrating how WM can adaptively manage information. However, this does not necessarily mean that emergent methods are not subject to the same limitations as neural learning models, such as poor extensibility. E. g., Huang et al. (2024) illustrates how EM can interact with consciousness within cognitive architectures, suggesting that specialized modules can be incrementally integrated to enhance cognitive capabilities.

2.3.3. Hybrid Cognitive Architectures

Hybrid cognitive architectures combine elements of both cognitivist and emergent approaches, allowing for greater flexibility and adaptability in cognitive tasks (Vernon, 2014a). These architectures leverage the strengths of both symbolic and subsymbolic processing, enabling robots to utilize memory systems that can handle both structured knowledge and experiential learning (Lieto et al., 2018). For instance, the Soar cognitive architecture exemplifies this integration by allowing for structured problem-solving through symbolic representations while also facilitating learning through sub-symbolic mechanisms (Laird, 2019; Laird et al., 2017; Ye et al., 2018). This dual approach enables systems to manage complex tasks that require both high-level reasoning and low-level sensory processing, thereby enhancing their adaptability and efficiency in real-world applications. Hybrid cognitive architectures are increasingly being applied in developmental robotics, where they facilitate the learning of complex behaviors through interaction with humans and the environment (Kotseruba and Tsotsos, 2018).

We already explained that modern versions of Soar are rather classified as hybrid cognitive architectures. Still, the goal of the Soar project is to develop an artificial system that has similar cognitive capabilities as humans, i. e., knowledge-intensive reasoning, reactive execution, hierarchical reasoning, planning, and learning from experience, and to find out what computational structures are required to support human-level agents. Beyond WM, Soar manages three different types of long-term memories: (1) A procedural memory that contains skills as if-then-rules, (2) a semantic memory that contains facts and declarative information about tasks, and (3) an episodic memory that manages experiences consolidated from the WM in form of snapshots. Laird (2019) emphasizes the importance of EM in the Soar cognitive architecture, where it supports the learning and recalling of prior events and situations. The integration of semantic memory with EM enables a more comprehensive understanding of context, as agents can draw upon both specific experiences and general knowledge when making decisions (Wood et al., 2012). Next to symbolic representations, Soar includes sub-symbolic processing, i. e., to generate symbols from sub-symbolic percepts or to control symbolic processing (Laird, 2008). Soar has multiple learning mechanisms for different types of knowledge: Chunking and reinforcement learning acquire procedural knowledge, whereas episodic and semantic learning acquire the corresponding types of declarative knowledge.

The Intelligent Soft Arm Control (ISAC) hybrid cognitive architecture (Kawamura et al., 2008) is composed of an integrated collection of software agents and associated memory systems. These software agents encapsulate all aspects of perception, cognition, and action, operating asynchronously to process information efficiently. Perceptual events, encoded via a Sensory Ego-sphere (SES) (Peters et al., 2001), are initially placed in short-term memory. An attentional network then evaluates their relevance to the current context and forwards pertinent information to working memory. In addition to storing relevant perceptual information, WM temporarily maintains data related to motivation, goals, actions, and internal processes as required for ongoing tasks. It also encapsulates expectations about future states, simulated by a *Central Executive Agent*. Long-term memory stores procedural, semantic, and episodic knowledge, which consists of abstractions of SES enriched with goal-directed actions, outcomes, and valuations. This multi-layered structure enables efficient memory retrieval. Furthermore, associations are represented as state transitions within episodic memory, supporting learning and adaptive behavior.

Another type of memory, a key component usually present in embodied systems, is procedural memory, which involves the knowledge of how to perform actions. This type of memory may be realized through learned models such as reinforcement learning mechanisms, where agents learn to optimize their actions based on feedback from their environment (Laird, 2009) or through fixed pre-programmed procedures (Peller-Konrad et al., 2023), e. g., in the form of symbols mapping to executable statecharts that can be instantiated and executed if needed (Wächter et al., 2018). The role of procedural memory in hybrid cognitive architectures is sometimes enhanced by the incorporation of internal simulations. These simulations allow agents to mentally rehearse actions and predict outcomes based on stored memories, thereby improving their decision-making capabilities (Shanahan, 2006). This aspect is particularly relevant in scenarios where agents must anticipate the consequences of their actions before executing them, thereby reducing the likelihood of errors and enhancing overall performance.

Learning Intelligent Decision Agent (LIDA) was developed as a biologically inspired cognitive architecture to model all aspects of cognition in the form of a global cognitive cycle (Franklin et al., 2013). It includes a large number of cognitive modules, some of which have short-term or long-term storage capabilities. Its cognitive cycle is divided into three phases: a perception and understanding phase, an attention phase, and an action and learning phase. During the perception and understanding phase, sub-symbolic data from the environment is analyzed and translated into symbols corresponding to objects, entities, or events in the *Perceptual Associative Memory* module. A *Current Situational Model* holds information about an agent's present situation, enriched through the recall of experiences from long-term memory modules using local associations or similarity measures. Information about the present may decay if not stored in long-term memory. During the attention phase, the content of the *Current Situational Model* is surveyed, and the most salient information is broadcast to all modules. During the action and learning phase, the modules use the broadcasted information for learning and execution. For example, the *Procedural Memory Module* instantiates behaviors in the form of symbols that can then be executed.

Another system is Cognitive Robot Abstract Machine (CRAM) (Beetz et al., 2010), which integrates perceptual information, semantic knowledge from multiple knowledge bases, and execution results of simulated actions to perform vaguely defined, goal-directed everyday activities. To abstract execution plans for such tasks, CRAM employs *designators* (i. e., placeholders) that require resolution at runtime. Once available, these placeholders are populated with knowledge from the internal knowledge base (Beetz et al., 2018). This system contains a large-scale ontology of symbolic information, enabling reasoning and generalization. To incorporate non-symbolic information, computable predicates are utilized. Episodic knowledge is represented as Narrative-Enabled Episodic Memories (NEEMs), a first-order time interval logic expression enriched with detailed episodic low-level information, such as perceptual or procedural events and signals. Sub-symbolic information is integrated via a logical interface using computational predicates, inherently grounding resulting representations and ensuring consistency with the environment. The chosen data structure enables the inspection of stored information for learning, reasoning, action simulation, and feasibility evaluation. Beyond acquiring knowledge from real-world experiences, episodic knowledge can also be generated or refined by simulating actions in an inner-world model, which includes a high-quality virtual reality system and physics engine. Generalization and specialization are achieved through meta-cognitive induction.

2.4. Forgetting

Although largely omitted in this work, memory is deeply interconnected with lots of cognitive processes, serving as a foundation for various mental functions. Its integration with attention, prediction, emotion, problem-solving, reasoning, and decision-making illustrates its central role in cognition. In the following, we focus on the process of forgetting, with emphasis on decay, interference, retrieval failure, and neural mechanisms that contribute to the loss or alteration of stored information.

2.4.1. Forgetting in Human Memory

Cognitive psychologists propose that forgetting occurs due to the natural decay of memory traces over time if they are not rehearsed or retrieved. This could happen during the consolidation of information from their senses into WM and from working into LTM (Baddeley, 2006). The longer information remains unused, the weaker the neural connections associated with that memory. The work of Hermann Ebbinghaus introduced the forgetting curve, demonstrating that forgetting occurs rapidly shortly after learning, with the rate of forgetting slowing down over time (Murre and Dros, 2015). This curve has been replicated and analyzed in various contexts, revealing that forgetting is not merely a linear process but can be influenced by multiple factors, including the nature of the material learned and the conditions under which it is recalled (Georgiou et al., 2023; Murre and Chessa, 2011).

Others expanded on Ebbinghaus's initial findings by exploring the mechanisms underlying forgetting. For instance, interference-based forgetting occurs when new information disrupts the retrieval of older memories, e. g., because of mental exertion. This can be categorized into proactive interference, where older memories hinder the recall of newer ones, and retroactive interference, where new information interferes with the retrieval of previously learned material (Wixted, 2004), analogously to a limited transfer bandwidth in artificial systems or because of cue overloading, where highly similar bits of information will be stored in the same location and therefore interfere with each other. This phenomenon highlights the complex interplay between memory retrieval and forgetting, indicating that forgetting can serve a functional purpose by allowing individuals to prioritize relevant information over less pertinent details (Maxcey et al., 2020). Research indicates that memories relying on recollection are primarily forgotten due to decay, while those based on familiarity are more susceptible to interference from irrelevant information (Sun et al., 2018).

Retrieval failure is another significant contributor to forgetting. This theory posits that memories may still exist in the brain but are inaccessible due to a lack of appropriate retrieval cues. For instance, studies have shown that retrieval failures can occur even when the original memory trace remains intact, suggesting that forgetting may not always be a result of decay or loss but rather a failure to access the stored information (Ageno and Iramina, 2024).

Psychological mechanisms often lead individuals to intentionally forget distressing or unwanted memories. This process often involves inhibitory control, where the PFC actively suppresses the retrieval of certain memories, particularly those associated with trauma or negative experiences (Anderson and Hanslmayr, 2014; Madore et al., 2020). Unmotivated forgetting of traumatic events may be due to encoding failures, where information is not adequately processed during the initial learning phase, leading to ineffective storage. Finally, consolidation failures, i. e., disruptions of the processes that stabilize a memory trace after initial acquisition, can lead to fragmented memories (Levine, 2015).

This motivates that forgetting is not merely a passive process but an active one that involves various brain regions. In humans, the PFC is implicated in the suppression of unwanted memories, while the HPC is essential for the retrieval of contextual information. Additionally, intrinsic forgetting may be a default state of the brain, suggesting that memory erasure is a natural part of memory management (Herz et al., 2022). Forgetting is not always a failure of the memory system but can serve an adaptive purpose. Both cognitive psychology and neuroscience highlight that forgetting can enhance efficiency by removing unnecessary or outdated information and help in emotional regulation by allowing individuals to move on from traumatic experiences. This multidimensional understanding underscores the dynamic and purposeful nature of forgetting in human memory.

2.4.2. Forgetting in Artificial Memory

From a computational perspective, forgetting is modeled as a process of information loss or reduced accessibility. As mentioned before, forgetting is not seen as a flaw but rather as a functional and adaptive feature that optimizes the storage and retrieval of relevant information. If all experiences are transferred from WM to LTM, this would overtax the bandwidth requirements. To address these issues, artificial cognitive architectures can leverage forgetting mechanisms inspired by similar processes in human memory.

Cognitive architectures in state-of-the-art literature use mechanisms that can mostly be divided into two categories: time-based decay methods and methods based on retroactive interference which model similarity/cue-overload effects in humans (Plewnia et al., 2024).

Time-based decay mechanisms - the more prominent type of the two - can be found in different forms. All forms calculate an initial memory strength value while consolidating the memory for the first time. This value then decreases over time and is later, upon retrieval or periodically, checked against a threshold. E. g., Anderson et al. (2004) simulates decay by decreasing memory activation logarithmically with time since the last retrieval. Factors, such as emotions (Bui et al., 2019), mental exertion (Freedman and Adams, 2011), or past retrieval times (Freedman and Adams, 2011), may alter the initial strength of the memory. Additional factors may also depend on the content of the memory, therefore causing different memory types to potentially decay at different rates. Some approaches highlight content independence by using similar factors for all memories.

The second type of forgetting mechanism found in artificial cognitive architectures is based on retroactive interference. These mechanisms assume that available storage space is limited, which is a valid assumption in artificial cognitive architectures and human WM (Kotseruba and Tsotsos, 2018). The mechanisms work by replacing memory snapshots with similar, but newer ones (Scheutz et al., 2019) or altering memory strength values based on mental exertion (Freedman and Adams, 2011). Similarity-based forgetting has the downside of requiring knowledge about the content of a memory snapshot to decide on forgetting or consolidation. The two types of mechanisms can also be combined as described in Freedman and Adams (2011) and Plewnia et al. (2024).

2.5. Rethinking Artificial Memory

In recent years, the understanding of memory, particularly EM, has undergone significant scrutiny and reevaluation. Wood et al. (2012) highlights the complexities and nuances of memory systems, urging a reconsideration of traditional views on artificial memory. Traditionally, memory has been viewed as a passive storage device, a centralized system, and a straightforward repository of knowledge. However, Wood suggests that these assumptions are overly simplistic and do not adequately capture the complexities of memory processes.

Recent studies indicate that memory is not merely about storing information but is fundamentally a constructive process. Schacter and Addis (2007) argue that especially episodic memory is inherently constructive, meaning that it involves the integration and alteration of past experiences to create new memories rather than simply retrieving stored information. This perspective aligns with findings from neuropsychological studies that highlight how memory can be distorted and influenced by various factors, including spatio-temporal context and emotional states (Schacter et al., 2008). Many cognitive architectures in the literature do not implement episodic memory. And even if it is implemented, episodic memory is rather reduced to simple storage and recognition tasks rather than genuine recollection, missing its introspective and temporal qualities. However, some architectures partially fulfil the requirements for active and episodic memory. For instance, in Soar, while WM does not evolve with new data, LTM incorporates mechanisms for learning.

However, many cognitive architectures focus heavily on implementing different types of memory without adequately addressing their interconnections (Kieras and Meyer, 1997). This includes the ability to associate entities across different modalities. Often, only associations between homogeneous data types are possible (Kawamura et al., 2008). Without the capacity to store and associate multi-modal information, it becomes challenging for the system to draw unified conclusions from knowledge derived from diverse sources. A few systems attempt to address this issue by using neural networks to associate information across data sources (Rothfuss et al., 2018). However, these associations are often inaccessible to other system components, such as reasoning modules, limiting their practical application.

Additionally, Wood et al. (2012) criticizes the way information is represented in artificial memory systems. Most artificial cognitive architectures rely on specialized representations, managing modality-specific knowledge in distinct, isolated containers (Beetz et al., 2010; Franklin et al., 2013; Kieras and Meyer, 1997; Langley and Choi, 2006). This simplification hinders the integration of new knowledge modalities and reduces the system's overall cognitive coherence (Paulius and Sun, 2019).

Furthermore, the idea of memory as a centralized system is increasingly being questioned. E. g., Vernon (2016) states that cognition, including memory, is distributed and situated within the context of interactions with the environment. This perspective emphasizes that memory is not localized in a single area of the brain but is instead a product of interactions across various neural networks and cognitive processes (Anderson et al., 2004). This shift in understanding suggests that memory systems may operate more like a network of interconnected and distributed processes rather than a centralized hub.

To summarize, memory is not merely a passive storage device or a centralized system; it is a constructive i. e., active, distributed, and multimodal process that is deeply intertwined with our interaction experiences with the world. Building on the requirements identified by Wood et al. (2012) and closely tied to the concept of enactive memory (Vernon et al., 2011), we highlight introspection as another critical component and foundation of memory and data representation. Introspection enables a system to adapt its behavior based on stored information, facilitating processes such as internal simulation, augmentation, and prediction. Furthermore, introspection is deeply connected to the ability to monitor and reflect on the system's internal cognitive processes. While some cognitive architectures focus primarily on perception-action coupling (Langley and Choi, 2006), others explicitly model introspection as a core capability (Beetz et al., 2010; Weng, 2002). This distinction underscores the importance of self-awareness in enhancing a system's adaptability and functionality.

A comparison between some cognitive architectures in terms of the features identified for memory is shown in Table 2.1. A detailed description of the architectures can be found in Section A.1. The table shows that no architecture completely fulfils the characteristics presented, even if some architectures come very close to the ideal. These are exclusively hybrid architectures. This is not surprising, as hybrid architectures pursue the goal of combining emergent learning with efficient reasoning.

Architecture	Paradigm	1	2	3	4	5
EPIC (Kieras and Meyer, 1997)	cognitivistic	✗	✗	✗	✗	✓
ICARUS (Langley and Choi, 2006)	cognitivistic	✓	✗	(✓)	✗	✓
ADAPT (Benjamin et al., 2004)	emergent	✓	(✓)	✓	(✓)	✗
MDB (Bellas et al., 2010)	emergent	✓	(✓)	✓	(✓)	(✓)
deep EM (Bärmann et al., 2021)	emergent	✓	(✓)	✓	(✓)	✗
SASE (Weng, 2002)	hybrid	✓	✓	✓	(✓)	✓
ACT-R (Anderson et al., 2004)	hybrid	✓	(✓)	✓	(✓)	(✓)
ISAC (Kawamura et al., 2008)	hybrid	✓	(✓)	✓	(✓)	(✓)
CRAM (Beetz et al., 2010)	hybrid	✓	✓	✓	(✓)	(✓)
LIDA (Franklin et al., 2013)	hybrid	✓	✓	✓	(✓)	✓
Soar (Laird, 2022b)	hybrid	✓	(✓)	✓	(✓)	✓
ArmarX (Peller-Konrad et al., 2023)	hybrid	✓	✓	✓	✓	✓

Table 2.1.: Comparison of the several cognitive architectures with respect to the identified requirements for a memory system in robotics. 1 = *active*, 2 = *multi-modal*, 3 = *associative*, 4 = *episodic* and 5 = *introspective*. (✓) indicates that a requirement is only partially fulfilled. ©Elsevier Peller-Konrad et al. (2023), modified

CHAPTER 3

The Role of Memory in Cognitive Robotics – An Example

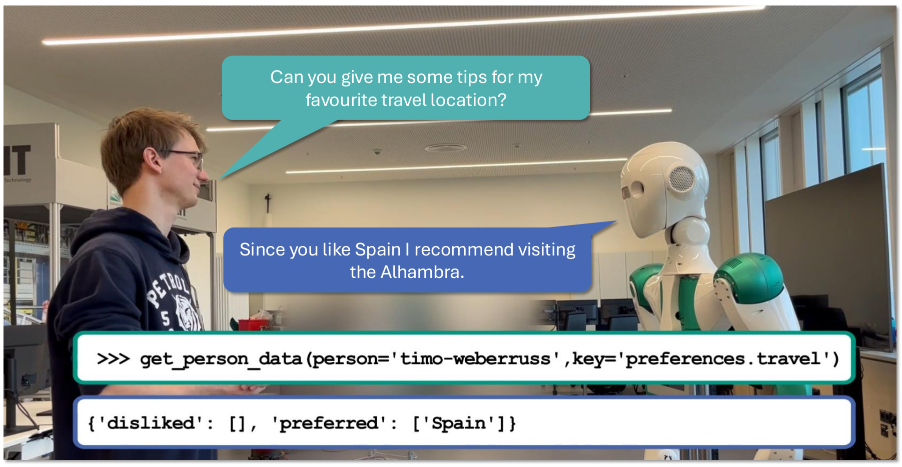
To show the importance of a memory-centred cognitive architecture in robotics, we will examine a typical task for home service robots from different points of view. This home scenario provides a practical, relatable, and challenging environment for evaluating memory systems, given its combination of dynamic conditions, human-robot interactions, and the need for context-sensitive adaptability (Gonzalez-Aguirre et al., 2021).

Service robots are expected to assist residents with daily tasks, adapt to user preferences, and handle complex, multi-step operations. Memory systems play a central role in enabling these capabilities. For instance, a robot's ability to recall past interactions, retain information about the environment, and manage temporary task-relevant data is crucial for effective functioning.

Assume a home consisting of various rooms, appliances, and objects (e. g., kitchen counters, living room tables, refrigerators, and coffee machines). The robot's primary goal is to assist a human resident, referred to as "the user", by performing tasks such as:

- Fetching and delivering objects (e. g., coffee cups, books) or performing other multi-step tasks (e. g., cleaning a table or organizing groceries).
- Remembering the user's preferences (e. g., coffee with milk and sugar).
- Remembering previous experiences for decision-making (e. g., successful and failed interactions) and adapting to dynamic changes in the environment (e. g., a misplaced object).
- Verbalizing past interactions with the user or the environment

We assume a humanoid robot that interacts with the user via speech, gestures, and contextual actions.


Can you give me some tips for my favourite travel location?
Since you like Spain I recommend visiting the Alhambra.

```
>>> get_person_data(person='timo-weberruss',key='preferences.travel')
```

```
{'disliked': [], 'preferred': ['Spain']}
```

Figure 3.1.: The humanoid robot ARMAR-DE (Asfour et al., 2019) reacting to a user request to provide travel recommendations for the user's favorite travel location. The robot knows the user's favorite travel location from a previous interaction.

3.1. Fetching and Delivering Objects

One of the most commonly used scenarios is probably simply pick and place tasks. These tasks involve the robot's ability to identify, grasp, transport, and accurately place objects in designated locations. In addition, the robot may have to remember past object locations, collision-free navigation routes, etc. Such a task usually consists of the following steps:

1. The user verbally instructs the robot ("Please bring me a cup of coffee from the kitchen counter.")
2. The robot then processes the command and encodes it into working memory, identifying the user's request, the object (coffee), and its location (kitchen counter).
3. Using its knowledge of the home layout (e. g., stored in long-term memory), the robot navigates to the kitchen and identifies the object on the counter.
4. After fetching the coffee, the robot delivers it to the user.

The realization of the individual steps and the design of the scenario can be infinitely complicated. The robot could, for instance, simply use preprogrammed knowledge of object locations or use object detection methods to find objects, locations, humans, etc. in its environment. The requested objects may be placed in easily accessible locations, or the robot may have to use additional actions, such as opening and closing doors, to reach the objects. The actions can be as simple as predefined trajectories or parameterized by knowledge, such as the current position of the person during handover.

In the following, extensions to this simplified scenario are described, which emphasize other aspects of memory in the individual steps.

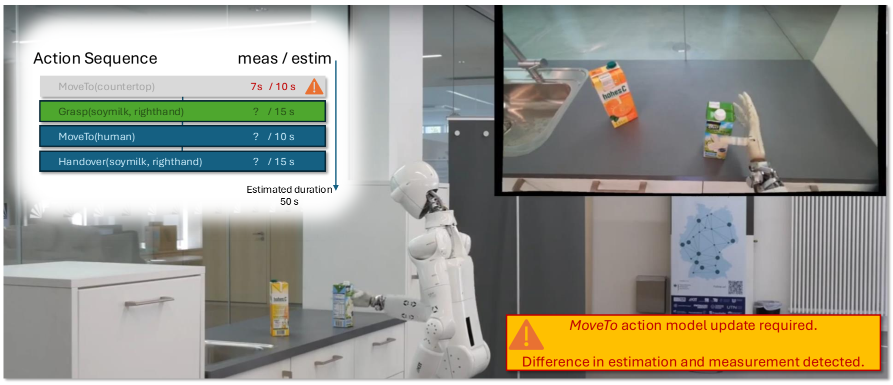

Action Sequence
meas / estim
MoveTo(countertop) 7s / 10 s
Grasp(soymilk, righthand) ? / 15 s
MoveTo(human) ? / 10 s
Handover(soymilk, righthand) ? / 15 s
Estimated duration 50 s
MoveTo action model update required.
Difference in estimation and measurement detected.

Figure 3.2.: The humanoid robot ARMAR-7 is grasping the milk from the kitchen counter. The grasping action is part of a plan where the robot first moves to the counter, grasps the milk, moves to the human, and finally hands the milk over to the human. The moving action to the counter has already been executed, whereby the estimated duration and the measured duration of the action execution differ significantly.

3.2. Remembering the User's Preferences

Remembering a user's preferences is a key capability for a cognitive robot operating in environments like a home, office, or retirement home. This task involves the ability to store, retrieve, and update information about individual users' likes, dislikes, habits, and requirements to personalize the robot's actions and interactions.

Consider the previous task, but this time the instructing person has specific preferences that the robot needs to know, which may be different from other people known to the robot – e. g., the user expects an espresso instead of a default café crema. During the identification of the user's request, the robot has to recognize who is the speaking person, remember the preferences and habits of that person, and use this information for decision-making. Additionally, the robot must be able to learn preferences from user interactions and explicit feedback and to update preferences when new feedback or changes in user behavior are observed. All the aforementioned additional steps and components require working- and long-term memory and introduce new modalities the memory has to manage. The way in which preferences are stored in memory may range from simple symbolic and graph-based representations to implicitly learned neuronal activation patterns (Weberruß et al., 2025).

3.3. Remembering Previous Experiences for Decision-Making and Adapting to Dynamic Changes

This involves a service robot leveraging its memory of past interactions and experiences to make informed decisions or to adjust previously made decisions based on unexpected modifications in its environment or user instructions. Past interactions and experiences may refer to successful strategies, frequently requested tasks, or failures in similar contexts. Unexpected

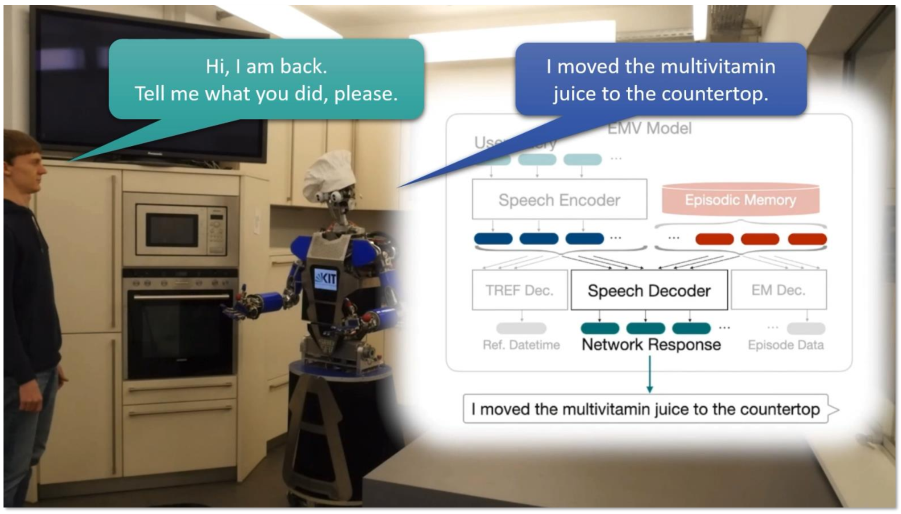


Figure 3.3.: A human commands the humanoid robot ARMAR-III (Asfour et al., 2006) to verbalize its past. The robot processes the human requests, queries its episodic memory, and finally generates and verbalizes its answer.

modifications could include obstacles in the robot's path, last-minute user requests, updated object locations, or changes in task priorities. By analyzing and applying this information, the robot can adapt its actions to better serve the user and handle similar situations with greater efficiency and accuracy.

Imagine the robot being asked to bring the user coffee from the kitchen counter for the first time. The robot successfully navigates to the correct location but is not able to find the cup. This unforeseen interruption triggers the robot to recall past interactions with the object or to query its long-term memory for alternative locations (e. g., it knows that the user often leaves cups on the dining table, or it was last seen in the living room). The robot finally finds the cup, but it grasped the coffee unstably, causing the coffee cup to fall down and resulting in a non-recoverable failure. When instructed again, the robot uses this interaction knowledge, stored in its semantic and/or episodic memory, to find the cup more quickly and to improve grasp candidate generation.

An additional example is the continuous refinement of action models. When the execution of an action fails to align with previously calculated expectations – such as action costs, duration, or probability of success – it becomes essential to analyze the cause of this discrepancy. If necessary, the internal model of the action must be updated accordingly. A specific instance of this process is illustrated in Figure 3.2, where the observed execution time of an action diverges from the anticipated value.

3.4. Verbalizing Past Interactions

Verbalizing past interactions involves a service robot recalling and communicating details about previous interactions with the user or its environment. This ability helps in maintaining context, improving user trust, and enhancing the robot's utility by demonstrating awareness of its history with the user (Bärmann et al., 2021).

After the execution, assume the user asking questions about the location of the cup, how long it took to find it, or if something failed during execution, and if the robot was required to search for alternative strategies, as shown in Figure 3.3. Such queried knowledge could range from simple symbols from semantic memory, over experiences stored in episodic memory to complex movement sequences from procedural memory. Next to the verbalization of knowledge, other externalization techniques, e. g., replaying experiences in a simulation environment (Beetz et al., 2018), are also used in the literature.

CHAPTER 4

The Memory-centric Cognitive Architecture in ArmarX

Disclaimer

Parts of the content presented in this chapter were previously published in

- F. Peller-Konrad, R. Kartmann, C. R. Dreher, A. Meixner, F. Reister, M. Grotz, and T. Asfour (2023). "A memory system of a robot cognitive architecture and its implementation in ArmarX". In: *Robotics and Autonomous Systems* 164, pp. 1–20
- J. Plewnia, F. Peller-Konrad, and T. Asfour (2024). "Forgetting in episodic long-term memory of humanoid robots". In: *IEEE International Conference on Robotics and Automation (ICRA).* Yokohama, Japan, pp. 6711–6717

As motivated by Kotseruba and Tsotsos (2018), in modern cognitive control architectures, memory systems play a crucial role in supporting autonomous robots by providing a structured, efficient, and scalable way to store and retrieve knowledge.

This knowledge often derives from sensory input, internal states, episodic experiences, and prior knowledge. The memory system we developed addresses necessary requirements by offering a multi-modal, distributed memory architecture that meets the demands of cognitive robotics (Peller-Konrad et al., 2023). It uses a specialized knowledge representation framework which fulfills data-related features of required key characteristics of a memory system in cognitive robotics. In this chapter, we present an in-depth overview of our memory implementation as well as its knowledge representation, discussing its key characteristics that led to the design, core components, structure, and operational mechanisms.

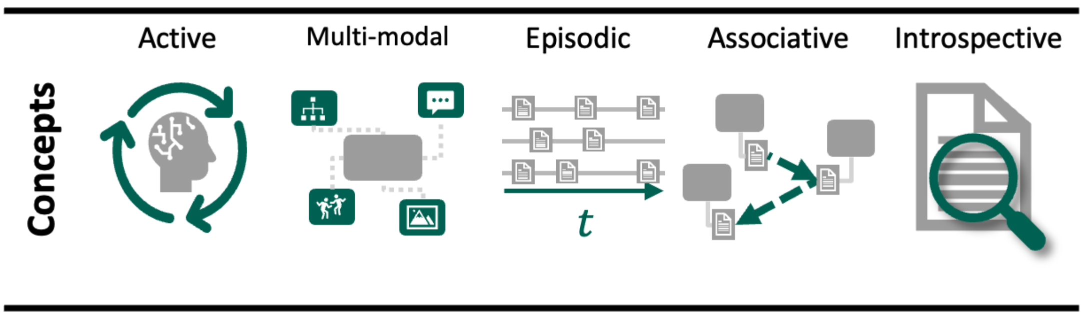


Figure 4.1.: Identified conceptual requirements for a memory system for a robot cognitive architecture. We argue that a memory system must be active, multi-modal, episodic, associative and introspective. These requirements allow high-level cognitive abilities such as explainability, reasonability, or prospection. ©Elsevier (Peller-Konrad et al., 2023)

4.1. Key Characteristics for Memory-based Cognitive Architectures

The implementation of the memory system in the robot software framework ArmarX is guided by several key characteristics that are fundamental to its role within a humanoid robot's cognitive architecture. These characteristics shape the system's design, ensuring it meets the demands of real-time processing, adaptability, and scalability, while maintaining a robust framework for learning and interaction. We believe that the memory system is not just a passive repository but an active, structured component that highly influences the robot's behavior, decision-making, and learning capabilities and performance.

This section motivates these characteristics – divided into conceptual and technical requirements – that have driven the implementation of our memory system, including its multi-modal, distributed, and episodic nature, as well as its access efficiency and scalability. Each of these traits was carefully chosen to address the complex needs of humanoid robotics, such as handling diverse data sources, supporting real-time perception and action, learning and development, and managing vast amounts of sensory information with limited computational resources.

By outlining these, we will show in Section 4.2 how they influenced the architectural decisions and technical approaches. Understanding these foundational principles provides insight into how our memory implementation efficiently supports cognitive tasks such as learning from experience, adapting to new environments, and making informed decisions in real time.

4.1.1. Conceptual Requirements

The identified conceptual requirements of memory, that are necessary to realize core cognitive abilities such as prediction, reasoning or learning from experience (Vernon et al., 2011) are shown in Figure 4.1. we distinguish between the ability to represent multi-modal data, associate knowledge, store episodic knowledge, and inspect knowledge in an active memory system.

Multi-modal Data Representation

The necessity for multi-modal knowledge representation in memory is underscored by the complexity of real-world environments and the diverse forms of information that robots encounter. Multi-modal representation integrates various types of data – such as sub-symbolic data like visual, auditory or tactile inputs; symbolic data, e. g., recognized words; and meta-cognitive and intermediate processing results (Leidner, 2024) – enabling robots to develop a more comprehensive understanding of their surroundings and tasks.

Ilinykh and Dobnik (2021) highlight the advantages of multi-modal architectures over uni-modal ones, noting that integrating semantic information from both language and vision significantly improves the quality of visual representations. Another example: multi-modal representations have been shown to be essential for effective human-robot interaction, as robots can interact better with humans when they can interpret and respond to gestures, speech and other forms of communication in an integrated way (Bremner and Leonards, 2016).

The cognitive processes underlying human perception and decision-making are inherently multi-modal, as humans rely on a combination of sensory inputs to form a coherent understanding of their environment. By adopting a similar approach, robots can emulate efficient human-like reasoning and learning.

Associative Knowledge Representation

The argument for associative knowledge representation in the memory system of a robot cognitive architecture is compelling, as it aligns with the fundamental principles of how both human cognition and robotic systems process information (Wood et al., 2012). Associative memory allows robots to create connections between different pieces of information, facilitating learning, reasoning, and decision-making in complex environments.

As highlighted by Tenorth and Beetz (2013), effective knowledge processing in robots requires the ability to draw connections between various knowledge pieces, enabling them to infer new information and make decisions based on past experiences. This is heavily linked to the aforementioned multi-modal data representation. Only a multi-modal representation supports the coupling of perception and action. which is key for tasks that require the robot to understand context and make decisions in incomplete or ambiguous environments.

Episodic Memory and Time-Series Data

The necessity for robots to learn from past experiences, adapt to new situations, and engage in complex decision-making processes requires an episodic knowledge representation in the memory system of a robot's cognitive architecture.

Liu et al. (2017) highlight the importance of episodic memory in robotics, where the robot records events in real-time and updates its memory to facilitate self-learning. This capability enables the robot to adapt its behavior based on previous experiences, improving its performance over time. In the context of human-robot interaction, Vinanzi et al. (2019) shows that integrating episodic memory into a cognitive architecture enhances a robot's ability to develop trust and theory of mind in human-robot interactions. By remembering past interactions and the context in which they occurred, robots can tailor their responses to users, fostering more meaningful and effective communication. Additionally, the integration of episodic memory can enhance a robot's ability to form a coherent narrative of its experiences, which is essential for self-awareness and autonomous functioning (Hassan and Kopp, 2020).

The ability to organize, recall, and retrieve episodic information, such as personally experienced events occurring at a particular time, place, or context is essential not only for episodic knowledge but also for semantic and procedural knowledge since temporal context often dictates the factual status of knowledge. A clear illustration of this is Pluto's reclassification from a planet to a dwarf planet in 2006, underscoring how seemingly static knowledge can shift over time due to context (Peller-Konrad et al., 2023). Thus, we believe that knowledge should not only be defined universally. It should also be managed in universal, episodic-like data structures, that make it possible to access past knowledge even if it has already been updated. The knowledge, why and when something has been updated in memory may be crucial for learning from experience, and for explainability.

Active Memory System

Active memory allows robots to not only store information but also to utilize, update, and adapt that information in real-time, which is critical for functioning in complex and unpredictable settings. The memory system should not be viewed as a passive storage mechanism but as an active component of the cognitive architecture (Vernon, 2014a; Wood et al., 2012), dynamically shaped by the context in which the robot operates. Erlhagen and Bicho (2006) emphasize the importance of goal-directed behavior in cognitive robotics, where the robot must continuously evaluate its actions and adjust its plans based on sensory feedback and environmental conditions, which requires a memory system that is not only passive but actively involved in processing and integrating information. Rosenthal et al. (2013) highlights the need for an active memory that evolves. It must learn in what context relevant information needs to be easily accessible and can be applied in real-time.

Context, however, not only determines what is encoded, updated or retrieved. It also influences *how* information is encoded. As a real-world example, highly emotional experiences in humans form more durable memories than routine events (Samsonovich, 2013). This shows, that the context – here a highly emotional situation – controls how intense an experience is stored.

Introspective Memory

The argument for incorporating introspective knowledge representation in the memory system of a robot cognitive architecture is crucial for enabling robots to evaluate their own knowledge,

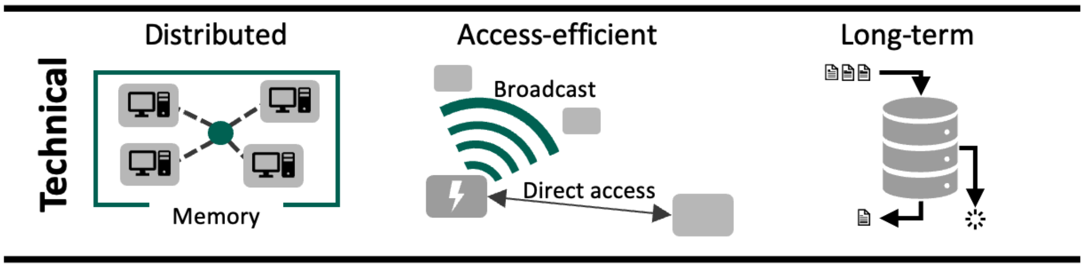


Figure 4.2.: Identified technical requirements for a memory system for a robot cognitive architecture. Due to technical limitations of robot systems, we argue that a memory system must be distributed, efficient and long-term. ©Elsevier (Peller-Konrad et al., 2023)

performance, and decision-making processes. Introspection allows robots to reflect on their internal states and experiences, leading to improved adaptability, learning, and interaction with their environments and human users and is highly correlated to an active memory. Updating models to encode information or to decide what to store, forget or retrieve requires the memory to analyze and to react to its content. The aspect of predicting information (Vernon et al., 2011) additionally requires the ability to "look inside" the data. As noted by Infantino et al. (2013), robots could also recognize knowledge gaps or contradictions through introspection and assume new connections, which is essential for continuous learning and improvement. Similarly, Sun and Hélie (2013) highlight that introspective mechanisms can help robots identify and correct errors in their reasoning and learning strategies. Finally, explainability requires the ability to trace back through decision processes and underlying data, requiring introspection (Tenorth et al., 2010).

In summary, we argue that the conceptual design of a memory system for robot cognitive architectures rests on a foundation of key characteristics essential for managing knowledge. These include the ability to handle multi-modal data, form associative links, and maintain an episodic structure, which supports the flexibility and extensibility of the overall system. The memory is designed to be active, shaping itself according to the robot's context, and introspective, continuously adapting based on the data it processes. Together, these characteristics ensure that the memory system not only serves as a repository but also actively contributes to reasoning, learning, and predicting outcomes in complex, real-world scenarios.

4.1.2. Technical Requirements

The implementation of a memory system for complex robotic architectures also presents several technical challenges. This section outlines these challenges, identifies the corresponding technical requirements, and introduces the necessary design paradigms. The requirements, summarized in Figure 4.2, address three key questions: (1) where to deploy the memory system, (2) how to optimize access efficiency, and (3) how to manage space limitations.

Deployment of the Memory System

Humanoid robots are highly data-intensive, usually relying on multiple specialized computing units, often connected via a common interface like Ethernet, to process sensory inputs and control action execution. Transferring large volumes of data generated across such interfaces can be inefficient, particularly in tasks involving computer vision – one of the largest data sources in such a system. To mitigate throughput and response time issues, a memory system should be designed in a distributed manner, with memory units running as individual processes on the machines where the data is produced. As another benefit, robots with a distributed memory can leverage parallel processing capabilities, thereby improving their responsiveness and adaptability to dynamic environments (Duro et al., 2019). Such a distributed approach allows the memory system to scale dynamically: additional memory units can be enabled or added as needed and disabled when no longer required, improving flexibility and resource management. In addition, failures of individual processes cannot damage the entire memory system.

Optimizing Access Efficiency

Another strategy to reduce data transfer load is to align memory access with the production rate of the data sources. We assume that data can be categorized as either periodically produced in streams or generated in response to specific conditions, i. e., event-driven. Polling mechanisms enable the system to periodically check the status of various sensors or components, ensuring that it remains aware of its operational context. This is particularly important in scenarios where continuous data flow is not feasible or required. On the other hand, event-driven communication allows robots to react directly to changes in their environment or internal states. This approach is particularly beneficial in dynamic settings where immediate responses are crucial.

In systems with limited communication bandwidth, unnecessary memory requests should be avoided, so both approaches have a justified purpose. A hybrid approach ensures that the robot can efficiently manage its memory while remaining responsive to critical events, thus enhancing its overall performance (Liu et al., 2017).

Evaluating Data for Storage

Given the large quantities of data generated by the sensors in humanoid robots, the memory system must manage these volumes efficiently while providing long-term storage capabilities despite the limitations of main memory. Since it is impractical to store all data indefinitely, the system must prioritize data based on its relevance. Data generated near significant and attention-attracting events, or "keypoints" should be strengthened and prioritized for storage (Bui et al., 2019; Freedman and Adams, 2011). Redundant or non-attention-grabbing information can be ignored in order to save computing power and storage space. Additionally, dimensionality reduction techniques can be employed to create meaningful representations of the data in reduced form. Depending on the specific application, specialized models can be used to aggregate data into appropriate representations (Bärmann et al., 2024b). Moreover, the system must be able to assess and delete previously stored data if it becomes outdated, irrelevant, or incorrect – so a continuous evaluation is required.

It is important to note that these storage requirements have parallels to long-term memory and forgetting in biological systems. However, from an engineering perspective, these are viewed as technical considerations. In an ideal system with unlimited storage and computational resources, long-term storage and data deletion would not be necessary. Nevertheless, the conceptual memory requirements outlined in Section 4.1.1 are equally applicable to long-term memory systems.

In conclusion, alongside the conceptual requirements, we argued that a memory system for humanoid robots must also be distributed, access-efficient, and equipped with long-term storage capabilities. These allow the memory to be responsive, scalable and efficient – even when large amounts or high-frequencies of data have to be processed and stored.

4.2. Fulfilling Key Characteristics: A High-Level Perspective

Having outlined the fundamental characteristics that guided the development of our memory system in the previous chapter, we now turn to an analysis of how our implementation effectively fulfills these requirements. Setting the stage for a deeper dive into the technical specifics of the system's implementation, this section provides a high-level perspective on how the system addresses the conceptual and technical demands of cognitive robotics, demonstrating its role as an active, structured, and scalable component within a robot cognitive architecture.

4.2.1. Conceptual Requirements in Practice

Our memory system implements a unified multi-modal data representation framework that enables the integration of sub-symbolic and symbolic information. By *unified* we mean that all knowledge is defined similarly, no matter if it is semantic, procedural or episodic information. This allows the containers and learning processes anchored in the memory system to be defined as universally as possible. Consisting of a fixed set of base types, the data representation framework can be used to represent arbitrary data – e. g., visual, auditory, tactile data or recognized text, objects or humans. Although containers and learning processes in the memory are defined as universally as possible, it must be able to identify the actual data. Otherwise, if data is treated too generally, important information about the data type is probably lost and cannot be used for learning or reasoning. Our knowledge representation framework supports introspection, i. e., data whose type is (at least partially) known at any point in time. This capability enables the robot to fully evaluate its own knowledge, assess the reliability of stored information, and adapt its learning strategies accordingly, supporting self-monitoring, allowing the robot to identify inconsistencies, and refine internal models.

As motivated, we assume that all data modalities should be stored in an episodic-like manner. Context, such as spatial or temporal cues, should serve as key for retrieving information from the episodic-like structure. Our system maintains an episodic memory that records experiences over time, allowing the robot to recall past events, analyze patterns, and learn from previous interactions. We capture temporal sequences of events, enabling contextual recall and improving decision-making in dynamic environments. Additionally, our design supports retrospective analysis, facilitating processes such as anomaly detection, event prediction, and autonomous adaptation to new environments.

As an active learning process, our memory supports, among others described in Section 4.2.2, the prediction of data. Through the general representation of knowledge in memory, general prediction models can be defined, which can then be used for any type of data. In addition, specialized models can be defined for specific data types. This ensures that the overall system is as versatile as possible.

To support reasoning and learning, our architecture incorporates associative knowledge structures. This implies, that knowledge in memory is uniquely identifiable, so that such associations can be defined.

4.2.2. Technical Requirements in Practice

The technical requirements were motivated by the fact that robots have to produce and process large amounts of data. These data volumes have a major impact on the responsiveness, scalability and efficiency of the overall system.

Given the high computational demands of humanoid robotics, our memory system is implemented in a distributed fashion using Ethernet as a communication interface. Memory units operate as independent processes across multiple computing nodes, colocated with data-producing components to minimize latency. To balance responsiveness and computational efficiency, the memory system employs a hybrid approach combining event-driven updates with periodic polling. Data access mechanisms are optimized based on the expected usage patterns of different memory components. High-frequency sensor data is managed using streaming techniques, while event-driven access ensures that memory operations align with critical decision-making events, reducing redundant queries and improving real-time performance.

Since we do not have an ideal system with unlimited storage and computational resources efficient memory utilization is achieved through adaptive storage strategies. The system selectively retains information based on relevance, and frequency of access. Key event-driven data points are prioritized for long-term storage, while redundant or outdated information is forgotten. Further, our system incorporates controlled forgetting mechanisms to manage long-term storage. Knowledge is continuously evaluated for relevance, and less critical information is either compressed or removed over time. As one compression technique we utilize auto-encoders (Rothfuss et al., 2018) to learn a spatial representation for experiences. Special training of these auto-encoders also allow prediction of information.

4.3. Knowledge Representation Framework

We argued that, in the development of memory systems for robot cognitive architectures, a unified knowledge management and representation are crucial for ensuring scalability, optimal performance and interoperability across different components. But how can we represent arbitrary knowledge in an artificial system, so that we can build a memory system on top, that is independent of the knowledge it stores and thus highly extensible, while fulfilling the aforementioned requirements of the data representation, namely introspection and multi-modality? Designing such an efficient data representation is crucial for a robotic system, as it requires the ability to inspect and analyze data in real-time, which is not inherently supported by programming languages like C++, often used in robotics for its performance benefits.

To address this, we propose the ArmarX Object Notation (ARON)[1], a variant-based knowledge representation framework that offers a flexible and introspectable data format implemented in C++ but with extensions in other programming languages like Python (also referred to as Introspectable Data Format (IDF) (Peller-Konrad et al., 2023)). ArmarX Object Notation (ARON) leverages a recursive variant formulation, tailored to support robotics-specific data types, and facilitates efficient data exchange between different processes and machines via network transfer. The core elements of ARON – *static type objects*, *dynamic data objects*, and *conversion utility* – are designed to separate business logic from data transfer logic, enabling greater flexibility and scalability in complex robotic architectures.

All knowledge is composed of a fixed set of basic types, which can be combined to form more complex types. Inspired by JavaScript Object Notation (JSON) (Pezoa et al., 2016), those basic types have been chosen to be as general as possible, while supporting special representations commonly used in robotics, such as pointclouds, images, trajectories, vectors and matrices, so that they can be used to represent arbitrary robot knowledge and being easily extensible. The data is introspectable, as parts of the type can be inferred from the data itself. Both, data and type, support multi-modality, as they can represent different things, e. g., plain text, a latent representation or a graph.

However, a variant-based representation is usually hard to use and maintain, as it is not clear, what exactly a variant represents statically. Programmers would have to check the type

[1]Source-code publicly available: https://sw.pages.h2t.iar.kit.edu/aron/core/aron.html

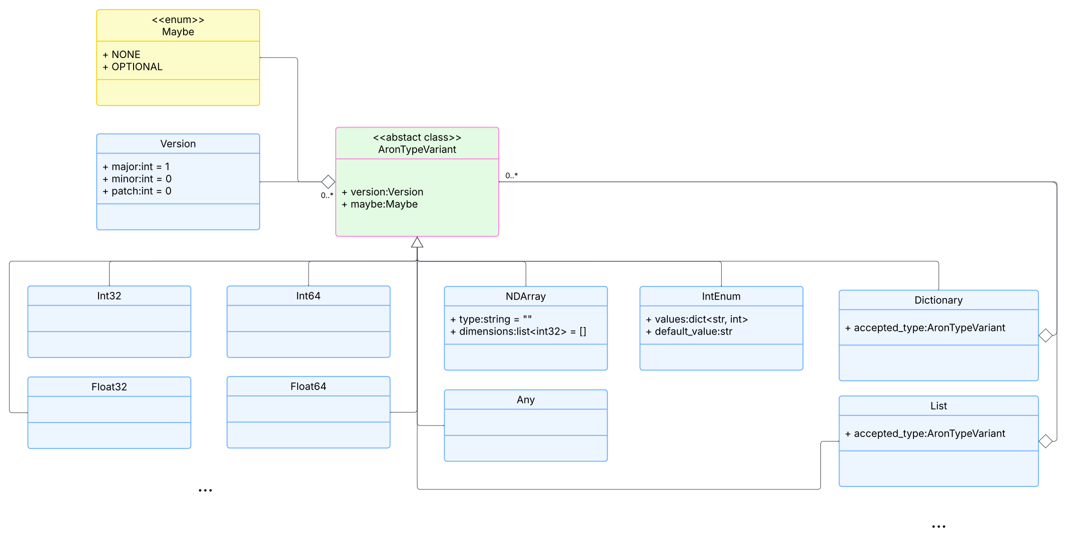


Figure 4.3.: Unified Modeling Language (UML) diagram of the ARON data object definition. Recursive objects, i. e., dictionaries and lists, recursively own their abstract base class. This allows defining a variant-like structure from a fixed set of base types. Type objects only contain information that is needed to identify the static type of an ARON object. Thus, primitives do not have any members as their static type is already complete by the class itself. Recursive objects contain one member to identify their accepted type.

of variant before accessing it, which is error-prone and cumbersome. To address this issue, our framework offers a large number of utility and convenience methods for this case, such as visitors and factories. Above all, we utilize code-generation to create a statically typed interface for the variant-based data representation. This allows us to use the variant-based representation in a statically typed language, still being able to reason about the data statically and to use features such as code completion and automated type deduction. Conversion between the variant-based representation and the statically typed representation is done automatically during network transfer, so that the programmer does not have to think about it.

In the following we will outline how type and data objects are represented and generated in our system.

4.3.1. Type Objects

ARON supports the annotation of data with static type information through type objects. These annotations enrich data objects with more details about “what” they actually represent and enable more robust processing and interaction.

Type objects come in five categories:

- Primitive type objects represent simple data types such as integers, floating-point numbers, booleans and strings.
- Multidimensional byte arrays will be used for large data types to allow efficient transfer.

- Enumeration type objects, e. g., integer enums.
- Container type objects like lists, dictionaries and objects, representing more complex data structures, which recursively contain other type objects.
- A special *any* type object, which should be used, if the parts of a type are not clearly determined.

The structure of the ARON type objects is shown in Figure 4.3. A detailed list of which types have been predefined (i. e., constructed from the basic categories but provided to the programmer as a separate type for usability reasons) in ARON follows in Section 4.3.3. Since all type objects share the same base class, it is possible to define recursive structures, which is required for container types. In contrast to data objects, which hold data in an untyped container, type objects only hold the type information which is needed to identify the static type information of knowledge. E. g., a dictionary $D[string \rightarrow int32]$, which should map from string to 32-bit integer is split into a data object $D_{data}[string \rightarrow any]$, and a type object D_{type} which only holds the accepted value type `int32` of the dictionary. Additionally, type objects store, whether they represent a *maybe*-type, i. e., nullable structures like optionals or pointers.

In summary, type objects enrich data objects with static type information. This allows a more precise analysis of the type. For example, consider the pick and place task as described in Chapter 3. As shown in Figure 4.3 (and further explained in Section 4.3.2), our knowledge representation framework allows images to be represented as `NDArrays`. If the object localization knows that the `NDArray` represents an RGB image, this image can be searched for features of the target object. However, if the type information is missing, the object localization cannot use any specialized algorithms. The `NDArray` could represent an RGB image, a depth image, a point cloud or a completely different type of data. However, with enough labeled data from experience (i. e., data in which the object is certainly present), an object recognizer could be trained to identify the target object in the unknown data. This is only possible because the elements of the `NDArray` can still be accessed, also for unknown types.

Even though type objects are optional, their presence enhances the efficiency and robustness of the system by enabling clear communication between different components. Further, splitting data and type into different representations optimizes their network transfer, as type objects usually only have to be sent once as static type information stays the same during operation.

4.3.2. Data Objects

Similar to type objects, data objects are designed as a recursive variant formulation. All data is composed of three primary types:

- Primitive data objects include primitive data types such as integers, floats, booleans and strings.
- Multi-dimensional byte arrays (`NDArray`) for large data.
- Container data objects represent more complex structures like lists and dictionaries.

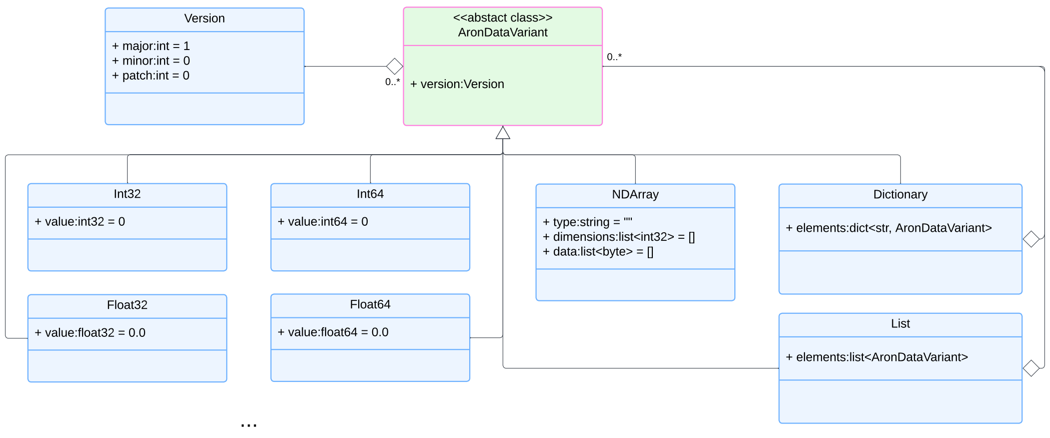


Figure 4.4.: UML diagram of the ARON data object definition. While type objects only describe the static type information, data objects describe the state of one instance of this type at a given time. Hence, all data object definitions have value members. We know that some information is redundant, e. g., an int data object already fully describes the static type – a corresponding type object would not provide any additional information. This is intended as data objects should be introspectable, i. e., it should be possible to at least derive parts of the static type, even if a corresponding type object is missing.

Type Object	**Data Object**
`int32`, `int64`, `float32`, `float64`, `string`, `bool`	`int32`, `int64`, `float32`, `float64`, `string`, `bool`
`matrix`, `orientation`, `image`, `pointcloud`, ...	`multi-dimensional byte array (NDArray)`
`int32 enumeration type`	`int32`
`object`, `dictionary`, `tuple`, ...	`dictionary`
`list`, `pair`, ...	`list`

Table 4.2.: Relation from type objects to data objects. For each type object there is one related data object. Type objects enrich data objects with static type information. E. g., given a multidimensional byte array data object, a type object can specify whether it should be reinterpreted as an image, a pointcloud, a quaternion, etc.

The structure and members of the ARON data objects is shown in Figure 4.4. The relationship between data objects and type objects, i. e., what type objects correspond to which data object, is listed in Table 4.2.

A data object *realizes* a type object if all elements of the data object either relate to the corresponding element of the type object or are not present in the type object and thus unknown. For example, a data dictionary D_1 with elements $"x" \rightarrow int32(0)$ and $"y" \rightarrow float32(1.0)$ realizes the type object T_1 with elements $"x" \rightarrow int32$, $"y" \rightarrow float32$. D_1 even fully realizes T_1 because D_1 and T_1 have the same amount of members. A data dictionary D_2 with elements $"x" \rightarrow \texttt{int32}(0)$, $"y" \rightarrow \texttt{float32}(1.0)$ and $"s" \rightarrow \texttt{string}("abc")$ would partially

Listing 4.1: Definition of UserCommand type in XML

```xml
<?xml version="1.0" encoding="utf-8" ?>
<typedefinition>
    <generatetypes>
        <object name="UserCommand" namespace="::example::dto">
            <objectchild key="cmd">
                <string/>
            </objectchild>
        </object>
    </generatetypes>
</typedefinition>
```

realize T_1. However, a dictionary D_3 with element $"x" \to$ `string`$("abc")$ would not realize T_1, because the member x should have type `int32` according to T_1.

Since type objects can define maybe types, data objects are always declared to be nullable. This means that a data object D_4 with value $NULL$ (i. e., no value is assigned) can realize a type object T_2 with is defined to accept e. g., optionals - regardless of how T_2 is defined. However, D_4 cannot realize T_1, as T_1 was declared as a non-nullable type by default.

4.3.3. From Business Logic to Network Transfer

Since dynamically typed objects often impair programmability, especially when statically typed programming languages are used, ARON employs code generation to generate statically typed classes that can be automatically converted into the dynamically typed representation. Our knowledge representation framework allows the specification of types in markup languages, such as Extensible Markup Language (XML), from which a class definition is generated and linked at compile time.

To illustrate how type and data objects are defined and how code is generated, let's assume the scenario as described in Chapter 3. Even such a simple pick-and-place task requires a lot of different knowledge modalities. First of all, the service robot needs to understand the command by the user. The command is most likely stored as a simple string, derived from the spoken text. Such a type specification for ARON in XML is shown in Listing 4.1. Every ARON type definition file must begin with the root tag `typedefinition`, which contains at least one `generatetypes`-tag. Nested to the `generatetypes`-tag, one can define object types and enum types. Here, an object type with name `UserCommand` and namespace `::example::dto` is defined. In our robot software framework ArmarX we usually use the `dto` namespace for Data Transfer Objects (DTOs), to make clear, that these objects should not be used in business logic.

A simplified version of the generated class for the `UserCommand`-type is given in Listing 4.2. For simplicity, we use a relaxed version of C++, as this is the language we usually use to program our robots for performance reasons. Next to constructors, destructors and operators, which

Listing 4.2: Generated class file for UserCommand type in relaxed C++

```cpp
// simplified version with omitted class namespaces, access levels, templates, const qualifiers and inheritance related specifiers
class UserCommand
{
    // member variables
    string cmd = "";

    // ctor, copy ctor, dtor
    UserCommand();
    UserCommand(UserCommand&);
    UserCommand(UserCommand&&);
    ~UserCommand();

    // assign, equality op
    UserCommand& operator=(UserCommand&);
    UserCommand& operator=(UserCommand&&);
    bool operator==(UserCommand&);
    bool operator!=(UserCommand&);

    // create an aron type object for this class
    static aron::type::WriterT::ReturnType writeType(::aron::type::WriterT& _aron_writer)
    {
        auto _aron_object_maybe = ::aron::type::Maybe::NONE;
        auto _aron_object = ::aron::type::write_object(_aron_writer,
                                                        "UserCommand",
                                                        _aron_object_maybe);
        {
            auto _aron_member_maybe = ::aron::type::Maybe::NONE;
            auto _aron_member = ::aron::type::write_string(_aron_writer,
                                                            _aron_member_maybe);
            _aron_object["cmd"] = aron_member;
        }
        return _aron_object;
    }

    // create an aron data object from the members of this class
    aron::data::WriterT::ReturnType write(::aron::data::WriterT& _aron_writer)
    {
        auto _aron_dict = ::aron::data::write_dict(_aron_writer);
        {
            auto _aron_member = ::aron::write_string(_aron_writer, this->cmd);
            _aron_dict->insert("cmd", _aron_member);
        }
        return _aron_dict;
    }

    // set the members of this class from the _input aron data object
    void read(::aron::data::ReaderT& _aron_reader,
              ::aron::data::ReaderT::InputT& _input)
    {
        _aron_member = ::aron::read(_aron_reader, _input->at("cmd"));
        this->cmd = _aron_member->value;
    }
}
```

generation highly depend on the target language (e. g., here, fulfilling the rule-of-five[2]), the generated class contains the methods

- `writeType(WriterT&)`: used to create a ARON type object, which matches the definition of the generated class. Here, the returned type object contains information about the class, e. g., the type, the class name and the string member `cmd`. This method is, by definition of the type objects which only contain static type information, independent of the value of the members and thus defined static.

[2]see https://en.cppreference.com/w/cpp/language/rule_of_three

Listing 4.3: Definition of Location type in XML

```xml
<?xml version="1.0" encoding="utf-8" ?>
<typedefinition>
    <generatetypes>
        <object name="Location" namespace="::example::dto">
            <objectchild key="name">
                <string/>
            </objectchild>
            <objectchild key="global_pose">
                <pose/>
            </objectchild>
        </object>
    </generatetypes>
</typedefinition>
```

Listing 4.4: Definition of Object type in XML

```xml
<?xml version="1.0" encoding="utf-8" ?>
<typedefinition>
    <generatetypes>
        <object name="Object" namespace="::example::dto">
            <objectchild key="name">
                <string/>
            </objectchild>
            <objectchild key="position">
                <object name="Location"
                        namespace="::example::dto" />
            </objectchild>
        </object>
    </generatetypes>
</typedefinition>
```

- `write(WriterT&)`: used to create a ARON data object from the values of the members of an instance of the class. Here, the returned dictionary has only one element, mapping the key "*cmd*" to the value of the member `cmd` at the time the method is called. This method copies the value of the members into the newly generated dictionary.
- `read(ReaderT&, InputT&)`: used to set the members of an instance of the class from an input ARON data object. To do this, the values are copied from the ARON data object.

Additional `read` and `write` methods that use move semantics are generated but have been omitted here for simplicity. The code generation assures that the generated `read` and `write` functions are compatible with general reader and writer interfaces, making the system highly extensible for new conversion strategies. Thus, `ReaderT` and `WriterT` implementing these interfaces, are used to separate a conversion algorithm from the data objects' structure. In addition to readers and writers for the variant representation, ARON offers to convert data to a textural representation (e. g., JSON). Note that even if a data object partially realizes a type, it can still be read by the generated class, with unmatched members remaining uninitialized or default initialized, making the system highly adaptable. Next to these methods, a lot of utility and convenience methods are generated.

Listing 4.5: Definition of Proprioception type in XML

```xml
<?xml version="1.0" encoding="utf-8" ?>
<typedefinition>
    <generatetypes>
        <object name="Proprioception" namespace="::example::dto">
            <objectchild key="global_pose">
                <pose/>
            </objectchild>
            <objectchild key="configuration">
                <dictionary>
                    <float32/>
                </dictionary>
            </objectchild>
        </object>
    </generatetypes>
</typedefinition>
```

After recognizing the command, the service robot needs to analyze its knowledge about the target object, in this case, the coffee, in particular it must know its position. If symbols are used (e. g., kitchen-counter or the robots' right hand), the robot additionally needs to know the global coordinates this symbol refers to. An exemplary definition of object related knowledge is given in Listing 4.4. Here, the `pose`-tag refers to a 4×4 matrix of type `float32`. To enhance usability, ARON offers several aliases and compound type definitions for datatypes commonly used in robotics, such as matrices, orientations, images, and pointclouds. Argument-dependent Lookup (ADL) ensures that the generated classes remain clear and consistent. Depending on the used alias or compound type, ARON offers optimization strategies for the transfer of data over the network. E. g., types only containing numeric values can be sent as a `NDArray`, whereby class-related information (e. g., class and member names) is intentionally omitted to reduce the overall size of the data object, allowing efficient network transfer – even for large data. In addition, programmers can easily add new compound types (i. e., created from the set of base types) or aliases without changing the core framework. This greatly increases usability and flexibility of the system as the core remains stable, but new types can be added at any time. A list of commonly used aliases and compound type definitions in ArmarX is listed in Section A.2.

Next, the service robot needs to navigate to the target location and to visually search for the coffee. To monitor its actions, the robot needs knowledge about its current state in the world. In our case, knowledge about its global position is bundled with its proprioception, as shown in Listing 4.5. The robot's configuration is modeled as a dictionary that maps from joint names to joint values while its global position is modeled as a 4×4 `float32` matrix (using the alias `pose`).

Perceptions of the camera, from which the robots' global pose and object poses can be recognised, may be defined as in Listing 4.6. Here we assume a robot equipped with a full-HD RGB-D camera system. Please note that other methods to calculate the global pose of the

Listing 4.6: Definition of Vision type in XML

```xml
<?xml version="1.0" encoding="utf-8" ?>
<typedefinition>
    <generatetypes>
        <object name="Vision" namespace="::example::dto">
            <objectchild key="image_rgb">
                <image width="1920" height="1080" type="rgb24"/>
            </objectchild>
            <objectchild key="image_depth">
                <image width="1920" height="1080" type="depth32"/>
            </objectchild>
        </object>
    </generatetypes>
</typedefinition>
```

Listing 4.7: Definition of KnownGrasp type in XML

```xml
<?xml version="1.0" encoding="utf-8" ?>
<typedefinition>
    <generatetypes>
        <object name="KnownGrasp" namespace="::example::dto">
            <objectchild key="reference_object">
                <object name="Object" namespace="::example::dto"/>
            </objectchild>
            <objectchild key="prepose">
                <framed_pose/>
            </objectchild>
            <objectchild key="grasppose">
                <framed_pose/>
            </objectchild>
        </object>
    </generatetypes>
</typedefinition>
```

robot or the objects may require other knowledge modalities, such as laser scanner data or stereo vision.

To fetch the coffee, the robot needs to know about possible grasps. One way of defining known grasps is to specify the relative poses for performing the grasp trajectory (e. g., prepose and grasp pose). Grasping is then realized by following this grasp trajectory, closing the robot's end effector at the end point of the trajectory, and returning the closed hand to a safe position. A type definition for such predetermined known grasps is shown in Listing 4.7. Since objects, just like robots, can have joints (e. g., doors on cupboards), ARON offers a special `FramedPose` type. Framed poses consist of a unique frame name, which the reference object must have (such as the joint name), and a pose relative to the origin of this frame.

Finally, for a targeted handover, the service robot requires the pose and configuration of the human. Both can be recognized from the robots' vision. A type for representing human agents in the robot's perception could be structured similarly to the robot's proprioception (i. e., combining global pose and configuration).

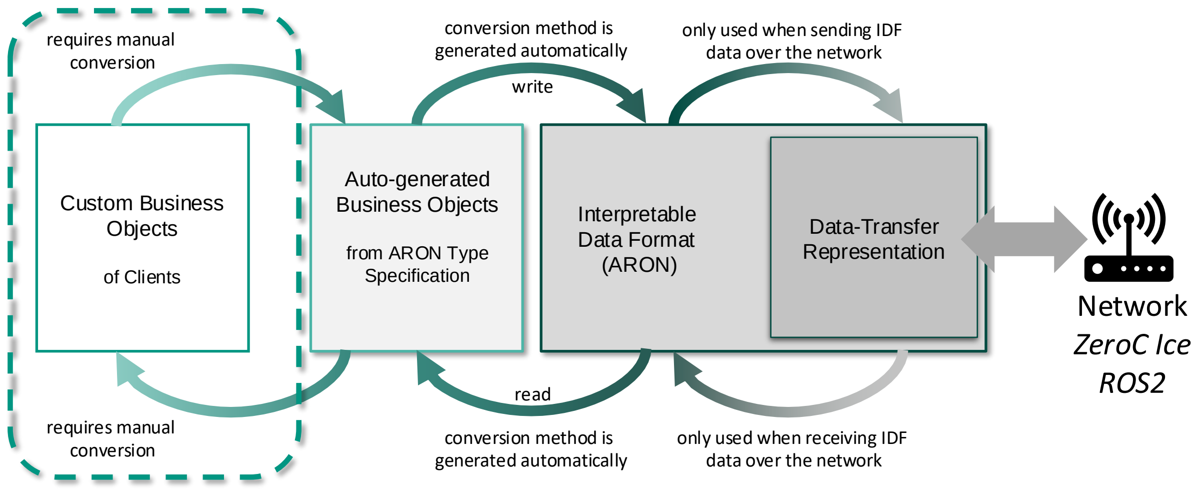

requires manual conversion
conversion method is generated automatically
write
only used when sending IDF data over the network
Custom Business Objects
of Clients
Auto-generated Business Objects
from ARON Type Specification
Interpretable Data Format (ARON)
Data-Transfer Representation
Network
ZeroC Ice
ROS2
requires manual conversion
read
conversion method is generated automatically
only used when receiving IDF data over the network

Figure 4.5.: Data representation levels when transferring ARON objects over the network. Almost all conversions are done automatically by the system. Only the conversions from custom business objects (if used) to the auto-generated representation require a manual definition. ©Elsevier (Peller-Konrad et al., 2023)

We have emphasized that ARON, next to the flexible definition of data and types, facilitates efficient data exchange between different processes and machines via network transfer. This makes further levels of representation necessary. As illustrated in Figure 4.5, data sent from one process to another via the network is assigned the following representations:

1. Manual Business Object (BO): Often, programmers implement functionality in a clean environment to limit dependencies and to be more flexible. By defining manual BO types, programmers can use their own language-specific representations up to the point when data needs to be sent to another process.
2. Generated BO: Language-specific statically typed representation to allow programmers to access advanced features of the target language, such as static type checking and code completion, while being able to convert the data into the variant-based representations.
3. Variant representation: The dynamically typed representation. This allows the general definition of containers and functions in, but not only, the memory framework. Generated business objects can be automatically converted into this representation.
4. Data-transfer representation: This representation allows the serialization and deserialization of data. Usually, this representation is given by a specified Remote Procedure Call (RPC) framework (in the following also called middleware). At the time of writing, ARON supports Robot Operating System (ROS) (Macenski et al., 2022) and ZeroC Internet Communication Engine (ZeroC Ice) (Henning, 2004) by providing reader and writer objects that allow the automated conversion to the RPC frameworks representation. ARON thus adds a layer of abstraction to the data transfer layer and allows not only to mediate between different processes and programming languages of the same RPC framework, but can also be used for the transfer between RPC frameworks.
5. Serialized representation: The final representation for transmitting data, e. g., over the network. Usually, text (such as JSON) or binary representations (byte stream) are used.

At the time of writing this document, ARON provides its core data and type definitions as well as its conversion utility in the programming languages C++ and Python. However, the code-generation framework and features are only implemented in C++. In Python, however, this is not necessary, as Python is not a strictly typed language.

Both type and data objects contain version information. This is particularly interesting when objects are created by one process and read by another process. The reading process could use an outdated standard for deserialization due to incorrect or non-updated dependencies. To avoid version conflicts, the version of the objects is sent and checked with every network call.

To summarize, ARON represents a flexible and introspectable knowledge representation framework that enhances data management in distributed cognitive robotic systems. By providing a robust mechanism for separating business logic from data transfer logic, and by supporting both data and type representation through recursive variants and code generation, ARON addresses key challenges in managing complex, multi-modal data in robotics. Its design ensures compatibility with standard programming languages and standard middlewares, making it a valuable tool for developing scalable, high-performance robotic architectures.

4.4. Working Memory

We follow the approaches by Atkinson and Shiffrin (1968); Baddeley and Hitch (1974); Tulving (1972), i. e., that a memory system is divided into Working Memory (WM) and Long-term Memory (LTM) equipped with Episodic Memory (EM). Working memory serves as the dynamic, short-term memory component of the cognitive architecture, playing a critical role in real-time processing, decision-making, and action execution for humanoid robots. It focuses on managing information that is immediately relevant to ongoing tasks, allowing the robot to interact with its environment in a flexible and adaptive manner. By maintaining a constantly updated snapshot of such knowledge and of the current state of the robot and its surroundings, the working memory enables the system to make quick, informed decisions and respond to dynamic changes in the environment. In our system, the WM is bounded by the main memory of the system.

Long-term memory and episodic memory store vast amounts of knowledge and experiences over time in persistent storage. To increase the amount of stored entities as well as for performance reasons, the LTM has the capability to compress information into a more graduated representation or to remove knowledge. For components that access the memory (in the following called *memory clients*), i. e., read from or write data into it, the WM acts as an interface. We believe, that the LTM should only be managed by the memory itself – not from extern.

While we do see sensory memory as a part of the architecture, we do not go into detail about it here. Sensory memory describes the management of knowledge for a very short duration until

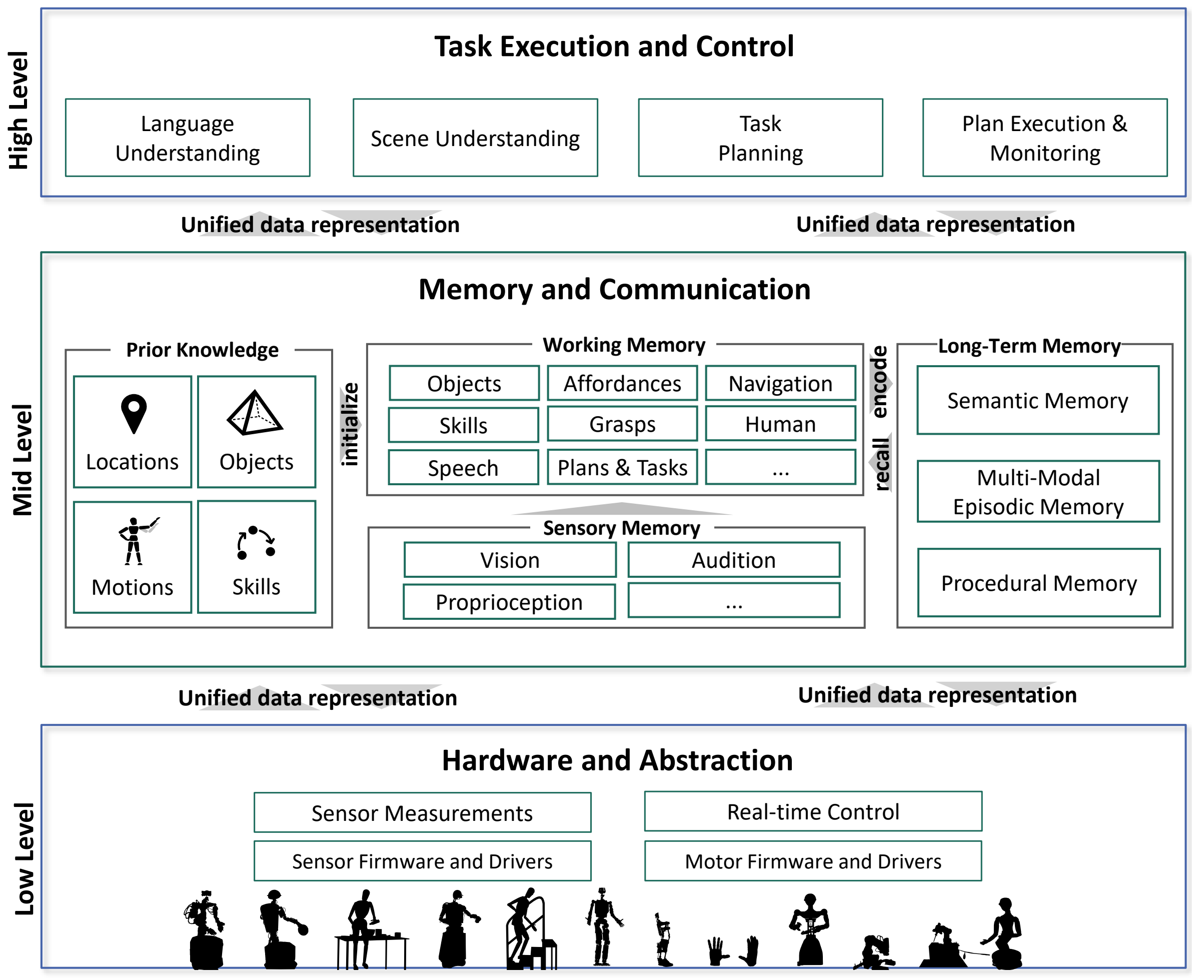


Figure 4.6.: Detailed overview of memory system within the cognitive architecture in ArmarX. The Prior Knowledge (PK), working memory and long-term memory are subdivided into several, possibly distributed subsystems. The unified data representation supports the extensibility of the system and ensures efficiency and robustness. ©Elsevier (Peller-Konrad et al., 2023)

it may be passed to WM. Although our memory provides containers and utility methods for sensory memory, in order to keep the communication paths as short as possible, we assume that the management of sensory memory is the responsibility of the individual components that produce knowledge. Additionally, our memory system uses Prior Knowledge (PK), i. e., information that is provided by the programmer and thus already known to the robot in advance, such as object or environment models or pre-defined motion trajectories. PK can be seen as a special sort of long-term memory, but in our system, the knowledge of the LTM is determined exclusively by the system itself, while PK is specified by the programmer.

We motivated that the memory should be highly flexible and extensible and motivated the requirement of a unified data representation for all memory-related data structures. Memory functions and containers should be as general as possible, hence, we utilize the aforementioned ARON knowledge representation framework.

This section provides a detailed overview of the working memory of our memory system, focusing on its architecture, functionality, and integration within the broader cognitive framework.

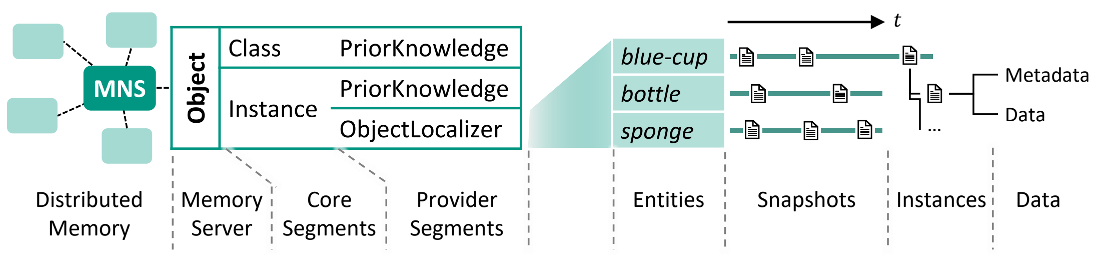


Figure 4.7.: Hierarchical structure of one single memory server (here as example *object*-memory). Data is organized in different segments and entities up to an episodic management of snapshots of the same entity. ©Elsevier (Peller-Konrad et al., 2023)

We discuss the technical challenges of implementing a memory system in a humanoid robot, the strategies employed to ensure efficient data flow and access, and how the WM supports both perception and action in real time. Through this, we aim to highlight the importance of a well-designed memory in achieving high-level cognitive performance in autonomous robots.

4.4.1. Data Organization

A hierarchical structure of our memory allows for flexible, organized storage of multi-modal data, using ARON as a knowledge representation framework in the lowest layers. This structure, also shown in Figure 4.7, is divided into:

- Memory Servers: Each memory server corresponds to a semantic category, such as objects, actions, or skills.
- Core Segments: These segments categorize data by modality (e. g., object instances, images, or grasp affordances). Core segments are homogeneous containers defined by the memory server and contain data of a shared type.
- Provider Segments: Providers (processes writing data to a memory server) create dynamic namespaces within core segments to avoid naming conflicts and allow for modality-specific extensions.
- Entities: An entity represents a concept or physical object (e. g., a specific object instance, a captured image, grasp candidates, etc.) and evolves over time as new data is received.
- Entity Snapshots: Each entity is episodic, with timestamped snapshots representing a specific time in the entity's evolution.
- Entity Instances: An entity snapshot contains one or more instances of data objects (e. g., multiple images from a stereo camera or multiple grasp candidates for one object).

Core and provider segments are the only layers that can be typed. If a segment is typed, then the memory assures that all data objects realize this type. If a core and a provider segment have type information set, then the core segment type must be a base for the provider segment type. We motivate that we consider each entity in memory as a potentially changing variable, whether it is factual knowledge, an experience or procedural memory skills. For this reason, all knowledge is stored in memory as an episodic-like structure as entity snapshots

Listing 4.8: Definition of MemoryID type in XML

```xml
<?xml version="1.0" encoding="utf-8" ?>
<typedefinition>
    <generatetypes>
        <object name="MemoryID" namespace="::armem::dto">
            <objectchild key="memory_name">
                <string/>
            </objectchild>
            <objectchild key="core_segment_name">
                <string/>
            </objectchild>
            <objectchild key="provider_segment_name">
                <string/>
            </objectchild>
            <objectchild key="entity">
                <string/>
            </objectchild>
            <objectchild key="timestamp">
                <datetime/>
            </objectchild>
            <objectchild key="instance">
                <int/>
            </objectchild>
        </object>
    </generatetypes>
</typedefinition>
```

where we assume timestamps as keys. It must therefore be possible to assign each snapshot in the memory to a unique timestamp, the so-called *reference time*. This can be, for example, the time at which the data was created or sent. In the case of sensors, the reference time is usually the same as the time at which the data was created, as the sensor records the state of the world at a point in time. Furthermore, we argue that knowledge about changes in entities is at least as valuable as the data itself. For this reason, there is no way to remove knowledge from working memory for memory clients; knowledge can only be added or updated.

Since memory servers use ARON data and type objects as a knowledge representation framework at the lowest level, our memory can handle diverse data types, from raw sensor input to high-level symbolic representations. Each element is identified by a unique memory ID (which can also be used to link knowledge on data level, see its type definition in Listing 4.8), which enables cross-references between different modalities and allows for the creation of associations between data elements. We differentiate between *CoreSegmentIDs* where only a memory name and a core segment name have been specified. *ProviderSegmentIDs* where a memory name, a core segment name, and a provider segment name have been specified. It continues with *EntityIDs*, *EntitySnapshotIDs* and *EntityInstanceIDs*.

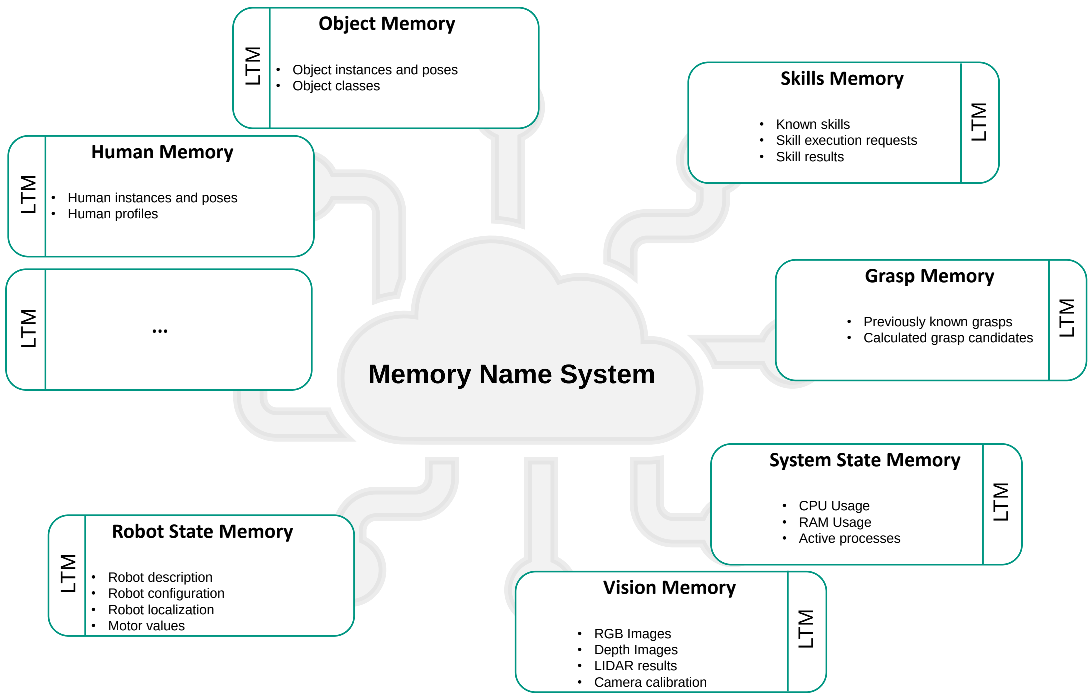


Figure 4.8.: Memory system as a set of distributed memory servers connected through the Memory Name System (MNS). Each memory server is implemented as an individual process within the system that may run on different machines. Here, we show some of the memory servers we usually use within the ArmarX robot software framework. ©Elsevier (Peller-Konrad et al., 2023)

4.4.2. Memory Name System

We argued that complex cognitive architectures, such as humanoid robots, require a distributed memory for responsiveness, scalability and efficiency reasons. Our memory system is implemented as a distributed architecture composed of multiple memory servers. Each memory server is a distinct process that communicates with other system components via Ethernet using middlewares. These servers can run on different machines and are linked through a central registry known as the Memory Name System (MNS), which allows memory clients to locate specific modalities using human-readable identifiers. The MNS resolves these human-readable identifiers to machine-readable IP addresses, port and process information, which is used to identify the target process on machine level. Memory servers can be added at any time and distributed to different computers. This decentralized structure ensures that the memory system can scale while optimizing performance by distributing the load and reducing network traffic and response times, if memory servers are located close (i. e., on the same machine) to related hardware and knowledge-producing components (e. g., a camera and a visual perception pipeline).

A schematic view of the MNS connecting several memory servers is shown in Figure 4.8. Each memory server runs as an individual process and on different machines, each one extending the system with new modalities. All memory servers together build the complete memory of the robot. A list of most of the memory servers we use in ArmarX is listed in Section A.3.

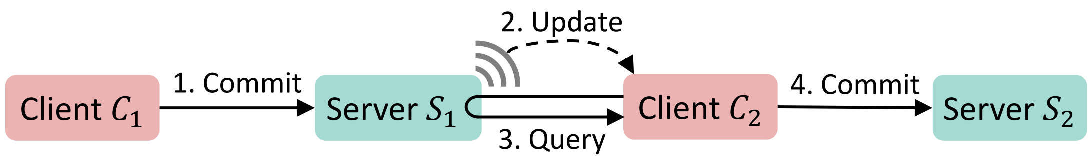


Figure 4.9.: Event-driven pipeline scheme. (1) Client C_1 writes data to server S_1 via a commit. (2) Server S_1 broadcasts an update notification, which is received by client C_2. (3) Client C_2 performs a query to S_1 to read the updated data. (4) Client C_2 processes the data and commits the result to another server S_2. In the case of periodic polling of information, step (2) is omitted and the client directly and periodically performs step (3). ©Elsevier (Peller-Konrad et al., 2023)

Listing 4.9: Construct Query Object through Builders

```
::armem::wm::QueryBuilder builder;
builder.memories.name("Object")
        .coreSegments.name("Instance")
        .providerSegments.name("ObjectLocalizer")
        .entities.name("blue-cup")
        .snapshots.all();
auto query = builder.buildQuery();
```

4.4.3. Querying and Committing Interfaces

Our memory system provides flexible mechanisms for reading from and writing to the memory. Writing is handled through *commits*, which bundle updates to entities and their snapshots. A commit requires a SnapshotID that is used to decide whether an entity in the memory needs to be updated or whether it represents new knowledge for which a new segment is created in the WM. The target memory server then accepts the commit and inserts the data it contains into its structures according to the specified timestamp. On the one hand, this ensures that knowledge that arrives late in the memory due to network latencies does not unintentionally overwrite newer data. On the other hand, the memory can issue a warning if a transfer has taken an unexpectedly long time, i. e., if the difference between arrival and dispatch time exceeds a certain threshold. After inserting, the memory server broadcasts a notification message to listening clients, containing the SnapshotIDs of the memory that have been updated by the commit. When a client receives a notification, it can determine whether a response is needed. If so, it can retrieve and process the updated data from the memory server. These update notifications enable the creation of event-driven data processing pipelines. In addition, each memory server can be requested at any time. This enables the high-frequency retrieval of data from memory. Both pipeline schemes are explained in Figure 4.9.

Reading from a memory server is done through *queries* by specifying a hierarchical search pattern following the memory data structure. Clients can request data at any level of the hierarchy – whether they want the latest snapshot, data within a time window, or a specific instance from a particular provider segment or all segments matching a given regex. These queries enable the retrieval of precise or approximate information depending on the application's requirements. Our memory framework offers builders to construct complex queries.

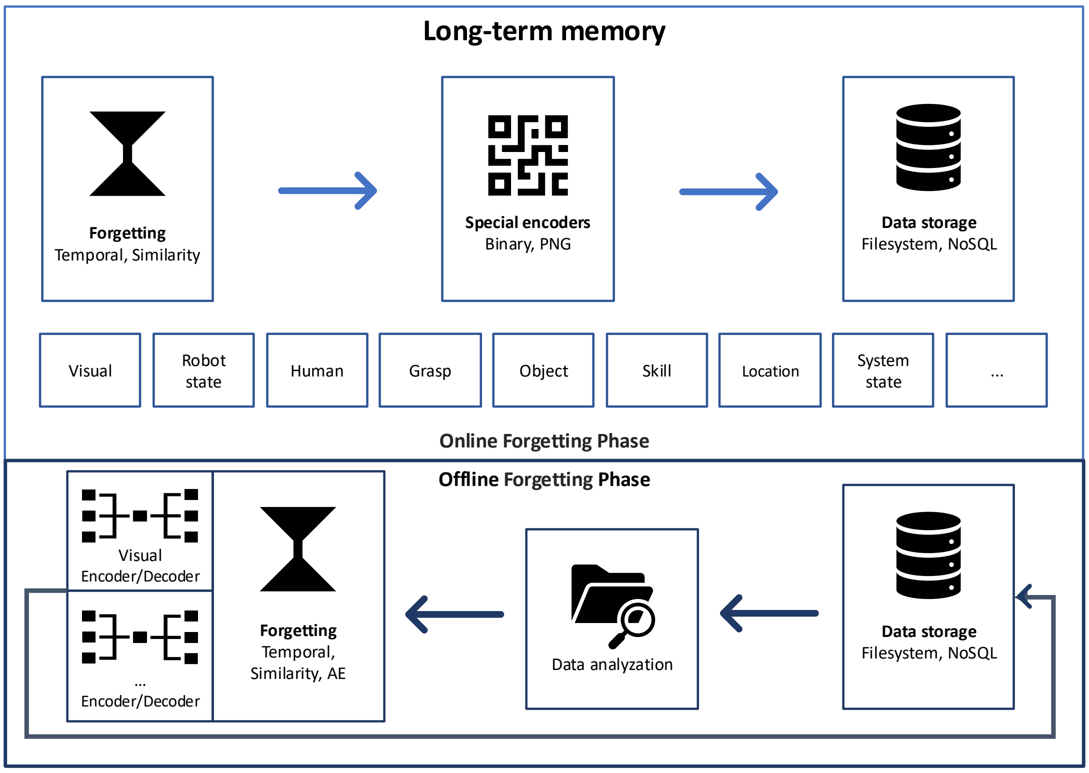


Figure 4.10.: The LTM processing pipeline. In an online forgetting phase, all incoming entities are filtered, converted into space-saving representations and finally stored in the filesystem or mongodb. When the robot is not in operation anymore, offline forgetting techniques further compress or remove knowledge. ©Elsevier (Peller-Konrad et al., 2023), modified

An example of such a query construction is shown in Listing 4.9. Here, all object instances of the blue-cup object localized by *ObjectLocalizer* from the object memory will be queried (the structure of knowledge in the object memory is depicted in Figure 4.7). This builder object also provides commonly used snapshot queries such as the latest snapshot (the current state), the snapshot at a specific point in time, the snapshots in a time window (e. g., in the last $10\,\mathrm{sec}$), and the latest n snapshots for convenience. Multiple queries at the same level can be combined to form a conjunction, such as retrieving data from two specific core segments in one single RPC. Each core segment query contains its own sub-queries for lower levels. This results in a tree-like query structure that mirrors the memory data structure, allowing for flexible and complex query combinations.

4.5. Long-Term Memory

While working memory handles short-term data, long-term memory provides scalable, persistent storage. In our system, LTM offers multiple backends – the filesystem where data is stored as simple files, or MongoDB (Chodorow, 2013). The LTM is therefore a slow but significantly higher capacity component in contrast to the WM, whose data is managed in the system's main memory. Nevertheless, even the LTM reaches its capacity limit when used for long periods. To optimize space and to balance data, the LTM in our memory system uses various conversion, compression and filtering techniques to store data efficiently, shown in Figure 4.12. Initially, incoming data is filtered based on frequency and relevance. Remaining data may be converted using methods such as PNG (Al-Ani and Awad, 2013) or MPEG (Pereira and Ebrahimi, 2002), reducing file sizes and enabling scalable storage. Due to the high frequency with which data is consolidated from WM into LTM, this decision and conversion must be real-time capable. In addition to online filtering and conversion, LTM performs periodic offline processing during inactive periods, using frequency-based, relevance-based or machine learning based techniques like Auto-encoders (AEs) to compress data into generalized latent representations. The latter creates a deep EM (Rothfuss et al., 2018) capable of reproducing and predicting stored data while maintaining the system's ability to query and retrieve information based on its original format.

4.5.1. Memory Consolidation and Online Forgetting

Data must be regularly removed from the WM to the LTM in order to free up space in the main memory for further data or other processes. On the one hand, the WM can be used as a queue, in that data can be removed according to the First-in-First-out (FIFO) principle. On the other hand, data can be prioritized. E. g., if a snapshot has been accessed very frequently in a short period of time, it is likely that it will be used again in the near future. It should therefore not be written from the main memory to the hard drive or even deleted. Instead, it should remain in the WM and other, less used data should be removed instead. Query statistics are therefore kept in our system for each snapshot so that the WM can be used as a priority queue.

In any case, not all data should be stored in the LTM if possible. For tasks that include searching for objects as described in Chapter 3, the robot may look at the same location for a long time to decide whether the target object can be found there. During this time, the camera continuously produces redundant data, which is first stored in WM and then transferred to LTM if no forgetting or filtering is involved. For this reason, our system supports the filtering of data from WM to LTM in real time using time-based decay, frequency- and similarity-based forgetting methods. The procedure is also shown in Algorithm 1. This procedure is used to check if snapshots have to be passed from working memory to long-term memory. This decision depends on a calculated activation value for snapshots, the frequency of consolidation attempts for snapshots of one entity and/or the similarity between the current entity snapshot and the last n consolidated entity snapshots $\mathcal{E}_{ltm}[: -n]$. Depending on preset thresholds (a_{min} for time-based decay, δ_{wait} for frequency-based mechanisms and d_{min} for similarity-based mechanisms), the mechanisms either forget an entity snapshot or consolidate it into the LTM.

Algorithm 1 Online forgetting procedure

```
1 : **Given**:
2 : $\mathcal{E}_{wm}$ – all snapshots of entity stored in working memory of entity $\mathcal{E}$
3 : $\mathcal{E}_{ltm}$ – all snapshots of entity stored in long-term memory of entity $\mathcal{E}$
4 : $a_{min}$ – minimum activation needed for time-based decay
5 : $\delta_{min}$ – minimum time difference needed for frequency-based forgetting
6 : $n$ – number of past snapshots to consider for similarity-based forgetting
7 : $d_{min}$ – minimum dissimilarity threshold for similarity-based forgetting
8 : $t_{now}$ – current timestamp
9 : $t_{ref}(x)$ – timestamp, a snapshot $x$ references
10 : $t_{used}(x)$ – timestamp, a snapshot $x$ has been used last
11 : $t_{cons}(E)$ – timestamp, a snapshot of entity $E$ has been consolidated to LTM last
12 :
13 : **Procedure**:
14 : **IF** new update $x_j \in \mathcal{E}$ arrives working memory:
15 :     **IF NOT** working memory overfilled:
16 :         **return**
17 :     **END IF**
18 :     **IF** time-based decay is used:
19 :         $a(x) = e^{t_{now}-t_{used}(x)}$ (or Ebbinghaus)
20 :         $x_i = \arg\min_{x \in \mathcal{E}_{wm}}(a(x))$
21 :         $rm = a(x_i) < a_{min}$
22 :     **ELIF** frequency-based forgetting is used:
23 :         $x_i = \arg\min_{x \in \mathcal{E}_{wm}}(t_{ref}(x))$
24 :         $rm = t_{now} - t_{cons}(\mathcal{E}) < \delta_{min}$
25 :     **ELIF** similarity-based forgetting is used:
26 :         $D(x, y) = mse(x, y)$ (or MAE or Cosine similarity)
27 :         $x_i = \arg\min_{x \in \mathcal{E}_{wm}}(t_{ref}(x))$ (or another decision criteria)
28 :         $rm = D(x_i, \mathcal{E}_{ltm}[: -n]) < d_{min}$
29 :     **END IF**
30 : **END IF**
31 : **IF** $x_i$ **AND** $rm$:
32 :     delete $x_i$ from $\mathcal{E}_{wm}$
33 : **ELIF** $x_i$ **AND NOT** $rm$:
34 :     move $x_i$ from $\mathcal{E}_{wm}$ to $\mathcal{E}_{ltm}$
35 : **END IF**
```

A snapshot x_i of entity $\mathcal{E}$ is only consolidated if

$$a_{min} \leq a(x_i), \text{when using time-based decay, or}$$
$$\delta_{wait} \leq t_{now} - t_{cons}(\mathcal{E}), \text{when using frequency-based forgetting, or}$$
$$d_{min} \leq D(x_i, \mathcal{E}_{ltm}[: -n]), \text{when using similarity based forgetting.}$$

$a(x_i)$ refers to the activation value of snapshot x_i. D describes a dissimilarity- or distance-metric, such as

$$\text{Mean Squared Error (MSE): } \frac{1}{n} \sum_{x_j \in \mathcal{X}_n} (x_j - x_i)^2$$

$$\text{Mean Average Error (MAE): } \frac{1}{n} \sum_{x_j \in \mathcal{X}_n} (x_j - x_i)$$

$$\text{mean cosine similarity: } \frac{1}{n} \sum_{x_j \in \mathcal{X}_n} (\frac{x_i \cdot x_j}{\|x_i\| \, \|x_j\|})$$

The other constants and variables are similar defined as in Algorithm 1. The activation value is calculated using an exponential function with a negative exponent ($a(x) = e^{-\Delta t}$, where Δt refers to the difference of the time x references and its last usage) or the Ebbinghaus curve (Murre and Dros, 2015).

While similarity-based methods are dependent on the modality used (i. e., they can only be applied to objects containing numerical data), frequency-based and time-based decay filters can be used for all entities.

4.5.2. Data Organization

We distinguish between two stages of data organization in the long-term memory: online filtered data and offline compressed data. Online filtering is performed in real-time, while offline compression is executed during inactive periods to keep the memory system responsive and efficient. Thus, consolidated data from WM to LTM, which is first filtered based on relevance or frequency, is stored in a cache $\mathcal{E}_{ltm\ new}$ in plain text format. During compression, e. g., using Auto-encoders, the data is transformed into a latent representation and stored in $\mathcal{E}_{ltm\ enc}$. The compression is usually lossy, meaning the original data cannot be fully reconstructed from the compressed version.

In both stages, data is organized in a hierarchical structure, similar to the working memory, with entities, snapshots, and instances. This structure allows for efficient storage and retrieval of data, enabling the system to manage vast amounts of information while maintaining a high level of organization and accessibility. As previously mentioned, when storing data first in plain text in the LTM cache, different conversion methods may be used depending on the data modality. This allows images, videos or depth images, for example, to be stored in a much more space-saving manner. As in WM, the long-term memory uses memory IDs as a key for direct access, regardless of whether it is stored in the file system or MongoDB.

If not converted in a special form, plain-text data is stored as JSON documents. As memory segments are only optionally typed, consistent data cannot always be guaranteed. For this reason, we only use methods of storing unstructured data.

4.5.3. Offline Forgetting and Predictiona

When the robot is not operational, we can make use of more powerful forgetting mechanisms that are not necessarily real-time capable. In addition to complete deletion, during offline forgetting, our system is able to filter out irrelevant information within one snapshot, successively removing unnecessary details from the entity, thus “blurrying” the raw input. To do this, we use machine learning methods in a Deep Episodic Memory (deep EM), trained in an offline phase. On the one hand, the snapshots are compressed. Secondly, knowledge is generalized and similar knowledge is summarized. The deep EM is modeled through an AE based deep neural network architecture that encodes data into a latent vector space in an unsupervised manner without requiring explicitly labeled data. Autoencoders are neural networks designed to learn efficient representations of data, typically for the purpose of dimensionality reduction. They consist of two main components: an encoder that compresses the input into a lower-dimensional latent space, and a decoder that reconstructs the original input from this compressed representation. The training objective is to minimize the reconstruction error, usually measured by the mean squared error between the input and the reconstructed output.

Specifically, we use Wasserstein Auto-encoder (WAE) (Tolstikhin et al., 2017), a generalization of VAE (Kingma and Welling, 2013), implemented using TensorFlow (Abadi et al., 2016). Variational Autoencoders extend the basic autoencoder framework by introducing a probabilistic approach to the latent space. Instead of learning a deterministic mapping from input to latent space, VAEs model the latent variables as distributions. The encoder outputs parameters of a distribution (typically a Gaussian), from which latent variables are sampled. The decoder then reconstructs the input from these samples. The training objective combines the reconstruction loss with a regularization term derived from the Kullback-Leibler divergence (Joyce, 2011), which encourages the learned latent distribution to be close to a prior distribution. This probabilistic framework allows for measuring distances between data points and generating new data points by sampling from the latent space, making them powerful generative models.

Wasserstein Autoencoders introduce another approach by leveraging the Wasserstein distance (or Earth Mover’s distance) (Tolstikhin et al., 2017) as a measure of the difference between the model distribution and the target distribution. WAEs aim to minimize a penalized form of this distance, which provides a more stable training process compared to traditional VAEs. Similar to VAE, the WAE architecture incorporates a regularization term that encourages the latent space to align with a prior distribution but with the added benefit of the Wasserstein distance, which is less sensitive to the choice of the prior and can lead to better generative performance. This makes WAEs particularly effective in scenarios where the data distribution is complex.

Instead of storing a full timeline of raw input data in LTM, our system maintains only a temporal sequence of latent vectors, along with the corresponding encoder and decoder weights. The encoding procedure for new data of entity $\mathcal{E}$ is shown in Algorithm 2. This collection of latent vectors in $\mathcal{E}_{lmt\ enc}$ over all entities $\mathcal{E}$ forms the EM of our system. The latent representation not only enables compression, reconstruction, and the computation of

Algorithm 2 Offline encoding-based gradual forgetting procedure

```
1 :  Given:
2 :  $\mathcal{E}_{ltm\ new}$ – all unencoded snapshots new in long-term memory of entity $\mathcal{E}$
3 :  $\mathcal{E}_{ltm\ enc}$ – all encoded snapshots already in long-term memory of entity $\mathcal{E}$
4 :  $enc_{\mathcal{E}}(x)$ – encoding model for entity $\mathcal{E}$
5 :
6 :  Procedure:
7 :  FOREACH $x \in \mathcal{E}_{ltm\ new}$:
8 :     $e = enc_{\mathcal{E}}(x)$
9 :     $\mathcal{E}_{ltm\ enc} = \mathcal{E}_{ltm\ enc} + [e]$
10 : END FOREACH
11 :
```

Algorithm 3 Offline similarity-based forgetting procedure

```
1 :  Given:
2 :  $\mathcal{E}_{ltm\ enc}$ – all encoded snapshots already long-term memory of entity $\mathcal{E}$
3 :  $n$ – number of past snapshots to consider for similarity-based forgetting
4 :  $d_{min}$ – minimum dissimilarity threshold for similarity-based forgetting
5 :
6 :  Procedure:
7 :  FOREACH $x \in \mathcal{E}_{ltm\ enc}$:
8 :     $D(x) = mse(x, y)$ – or another distance function
9 :     $x_i = \min_{x_j \in (\mathcal{E}_{ltm\ enc}/x)[:-n]}(x_j, x)$
10 :    $rm = D(x, x_i) < d_{min}$
11 :    IF $rm$:
12 :       delete $x_i$ from $\mathcal{E}_{ltm\ enc}$
13 :    END IF
14 : END FOREACH
15 :
```

distances between data points, but also facilitates the prediction of future data states. As a result, the EM continuously expands with the acquisition of new knowledge in the LTM and can utilize the deep EM to re-activate relevant information into WM by reconstructing it or generating predictions.

As described in Algorithm 2, a separate autoencoder is trained for each entity. Snapshots $x \in \mathcal{E}_{ltm\ new}$ – which remain in their original form (plain text) but have been subject to preliminary online filtering – are periodically processed, typically when the robot is not in operation. In the first step, numerical values are recursively aggregated across the possibly hierarchical object structures, and the ARON object is transformed into a vectorized format. This transformation can be performed automatically due to the introspective nature of ARON.

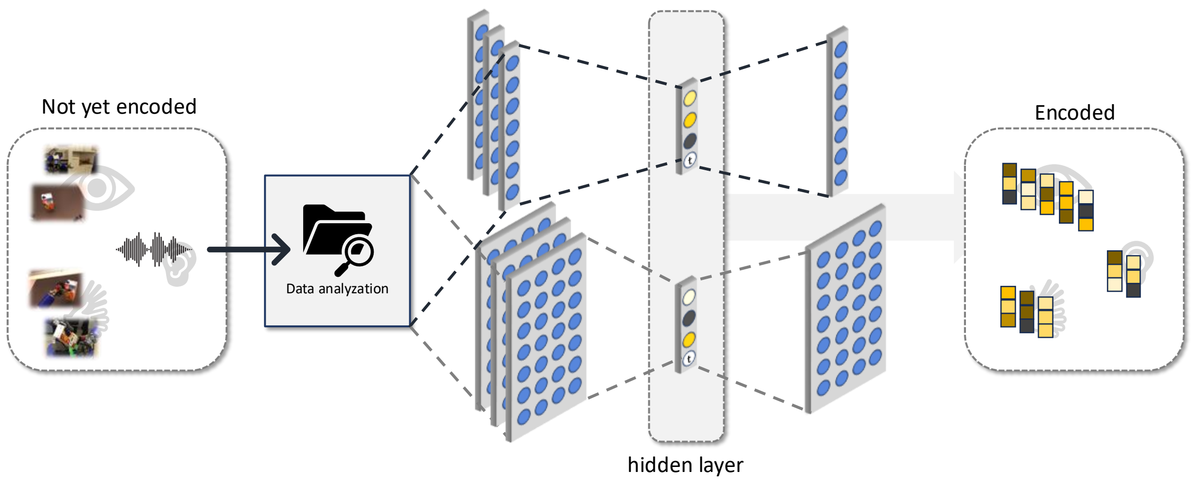


Figure 4.11.: Learning architecture in LTM using the deep episodic memory. At regular intervals, the data collected in the LTM that has not yet been converted into a vector representation is converted using AEs. The data is first analyzed. Multi-dimensional data is converted using a convolutional approach, while one-dimensional data is processed classically. A separate network is trained for each entity. We use windows above the input data in order to map temporal dependencies in the input data. The latent vectors of the input are finally stored in the LTM. The weights of the encoder and decoder are also stored. The encoder is required to convert future data. The decoder is required to reconstruct data from the LTM or to predict data. In order to combine both capabilities in one model, the input data to the decoder is enriched by a value t during decoding. This value indicates whether it is a reconstruction ($t = 0$) or a prediction ($t > 0$).

Symbolic components, on the other hand, may be numerically encoded using techniques such as word embeddings(Bamler and Mandt, 2017). However, the current system does not provide automated conversion mechanisms for symbolic data.

The architecture of the network is defined during training and is then fixed for an entity $\mathcal{E}$. This necessitates a consistent input dimensionality to the encoder network $e_{\mathcal{E}}$. To ensure this, the system verifies the structural consistency of all incoming data. Furthermore, the mapping between vector positions and object members is retained together with the static type, allowing the vectorized representation to be reverted back into an ARON structure.

We differentiate between one-dimensional and multi-dimensional data. For multi-dimensional inputs – such as those containing `NDArrays` with more than two dimensions – specialized models preserve structural information, including spatial relationships. For example, image data necessitates convolutional and pooling layers, while one-dimensional signals (e. g., force measurements) are processed using standard feedforward networks. All models employ a temporal window over the input data to capture dynamic variations. The hidden representation $h(x)$ is constrained to be one-dimensional, with its size either derived from the input dimensionality (up to a maximum of 255 units) or manually specified. The encoded latent vector is defined as $enc = e_{\mathcal{E}}(x) = h(x)$. Representative autoencoder architectures tailored to various entity types are illustrated in Figure 4.11.

To support temporal prediction, this latent vector is augmented with an additional scalar value referred to as the *temporal prediction unit*, which encodes the desired prediction offset

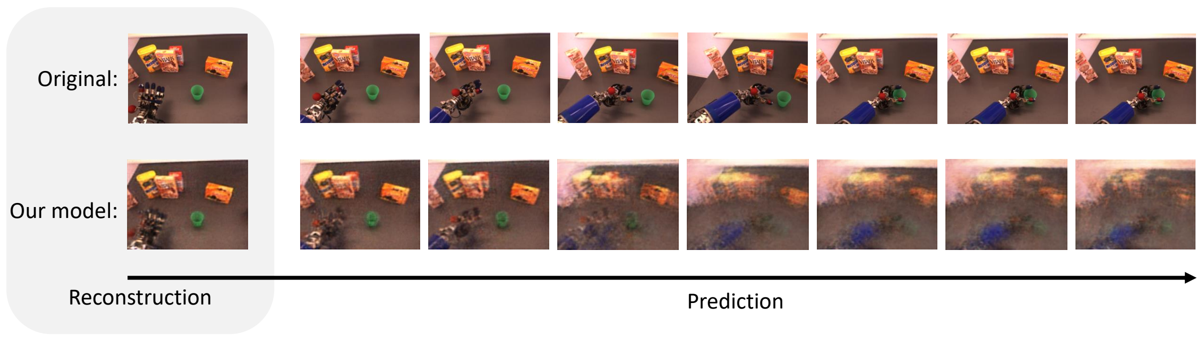


Figure 4.12.: Reconstruction and predictions of the next few frames returned by the deep EM. Only the first original image is encoded into a latent space. A decoder model reconstructs the image or predicts the next frames. ©Elsevier (Peller-Konrad et al., 2023)

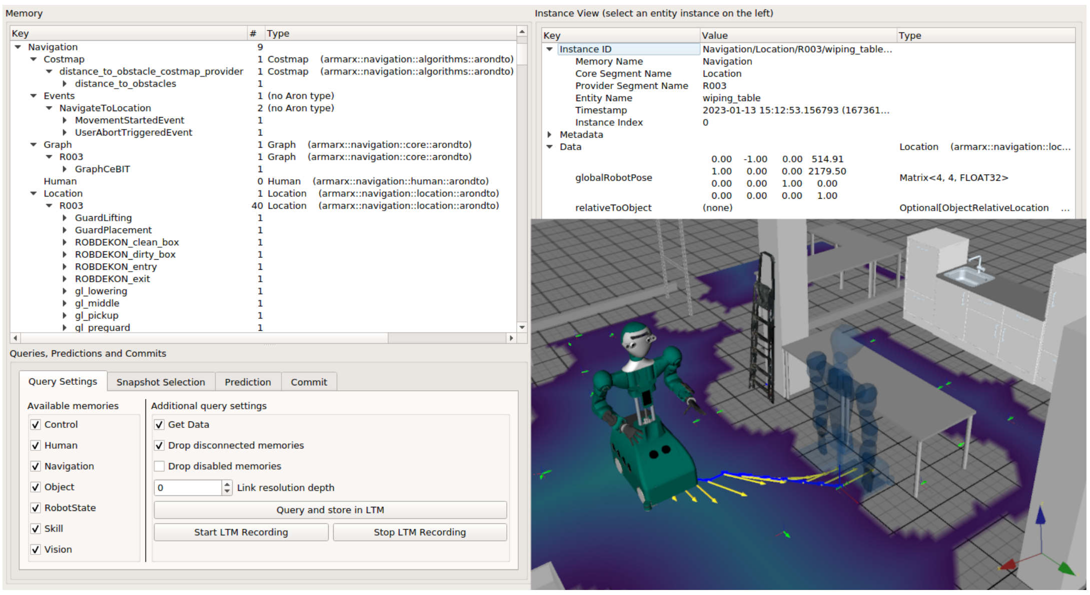
in seconds. For instance, appending a 0 enables exact reconstruction, while appending 0.5 yields a prediction $0.5\,\mathrm{s}$ into the future. Consequently, the decoder network receives input of size $|h(x)| + 1$, enabling it to perform both reconstruction and prediction using a unified model. Unlike autoregressive approaches such as that of Rothfuss et al. (2018), our system can generate multiple future predictions without iteratively feeding prediction errors back into the model.

Due to the dimensionality reduction inherent in auto-encoding, all reconstructions and predictions are lossy, meaning some information is inevitably discarded. The nature of this loss is governed by the training data distribution: the autoencoder optimizes to preserve features that are statistically prominent or relevant for the reconstruction of similar samples, while suppressing less informative variations. This selective retention mechanism constitutes a form of gradual forgetting, in which less relevant details are systematically omitted. However. in datasets with strong internal structure or redundancy, the model uses these patterns to improve the average reconstruction and prediction ability. This fact motivates a precise analysis and early removal of redundant data, so that a dataset that is as balanced as possible is already available for training.

Moreover, because the latent space is regularized to approximate a prior distribution, distances between latent vectors can be interpreted meaningfully. Similar data points are mapped to proximate locations in the latent space, while dissimilar ones are more distant. This property is exploited in our system for various purposes, including filtering operations based on measures such as cosine similarity or MSE, similar to Section 4.5.1 as shown in Algorithm 3. In this way, additional redundant data can be identified by analyzing the proximity of data points in the latent space.

4.6. Graphical User Interface

To support the detailed analysis and inspection of the robot's working memory and long-term memory at specific points in time, a GUI was integrated into the ArmarX framework. This GUI



Memory

```
Key                                               #   Type
▾ Navigation                                      9
  ▾ Costmap                                       1  Costmap   (armarx::navigation::algorithms::arondto)
    ▾ distance_to_obstacle_costmap_provider       1  Costmap   (armarx::navigation::algorithms::arondto)
      ▸ distance_to_obstacles                     1
  ▾ Events                                        1  (no Aron type)
    ▾ NavigateToLocation                          2  (no Aron type)
      ▸ MovementStartedEvent                      1
      ▸ UserAbortTriggeredEvent                   1
  ▾ Graph                                         1  Graph   (armarx::navigation::core::arondto)
    ▾ R003                                        1  Graph   (armarx::navigation::core::arondto)
      ▸ GraphCeBIT                                1
    Human                                         0  Human   (armarx::navigation::human::arondto)
  ▾ Location                                      1  Location   (armarx::navigation::location::arondto)
    ▾ R003                                       40  Location   (armarx::navigation::location::arondto)
      ▸ GuardLifting                              1
      ▸ GuardPlacement                            1
      ▸ ROBDEKON_clean_box                        1
      ▸ ROBDEKON_dirty_box                        1
      ▸ ROBDEKON_entry                            1
      ▸ ROBDEKON_exit                             1
      ▸ gl_lowering                               1
      ▸ gl_middle                                 1
      ▸ gl_pickup                                 1
      ▸ gl_preguard                               1
```

Queries, Predictions and Commits

Available memories
- [✓] Control
- [✓] Human
- [✓] Navigation
- [✓] Object
- [✓] RobotState
- [✓] Skill
- [✓] Vision

Additional query settings
- [✓] Get Data
- [✓] Drop disconnected memories
- [ ] Drop disabled memories
- 0 Link resolution depth

Instance View (select an entity instance on the left)

```
Key                        Value                                          Type
▾ Instance ID              Navigation/Location/R003/wiping_table...
    Memory Name            Navigation
    Core Segment Name      Location
    Provider Segment Name  R003
    Entity Name            wiping_table
    Timestamp              2023-01-13 15:12:53.156793 (167361...
    Instance Index         0
▸ Metadata
▾ Data                                                                    Location   (armarx::navigation::loc...
                             0.00  -1.00   0.00  514.91
    globalRobotPose          1.00   0.00   0.00 2179.50                   Matrix<4, 4, FLOAT32>
                             0.00   0.00   1.00    0.00
                             0.00   0.00   0.00    1.00
    relativeToObject       (none)                                         Optional[ObjectRelativeLocation  ...
```

Figure 4.13.: The memory Graphical User Interface (GUI) and the 3D visualization. On the left, the GUI shows the content of the navigation memory as a tree view. Each entity instance can be investigated as shown on the right. In this example, the view shows the robot's platform location associated with the table. On the bottom right, the content of the memory is visualized in 3D. ©Elsevier (Peller-Konrad et al., 2023)

enables real-time access to the memory contents, not only for the system itself but also for the user, making the robot's internal state fully transparent and accessible. It plays a crucial role in understanding, debugging, and improving memory-based behaviors in our system.

The interface, shown in Figure 4.13, allows users to directly access and interact with the robot's memory by visualizing current and past observations, such as detected objects, environment models, or historical states of the robot. Through this, it becomes possible to inspect the expected internal model of the robot and its environment by querying specific memory entries. The data can be presented in both textual form and through rich visualizations. For instance, memory servers can offer to render their data in a 3D reconstruction of the robot's environment and its own kinematic state, allowing for intuitive assessments of spatial understanding and situational awareness.

Importantly, the GUI also provides temporal introspection capabilities. Users can navigate through time-indexed snapshots of memory to investigate how a particular scene evolved or how an internal belief changed across time steps. This functionality is particularly beneficial during debugging sessions, as it allows for the restoration or examination of the memory state immediately preceding or following a failure event, thereby providing crucial insights into causal relationships and error propagation.

Moreover, the interface supports the exploration and control of predicted future states, giving users the ability to inspect and validate the robot's internal anticipations. Memory server configurations can also be dynamically modified through the GUI, offering a flexible experimentation platform for testing different memory handling strategies. Additionally, memory

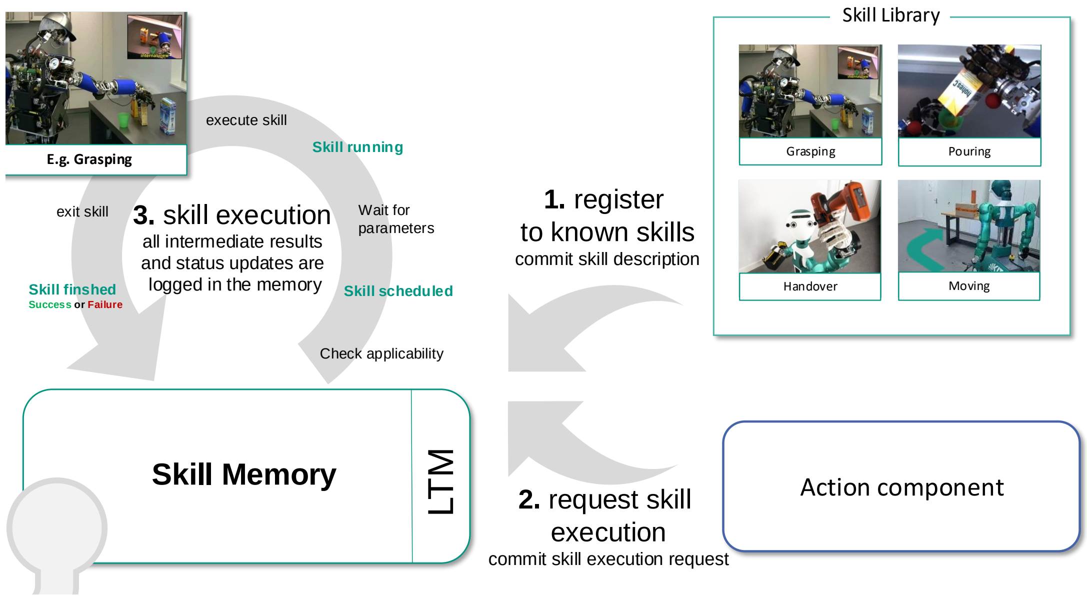


Figure 4.14.: The lifecycle of executable skills that are managed through memory. First, skills are registered to the memory via *Skill Descriptions* from *Skill Providers*. The Skill Descriptions contain information on how to execute a skill, how long the execution probably will take as well as a narrative of the skill. Thus, after some component requests the execution of a skill, a new execution object for the skill is being constructed and executed. All final and intermediate results are stored in memory so that successive skills can use them or for debugging purposes.

contents can be exported as structured data logs, supporting offline analysis and replication of experiments.

4.7. The Representation of Action

Actions are a special data type, as they not only represent knowledge, but must also be executable. While the knowledge of how to perform actions is seen as part of procedural memory, the knowledge of the existence of an action is part of semantic memory. The experience of an execution is first part of the working memory (during the execution) and is finally located in the episodic memory. What is learned from procedural and episodic memory can gradually be converted into explicit knowledge, as also suggested by Sun (2004). For this reason, we will take a closer look at the representation and management of actions in our memory-based architecture.

In the proposed memory-based cognitive architecture, actions are represented following the principle of OACs (Krüger et al., 2011), which encapsulate both the symbolic and sub-symbolic aspects of action execution. These actions are persistently stored in the LTM and are dynamically loaded into the *Skill Memory* upon system initialization. Within this memory server, actions are instantiated as executable entities referred to as skills. This architecture enables the seamless integration of declarative knowledge and procedural execution by linking memory representations with executable codelets. The system ensures that skills can be instantiated and executed adaptively in response to environmental stimuli, internal cognitive processes, or direct user commands.

Skill execution is governed by *Skill Providers*, which serve as intermediaries that register executable codelets in the Skill Memory. Each Skill Provider encapsulates a set of related skills, grouping them according to their functional domain. Skills are parameterizable by arbitrary ARON objects and may be defined at varying levels of abstraction, ranging from high-level symbolic descriptions (e. g., “Navigate to the table”) to precise sub-symbolic formulations (e. g., “Move to coordinates (10,10)”) (Rosenthal et al., 2016). The instantiation and execution of a skill are contingent on the fulfillment of predefined activation conditions. The simplest activation mechanism is a direct execution request, but more complex conditions may involve event-driven triggers, such as object recognition in the context of manipulation tasks.

The execution of a skill follows a predefined lifecycle, shown in Figure 4.14, to ensure systematic processing and controlled termination. Upon invocation, a skill transitions through distinct execution phases, beginning with the construction and initialization phase, where the execution object is constructed and statically available necessary resources are allocated (such as loading computational models or performing preparatory operations). A skill can be requested even though all the required dynamic resources (given by input parameters) are not yet available. The initialization phase ensures that complex work can already be carried out while waiting for all input parameters. Once initialized, the skill enters the scheduled phase, where it awaits the provision of all required execution parameters. When all parameters have been specified, the skill transitions into the running phase, wherein its primary execution logic is performed. Following execution, the skill enters the exiting phase, where it releases allocated resources and finalizes outstanding processes before reaching a terminal state. Multiple skills (even of the same entitiy) can be run in parallel.

Throughout its lifecycle, a skill may be subject to external interruptions. It can be aborted through an explicit termination request, timeout if execution exceeds a predefined temporal threshold, or fail due to errors encountered during execution. The architecture ensures that, irrespective of the termination condition, the exiting phase is entered to guarantee a controlled shutdown and resource deallocation. The final outcome of a skill execution consists of a structured result, which includes either a success confirmation or an error message in the case of failure. Additionally, a termination event is generated, typically indicating successful completion but extendable to more complex outcome representations. This event, alongside associated data fields, facilitates inter-skill communication, allowing subsequent skills to adapt their behavior based on the execution result of preceding skills.

All execution results, including intermediate and final outcomes, are stored in the “Skill” memory server. This ensures that execution traces can be leveraged by subsequent skills for contextual adaptation and decision-making. Furthermore, the recorded data serves as a valuable resource for system debugging and post-execution analysis, facilitating the refinement of skill execution strategies. By maintaining a comprehensive execution history, the architecture enhances robustness, supporting both autonomous learning and system transparency.

The proposed skill execution and management framework is designed to be dynamic and adaptive. While a foundational set of core skills is loaded from prior knowledge during system initialization, the framework allows for skills to be added, removed, enabled, or disabled at runtime. This modular and flexible design ensures that the system remains extensible and capable of adapting to evolving task requirements and environmental constraints. The skill representation in memory is structured to encode symbolic descriptions, execution parameters, and associated conditions, enabling the architecture to bridge the gap between high-level cognitive representations and low-level motor execution. Through this approach, the memory-based architecture facilitates robust and scalable action execution in autonomous robotic systems.

CHAPTER 5

Evaluation of Memory System

Disclaimer

Parts of the content presented in this chapter were previously published in

- F. Peller-Konrad, R. Kartmann, C. R. Dreher, A. Meixner, F. Reister, M. Grotz, and T. Asfour (2023). “A memory system of a robot cognitive architecture and its implementation in ArmarX”. In: *Robotics and Autonomous Systems* 164, pp. 1–20
- J. Plewnia, F. Peller-Konrad, and T. Asfour (2024). “Forgetting in episodic long-term memory of humanoid robots”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Yokohama, Japan, pp. 6711–6717
- J. Plewnia, F. Peller-Konrad, and T. Asfour (2025). “Beyond recall: evaluating forgetting mechanisms for multi-modal episodic robotic memory”. In: *German Robotics Conference (GRC)*. Nuremberg, Germany

This chapter describes the implementation and results of the evaluation of our memory system. First, we will explain general evaluation criteria used to evaluate memory and forgetting methods in related work. Next, we will discuss the evaluation of working memory. Our system is compared with direct Peer-to-Peer (P2P) requests and Publish-Subscribe (PS) services.Finally, long-term memory will be evaluated in two different studies. On the one hand, deep episodic memory is analyzed for its ability to be reconstructed and predicted using various forgetting methods. Secondly, the entire memory, including various forgetting methods, is tested for the probability of success using a sample task. All results are then discussed in the respective sections.

5.1. Evaluation Criteria

We distinguish evaluation criteria, which cover the definition of evaluation scenarios, from evaluation metrics. On the one hand, selecting the right evaluation scenarios is crucial for a complete and accurate evaluation. If a system is to be used for a specific task, the evaluation scenarios should preferably come from a suitable domain. On the other hand, evaluation metrics determine which data is collected during the evaluation and then compared against a reference system.

5.1.1. Evaluation Scenario Definition

To categorize evaluation scenarios effectively, we discuss key criteria for assessing memory and forgetting. One fundamental criterion is the **type of environment** used in the evaluation scenario. Evaluation methods can be categorized into real-world experiments and simulated settings. Studies such as Waibel et al. (2011) use real robots to assess LTM transfer, while Sánchez et al. (2015) evaluate memory through human-robot conversations. Others, like Derbinsky and Laird (2009) and Pasukonis et al. (2022), employ simulated environments to assess memory via game performance. Similarly, forgetting mechanisms are often evaluated in simulated games (Reitter and Lebiere, 2012; Wang et al., 2012) or with simulated robots (Sigalas et al., 2017), though real-world experiments incorporating sensory inputs (Buoncompagni and Mastrogiovanni, 2019; Freedman and Adams, 2011) provide valuable complementary insights. Studies such as Plewnia et al. (2024) combine real and simulated environments, demonstrating that forgetting mechanisms may perform differently across contexts.

Another crucial classification involves the **level of embodiment** of the cognitive architecture. Most studies, especially in the context of robotics, focus on embodied architectures (Derbinsky and Laird, 2013; Pasukonis et al., 2022). Other examples are dataset-driven works like Grauman et al. (2022); Ragusa et al. (2023), where egocentric data is collected. Since egocentric recordings with temporal and spatial annotations were recorded, we consider these works analogous to memory, which stores data in plain text. Evaluations of forgetting mechanisms in humanoid robots remain rare, often relegating robots to data collection roles rather than active participation (Buoncompagni and Mastrogiovanni, 2019; Freedman and Adams, 2011). Non-embodied architectures, such as those in (Reitter and Lebiere, 2012; Wang et al., 2012), typically operate in simulated environments.

Scenario duration also plays a key role. Shorter evaluations often focus on forgetting mechanisms, whereas longer studies assess memory retention. Durations range from minutes (Idrees et al., 2020) to years (Nelson et al., 2016), with large datasets like Ego4D (Grauman et al., 2022) providing extensive video recordings. Forgetting studies vary similarly, from a few minutes (Reitter and Lebiere, 2012) to multiple days (Sigalas et al., 2017), impacting the observed performance of forgetting mechanisms.

The **evaluation tasks** exhibit significant diversity. Learning-based assessments for Episodic Memory include clustering (Yang et al., 2021), action recognition (Ragusa et al., 2023), and object localization (Idrees et al., 2020). Question-answering tasks (Bärmann et al., 2021; Grauman et al., 2022) and game-based evaluations (Derbinsky and Laird, 2009; Pasukonis et al., 2022) are common. Navigation tasks (Endo, 2008; Nelson et al., 2016) measure Episodic Memory's role in path planning and map-building. For forgetting, common evaluation methods include object classification (Logacjov et al., 2021), text-based alignment (Scheutz et al., 2019), and retrieval accuracy (Reitter and Lebiere, 2012; Wang et al., 2012).

Anticipation tasks are another example, assessing predictive capabilities in EM. Auto-encoders are used for future frame prediction (Rothfuss et al., 2018), while Grauman et al. (2022) propose forecasting benchmarks. Similarly, forgetting evaluations examine retrieval accuracy under imperfect memory conditions, such as event retrieval in noisy environments (Wang et al., 2012).

Lastly, **data continuity** influences both EM and forgetting studies. Some studies record continuous memory snapshots, such as Derbinsky and Laird (2009), while others collect non-continuous datasets like Waibel et al. (2011) and Grauman et al. (2022). Continuous recording introduces challenges related to storage, inactive periods, and data processing, affecting both episodic memory retention and the efficacy of forgetting mechanisms.

5.1.2. Evaluation Metrics

Evaluating memory systems in artificial cognitive architectures involves several critical metrics, including processing speed, retrieval time, and storage capacity.

First of all, **retrieval** and **commit time** are critical metrics that measure how quickly a cognitive system can access or create information. Fast times are essential for real-time applications, such as robotics and interactive systems, where delays can hinder performance (Jaime et al., 2012). The same applies to the **availability and responsiveness** of the system, as it reflects the system's ability to react promptly to stimuli and process information effectively. Findings support the notion that cognitive mechanisms prioritize quick and predictive responses to unexpected events, thereby enhancing the overall responsiveness of the system (Engström et al., 2022).

Storage capacity refers to the amount of information that a cognitive architecture can retain. This metric is vital for ensuring that the system can handle complex tasks requiring extensive knowledge (Freedman and Adams, 2011; Plewnia et al., 2024). In Working Memory, capacity generally only plays a subordinate role, whereas in Long-term Memory, capacity is essential, especially in lifelong learning systems, in order to store as many experiences as possible. There, capacity is used to evaluate the data savings of different forgetting methods while maintaining other metrics.

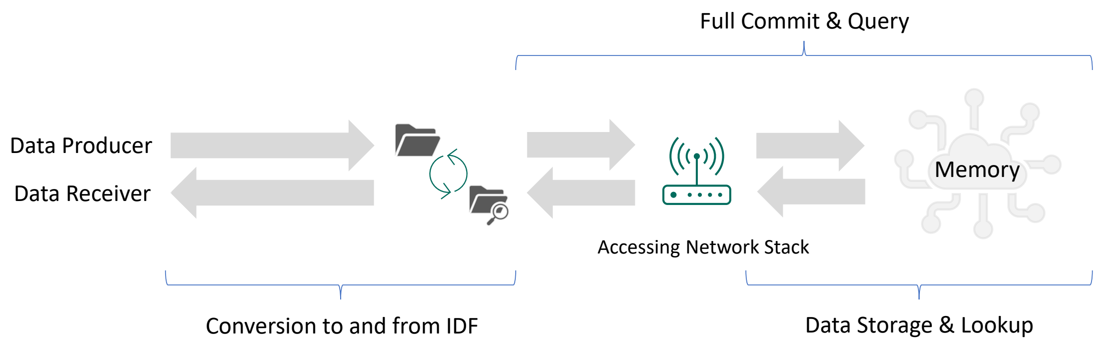


Figure 5.1.: Overview of data transfer to and from the memory from arbitrary components. We evaluated our memory by measuring the times required to commit to and query data from the memory. ©Esevier (Peller-Konrad et al., 2023)

One of such metrics is **task performance**. This includes assessing the accuracy and efficiency of the system in executing cognitive functions. At a high level, one could explore how integrating Episodic Memory into cognitive architectures can enhance task performance by allowing robots to utilize past experiences (Bärmann et al., 2024b; Vinanzi et al., 2019). At the data level, it must be analyzed to what extent the encoding or prediction of the data has an influence on the feasibility or success rate of a task. Forgetting methods can be evaluated similarly (Plewnia et al., 2024).

Adaptability of memory systems is a crucial metric that reflects how well a cognitive architecture can adjust to new information and changing environments. This adaptability is often linked to the architecture's ability to learn and update its structure based on experiences (Li and Laird, 2013; Rothfuss et al., 2018).

Usually, not one but several metrics are used for evaluation. Evaluating memory systems in artificial cognitive architectures requires a multifaceted approach that considers retrieval time, responsiveness, storage capacity, task performance, and adaptability. These metrics collectively provide a comprehensive understanding of how well these systems can emulate human cognitive functions and respond to complex tasks in dynamic environments.

5.2. Working Memory

In our system, the Working Memory provides data with high availability in plain text. Responsiveness and availability are therefore essential for the overall system. Since no encoding and no forgetting methods are used in WM, we only evaluate these properties. In the following, we discuss the evaluation environment used, the data collection and the results.

5.2.1. Methodology

In this evaluation, we measure the time a producer needs to convert business objects and send data-objects to the memory, and how long a consumer needs to receive and convert the information back after being notified by the memory. Thus, in this evaluation, the commit and retrieval times as well as the availability and responsiveness of the system, are measured and evaluated. A clear scenario is not evaluated.

Since all components run in different processes and maybe on different machines, the chosen middleware requires time to serialize the information of the DTO, access the network stack, transfer the information to the remote machine, and finally deserialize the information. We treat the middleware as a black box system, thus, we have no influence on the times required for each of those steps. In this evaluation we use ZeroC Ice (Henning, 2004). Because our memory is designed independent of the chosen middleware, it could also be replaced by e. g., ROS, which has lower mean latency values for the transmission of data. The procedure of sending data to and receiving data from the memory is shown in Figure 5.1.

For the evaluation we define three different data objects with an increasing complexity taken from real-use implementations in ArmarX.

1. A simple object only containing a single long value ($8\,\mathrm{B}$ of information).
2. A slightly larger moderate object containing a long, a string (with a fixed length of five characters), and a memory id ($33\,\mathrm{B}$ of information)
3. A complex object containing memory links, nested objects, an image with resolution $128 \times 128 \times 3$, and several maps with fixed size. In total, this object contains $49.225\,\mathrm{B}$ of information.

For each test, we calculated the means and variances over 1.000 samples. Further, we performed each test either with one single data entry or with a batch of 20, 50 or 100 entries.

We evaluate the worst case, which means that all data must be sent over the network to a different pc. All measurements were performed on computers equipped with an AMD Ryzen 9 5900X with 12 cores, $64\,\mathrm{GB}$ RAM and a NVIDIA GeForce RTX 3060 running Ubuntu 20.04.

5.2.2. Results

The time required to commit information to memory is illustrated in Figure 5.2. The “Full Commit” diagram represents the time taken to commit data objects to memory, including the network overhead introduced by the middleware. In contrast, the “Data Storage” diagram exclusively depicts the time required to insert the received information into the Working Memory structure.

As expected, the time required to convert and commit large batches of objects increases linearly with data size. However, when transferring small amounts of data over the network, a minimum cost associated with accessing the network stack is observed. Even for small object transfers, a minimum delay of $100\,\mu\mathrm{s}$ is required. Additionally, the time to convert and transfer moderate-sized data batches is closer to that of complex data, despite the moderate data size being more similar to the simple type. It is important to note that the majority of the

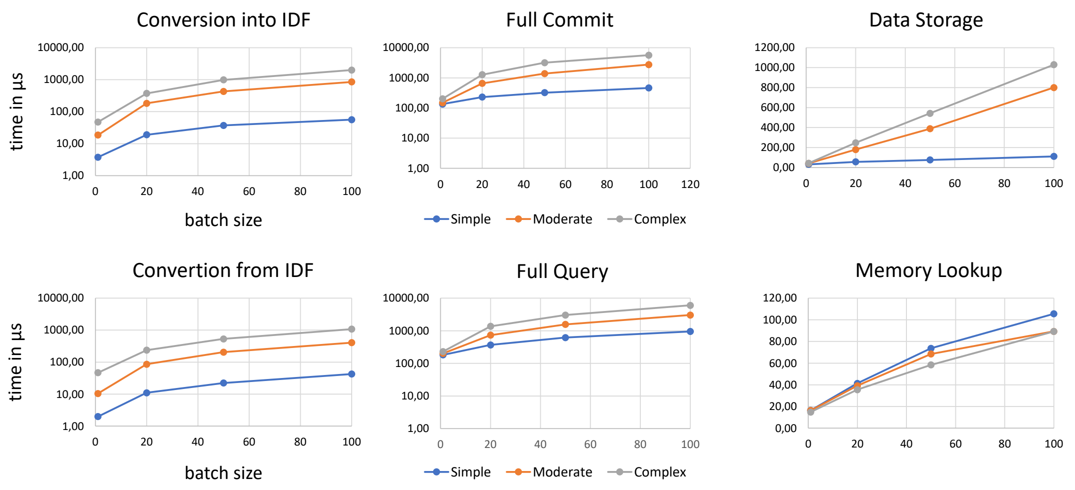


Figure 5.2.: Measured times to send (top row) or query (bottom row) a single or multiple instances to the memory, including the time to convert the auto-generated business objects to ARON. We calculated the mean values over 1000 individual samples per data type and per batch size. Note that the first two charts on each row have a logarithmic scale on the y-axis. ©Esevier (Peller-Konrad et al., 2023)

complex type's data volume originates from an image ($49.225\,\mathrm{B}$), which is converted into a single `NDArray`, allowing for more efficient conversion and transfer. Apart from the image, the complex type is approximately twice the size of the moderate type, a trend that is also reflected in the comparative transfer durations of both experiments.

The conversion time of a BO to its DTO representation is influenced not only by data size but also by the number of members, as each member requires the allocation of new objects, and the batch size, as the writer object must be reallocated for every instance. Consequently, the conversion of a moderate-type object is approximately 15 times slower than that of a simple one. Furthermore, the moderate and complex objects have a comparable number of members, further influencing conversion time.

The time required to arrange information in memory is primarily dependent on data size. In these experiments, a single memory server with an initial dataset of 1000 entries was used to ensure comparability of insertion times. This setup also better reflects real-world usage scenarios, where pre-existing data is common. A small static overhead is observed in accessing the memory structure and inserting commits during experiments. However, the results for batched experiments confirm the aforementioned dependency on data size.

Once the working memory system receives new data, a notification is broadcast to all subscribers, who may then query the information to retrieve the newly available data. The time required to query information is also depicted in Figure 5.2. The "Full Query" chart illustrates the time required to query and receive data from a memory server without converting it from ARON to its BO representation, whereas the "Memory Lookup" chart displays the time needed to locate a specific entry within memory.

Similar to the process of committing information over a network, querying is influenced by both data size and batch size, while also exhibiting a static overhead due to network stack

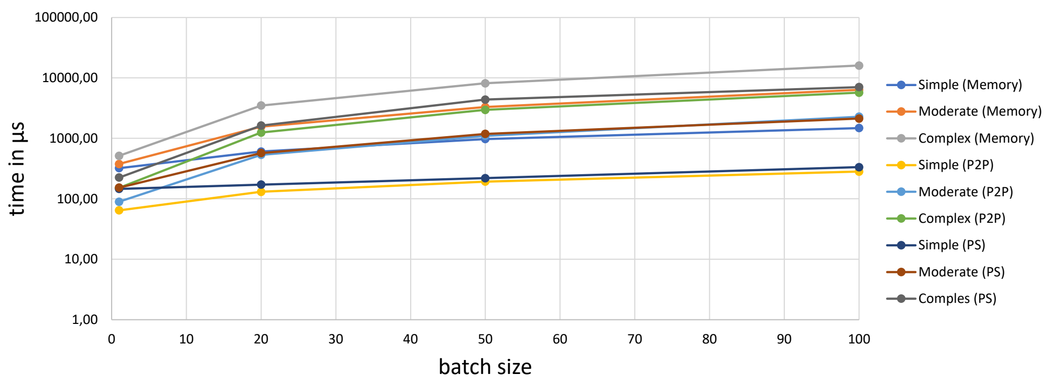


Figure 5.3.: Averaged age of data as difference of timestamps between data reception and production, either using our memory system, P2P connections or PS channels provided by the middleware ZeroC Ice. ©Elsevier (Peller-Konrad et al., 2023)

access. The time required to search for instances within memory remains relatively consistent across experiments with the same batch size, as the memory server primarily returns references to stored information. During a query, the retrieved data is automatically converted back into its BO representation. This conversion process exhibits a similar dependency on both the number of members and data size as observed during the initial transformation.

To summarize the findings of this experiment, the total time from data production to its reception in the consumer component was compared for direct P2P communication, PS channels managed via a centralized service, and memory-based storage, as shown in Figure 5.3. The P2P communication time represents a theoretical lower bound for distributed data transmission. In contrast, a PS service behaves similarly to memory in that it receives data from producer components and forwards it to multiple subscribers. However, unlike memory, it lacks the capability to provide historical data access or active data retrieval.

Among the three methods, memory exhibits the longest processing times. In contrast, it provides essential functionalities for cognitive systems, such as querying past information and offering general data-driven services to clients. Moreover, this evaluation represents a worst-case scenario. In practical implementations, producers, memory servers, and consumer processes are often integrated within the same system or executed on the same machine, reducing network transfer overhead. In many cases, data transfer times can be reduced by half through an optimized component distribution strategy. Nevertheless, even under the measured latencies for moderate data types, retrieval from memory can occur at frequencies exceeding $160\,\mathrm{Hz}$, which is sufficient for state-of-the-art model predictive control algorithms, such as those described in Bhardwaj et al. (2022). For components requiring higher access frequencies, such as self-collision detection and emergency stop mechanisms, alternative communication methods such as P2P or PS channels remain viable solutions.

5.3. Deep Episodic Memory

In our system, deep episodic memory is used to store data in a compressed form that allows later reconstruction and prediction. It is a learning system that uses past experience to calculate the compression. If less data is available for learning due to one or more forgetting methods, this can have a positive or negative influence on the learning ability depending on what has been forgotten. For example, unique and important data can be forgotten. This diminishes the ability of the deep EM to reconstruct or predict this kind of information. On the other hand, redundant data can also result in overfitting, which could be mitigated by forgetting the surplus data points, thus balancing the dataset. In the following, we evaluate the deep episodic memory for reconstruction and prediction capability with partially incomplete input data from online filtering. In addition, we examine the extent to which incomplete input data influences the regularization of the learned encoding used in offline filtering.

5.3.1. Methodology

We evaluate the metrics storage capacity and task performance using two different datasets:

1. Real-world recordings using the humanoid robot ARMAR-6, where the robot was asked to solve different tasks. Initially, the robot observes a human worker to determine when assistance is required in warehouse tasks. These tasks include helping in lowering a conveyor belt cover, collaboratively transporting it with the human to a secondary position, grasping, transporting, and positioning a ladder, retrieving objects, executing handovers, placing objects, and assisting in inserting the conveyor belt cover. Furthermore, real-world data was collected during robot maintenance, testing, and exploratory tasks, wherein ARMAR-6 autonomously navigated and interacted with its environment. Data collection was conducted across three sessions spanning $11\,\mathrm{days}$, capturing variations in external conditions such as lighting changes and different human interactions. In total, 6 episodes with an overall length of $2.2\,\mathrm{h}$ of real-world data were recorded, encompassing visual data from the robot's RGB and RGB-D cameras, proprioceptive data, and object interaction information. The episodes have different lengths, ranging from $5\,\mathrm{min}$ for the first episode to $1\,\mathrm{h}$ for the fourth episode, all recorded in the same environment but with different objects, people, lighting, and performed tasks.
2. Real-world recordings from the 20 Billion Something-Something (20BN) dataset, a large collection of labeled video clips that show humans performing pre-defined basic actions with everyday objects. These tasks encompass placing objects adjacent to one another or positioning them on a surface. The dataset includes a diverse range of scenes, enabling the evaluation of forgetting mechanisms across a broad spectrum of memory snapshots derived from scene images.

Both data sets originate from real recordings. While the recordings of ARMAR-6 exactly match the target scenarios in terms of embodiment, duration or continuity of the data, we use the

20BN dataset to test our system with large amounts of data. However, this is only video data without motion information. Also, many of the videos can only be interpreted as very short episodes with a duration of a few seconds. The recordings of ARMAR-6 were significantly longer and more coherent in comparison. Finally, the external dataset lacks metadata, which is why default values for timestamps or query frequency were assumed.

The recordings of ARMAR-6 are quite similar, which allows a randomized selection of training and test episodes. The recordings of the 20BN dataset, on the other hand, are very different from each other. For this reason, the last 20% of a training episode is used as test data and 1800 randomly chosen episodes were used for training (around 67000 images).

For the evaluation of online forgetting methods on the basis of the deep episodic memory, we first compare the compression, reconstruction and prediction ability of an instance trained with incomplete data with those of a deep EM that had all data available for training, in the following called *baseline* model. We use the methods described in Section 4.5.1. The reconstruction task is comprised of passing an episode (not known from training) to the deep EM for encoding and subsequent decoding. The reconstruction performance can then be measured by comparing the original memory snapshots to the result of the reconstructions. Typical quantitative evaluation metrics are Peak-Signal-to-Nnoise-Ratio (PSNR) and Structural Similarity Index Metric (SSIM) (Wang et al., 2004) from which we focus on PSNR. The prediction task consists of passing an episode and a time-delta value to the deep EM. The deep episodic memory transforms the memory snapshots into latent space representation, which is then used to predict a memory snapshot that will occur after the specified amount of time. This task can be evaluated by comparing a predicted memory snapshot with the original one that followed the given input after the specified time. Similar to the reconstruction task, this can be quantitatively evaluated by leveraging PSNR.

In order to evaluate the regularization ability of deep episodic memory, as used in offline forgetting, we use the multi-stage training process (also shown in Algorithm 4). First, the (empty) deep episodic memory model is trained with a first batch of data. This data can then be filtered using the learned similarity function. Once new data (batch 2) reaches the system, the network requires retraining. To do this, we use the new data and the filtered data from the first batch for training. The resulting new deep episodic memory can now be used to evaluate the similarity-based filtering of the data from batch 1. Similar to the evaluation of online forgetting, the reconstruction and prediction capability (using PSNR) of the retrained network is then compared to the baseline model using a third batch of unseen data.

All experiments in this section were performed on a computer with an AMD Ryzen 9 3900X 12-Core Processor, an NVIDIA GeForce RTX 3060, and $80\,\mathrm{GB}$ RAM .

5.3.2. Results

We evaluate online frequency-based forgetting mechanisms as described in Section 4.5.1 for four different parameters $\delta_{wait} \in \{100\,\mathrm{ms}, 250\,\mathrm{ms}, 500\,\mathrm{ms}, 1000\,\mathrm{ms}\}$, resulting in memory

Algorithm 4 Multi-stage training of deep episodic memory

```
**Given**:
$\mathcal{E}_{enc}$ – all encoded snapshots of entity $\mathcal{E}$ (initially empty)
$\mathcal{E}_{batch\ 1}$ – first batch of data of entity $\mathcal{E}$
$\mathcal{E}_{batch\ 2}$ – second batch of data of entity $\mathcal{E}$
$\mathcal{E}_{batch\ 3}$ – third batch of data of entity $\mathcal{E}$
$train_{\mathcal{E}}(X, \mathcal{L})$ – training function of deep episodic memory for entity $\mathcal{E}$ and error $\mathcal{L}$
$enc_{\mathcal{E}}(X)$ – encoding model for entity $\mathcal{E}$
$rec_{\mathcal{E}}(X)$ – reconstruction model for entity $\mathcal{E}$
$pred_{\mathcal{E}}(X, t)$ – prediction model for entity $\mathcal{E}$

**Procedure**:
$\mathcal{L} = mse$ – or another loss function
$train_{\mathcal{E}}(\mathcal{E}_{batch\ 1}, \mathcal{L})$ – updates model weights
apply Algorithm 2 with $\mathcal{E}_{batch\ 1}$ as $\mathcal{E}_{ltm\ new}$
apply Algorithm 3
$\mathcal{E}_{filtered\ batch\ 1} = rec_{\mathcal{E}}(\mathcal{E}_{enc})$
$train_{\mathcal{E}}(\mathcal{E}_{filtered\ batch\ 1} + \mathcal{E}_{batch\ 2}, \mathcal{L})$
evaluate $rec_{\mathcal{E}}$ and $pred_{\mathcal{E}}$ based on $\mathcal{E}_{batch\ 3}$

```

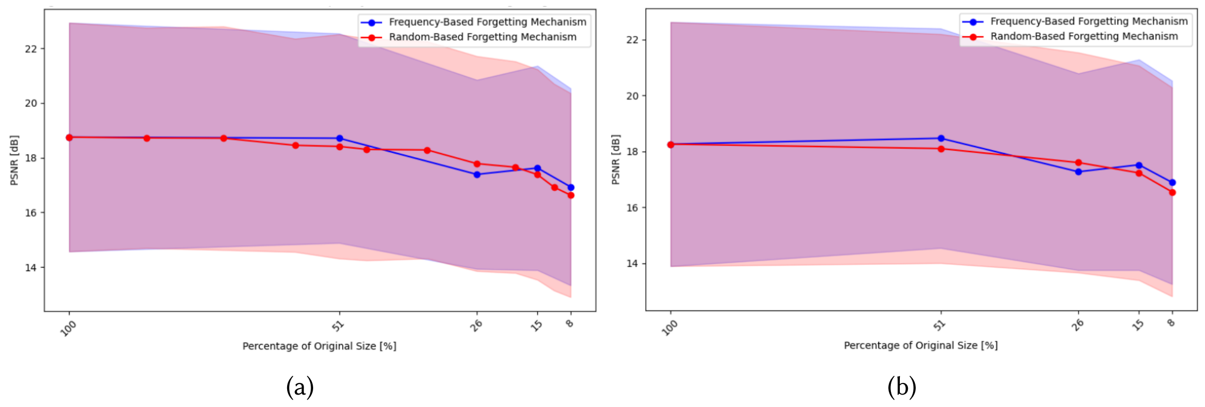


Figure 5.4.: Average PSNR values of the reconstruction (a) and prediction task (b) for different thresholds of online frequency-based forgetting mechanisms, compared to a random forgetting strategy. Higher thresholds of δ_{wait} lead to smaller episodic memory sizes and smaller episodic memory sizes lead to worse average PSNR values for reconstructed or predicted images.

sizes of $size \in \{51, 26, 15, 9\}\%$ of the original episodic memory size. In addiiton to the baseline model, we use random-based forgetting mechanisms as a comparison model, where the same number of snapshots was randomly removed.

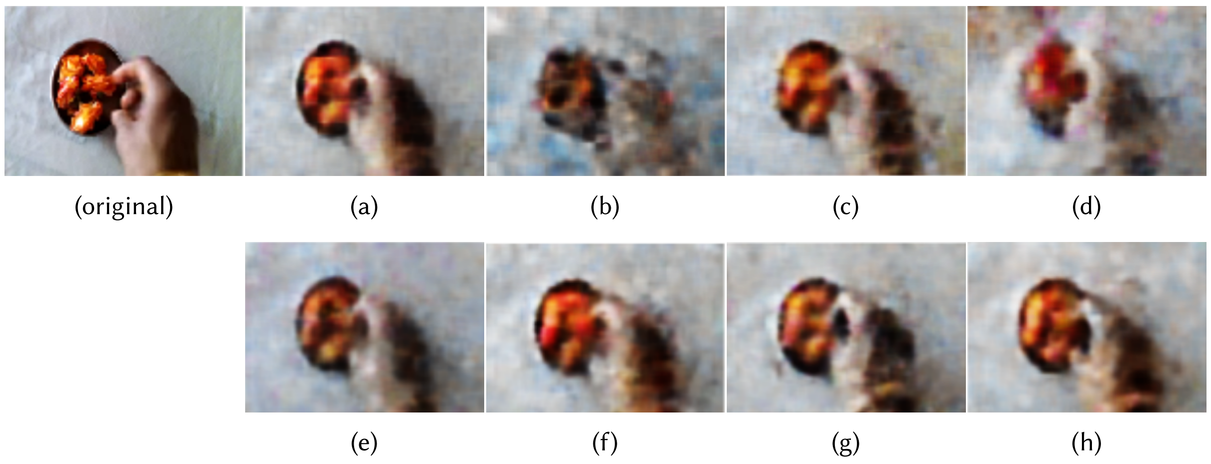


Figure 5.5.: Example of a reconstructed entity snapshot from the 20BN dataset. The model was trained on a filtered dataset using frequency-based forgetting with thresholds $\delta_{wait} = 100\,\mathrm{ms}$ (a), $\delta_{wait} = 250\,\mathrm{ms}$ (b), $\delta_{wait} = 500\,\mathrm{ms}$ (c), $\delta_{wait} = 1000\,\mathrm{ms}$ ((d)). For comparison, reconstructions from a randomly filtered dataset using probabilities $p = 0.51$ (e), $p = 0.26$ (f), $p = 0.15$ (g), $p = 0.09$ (h) is shown in the bottom row.

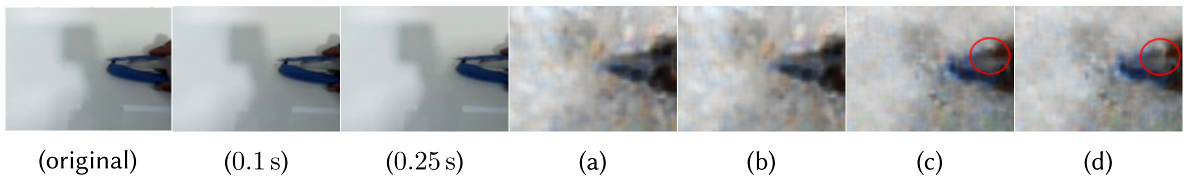
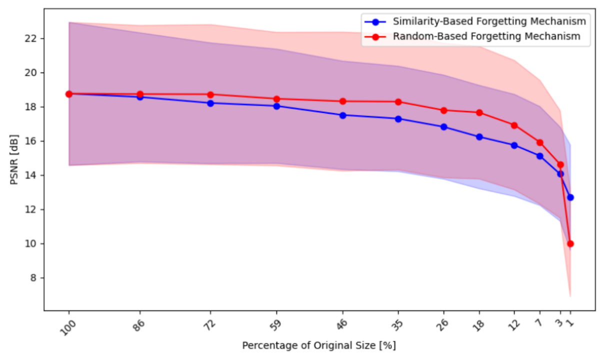


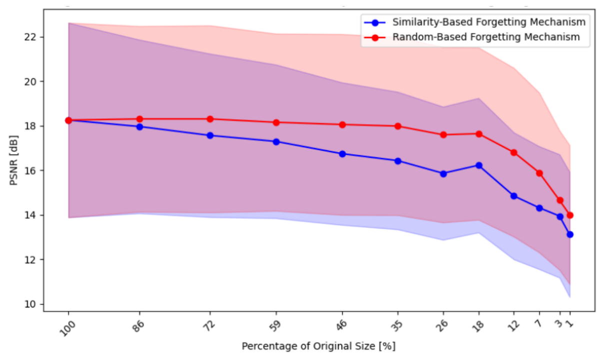
Figure 5.6.: Example of future frame prediction for two time delta values on the 20BN dataset: $0.1\,\mathrm{s}$ and $0.25\,\mathrm{s}$ seconds into the future. (a) (prediction of $0.1\,\mathrm{s}$) and (b) (prediction of $0.25\,\mathrm{s}$) show the results for a model trained on a memory using frequency-based forgetting with δ_{wait} = $250\,\mathrm{ms}$. (c) and (d) show the results for a model trained on an episodic memory using random-based forgetting with $p = 0.26$, thus achieving the same memory sizes. The red circle marks the small difference between the results.

20 **Billion Something-Something Dataset**

A comparison of the PSNR values of the baseline model, random-based forgetting and frequency-based forgetting on the 20BN dataset is shown in Figure 5.4. On average, the reconstruction and the prediction values for random- and frequency-based forgetting decrease slightly with a decreasing episodic memory size. However, due to the difference between the individual episodes, a high variance is shown. For a specific episode, we sometimes notice that an initial increase of the threshold $\delta wait$ may first degrade a model's reconstruction capability for a specific image. Later increments, however, could increase the reconstruction capability for the episode, illustrating a non-linear relationship between threshold level and reconstruction quality as shown in Figure 5.5. This behavior is expected, as both random- and frequency-based forgetting does not take into account the significance of an input, and therefore, important data may be forgotten. The prediction capability is highly dependent on δ_{wait}. If δ_{wait} is set too high, the model is not able to predict to time deltas $< \delta_{wait}$ as no training data is available. This does not happen when using a random-based forgetting



(a)



(b)

Figure 5.7.: Average PSNR values of the reconstruction (a) and prediction task (b) for different thresholds of online similarity-based forgetting mechanisms, compared to random-based forgetting.

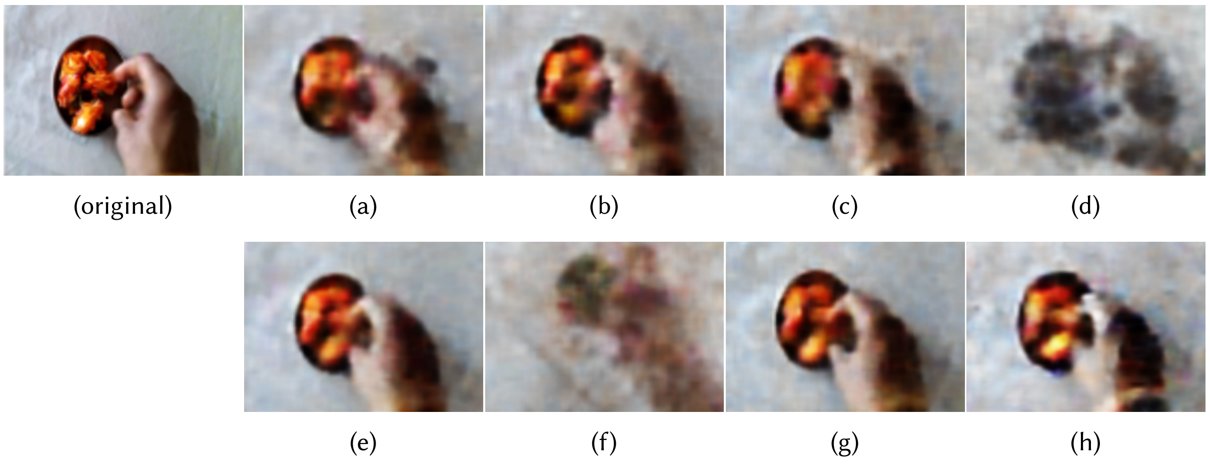


Figure 5.8.: Example of a reconstructed entity snapshot from the 20BN dataset. The model was trained on a filtered dataset using similarity based forgetting with thresholds $d_{min} = 10$ (a), $d_{min} = 30$ (b), $d_{min} = 50$ (c), $d_{min} = 70$ (d). For comparison, reconstructions from a randomly filtered dataset using probabilities $p = 0.86$ (e), $p = 0.59$ (f), $p = 35$ (g), $p = 12$ (h) is shown in the bottom row.

mechanism. An example of this is shown in Figure 5.6. While the reference prediction differs for $0.1\,\mathrm{sec}$ and $0.25\,\mathrm{sec}$, the actual predictions are very similar for the frequency-based model using $\delta_{wait} = 250\,\mathrm{m\,sec}$. The random-based model with $p = 0.26$ correctly anticipates a movement of the stapler.

We evaluate online similarity-based forgetting mechanisms for the parameters $d_{min} \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$, where d_{min} denotes the minimal dissimilarity a snapshot needs to exhibit compared to the latest accepted snapshots to be consolidated as described in Section 4.5.1. These parameters result in episodic memory sizes of $size \in \{86, 72, 59, 46, 35, 26, 19, 12, 7, 3, 1\}\%$ of the original episodic memory size. Similar as for frequency-based forgetting, we use the baseline model and random-based forgetting for comparison of the PSNR values of the reconstruction and prediction task. Still, when reducing the size of the episodic memory using forgetting, the performance of the model to reconstruct or predict information decreases significantly, as shown in Figure 5.7, whereby on aver-

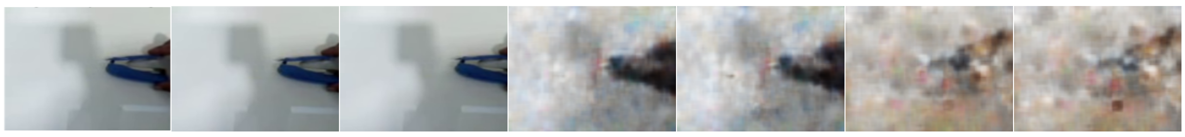


Figure 5.9.: Example of future frame prediction for two time delta values: $0.1\,\mathrm{s}$ and $0.25\,\mathrm{s}$ seconds into the future. (a) and (b) show the results for a model trained on a memory using similarity-based forgetting with $d_{min} = 10$ and (c) and (d) with $d_{min} = 30$.

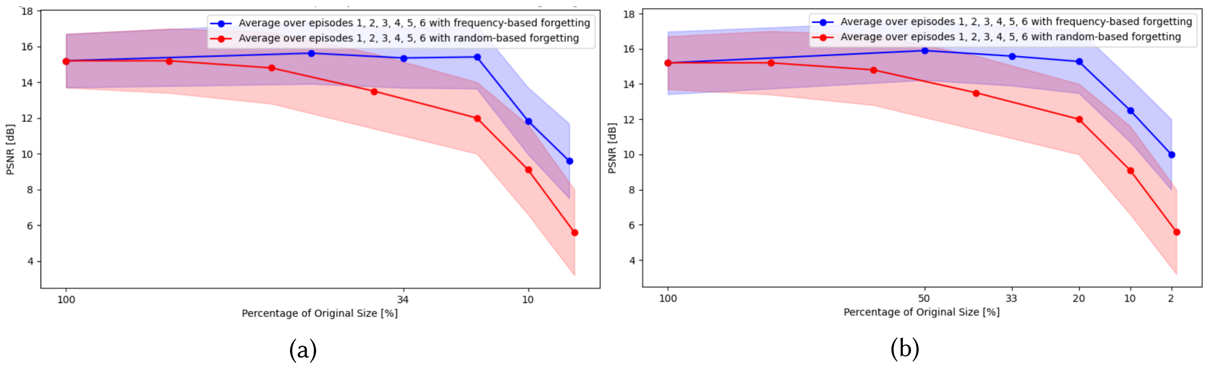


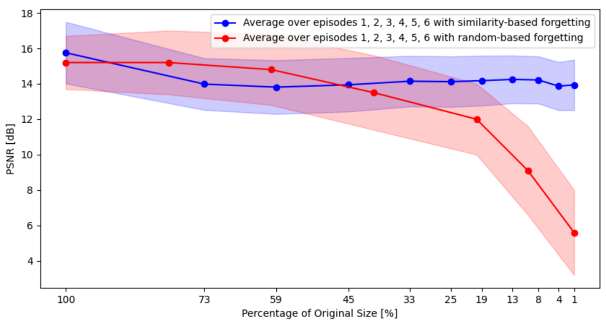
Figure 5.10.: Average PSNR values of the reconstruction (a) and prediction task (b) for different thresholds of online frequency-based forgetting mechanisms on data recorded on the humanoid robot ARMAR-6, compared to random-based forgetting.

age, similarity-based forgetting performs worse than random-based forgetting. Nonetheless, similarity-based mechanisms show a slightly lower variance than random-based forgetting. We observe that for similarity-based forgetting, the reconstruction capability of the model becomes worse with an increase in d_{min} (Figure 5.8). This corresponds to the start of a more drastic reduction in average PSNR values observable for threshold values of $d_{min} \geq 60$ (where the memory size gets smaller than 26% of the original size). Random-based forgetting, however, shows the aforementioned behavior of first worsening and then improving again. While smaller d_{min} results in interpretable predictions, large reductions in memory size arising from large similarity thresholds greatly worsen the prediction quality as shown in Figure 5.9. As before, on average, similarity-based forgetting performs worse than random-based forgetting for large reductions of memory size.

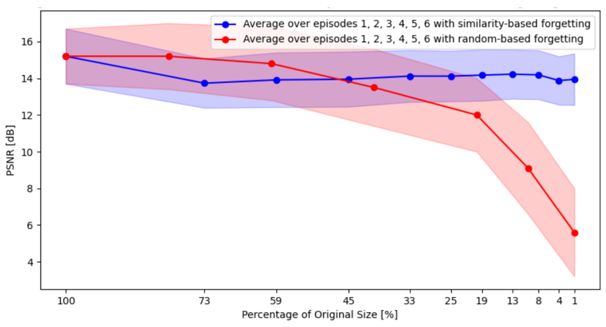
In summary, online frequency- and similarity-based forgetting mechanisms applied to the 20BN dataset can reduce memory size by up to 80% while maintaining minimal performance degradation in reconstruction and prediction. Similarly, and random-based forgetting mechanisms yielded comparable yet slightly better results within this amount of forgotten data. Due to the fact that episodes of the 20BN dataset are very distinct from each other but very similar within an episode, the proposed offline forgetting method could not be applied to this dataset.

ARMAR-6

On the real-world recordings, we evaluate the reconstruction and prediction capabilities for models trained on episodic memories that used online frequency-based forgetting mechanisms

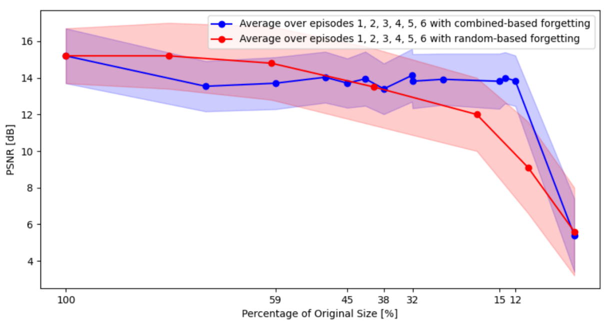


(a)

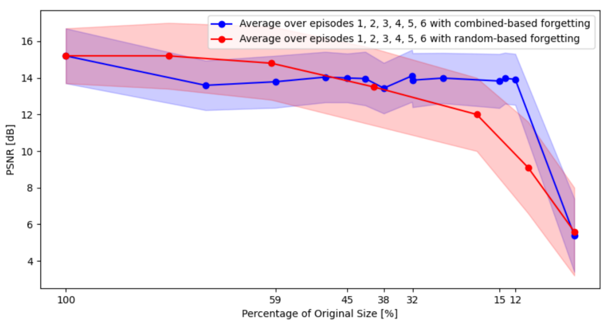


(b)

Figure 5.11.: Average PSNR values of the reconstruction (a) and prediction task (b) for different thresholds of online similarity-based forgetting mechanisms on data recorded on the humanoid robot ARMAR-6, compared to random-based forgetting.



(a)



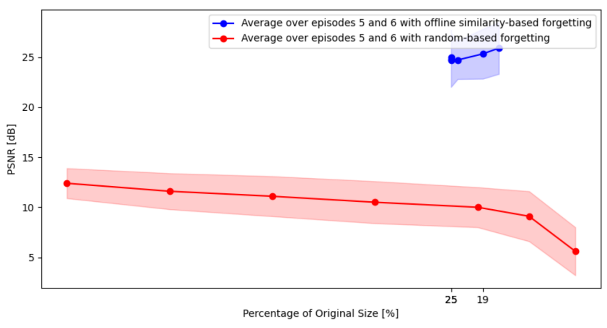
(b)

Figure 5.12.: Average PSNR values of the reconstruction (a) and prediction task (b) for a combination of frequency- and similarity-based forgetting mechanisms on data recorded on the humanoid robot ARMAR-6, compared to random-based forgetting.

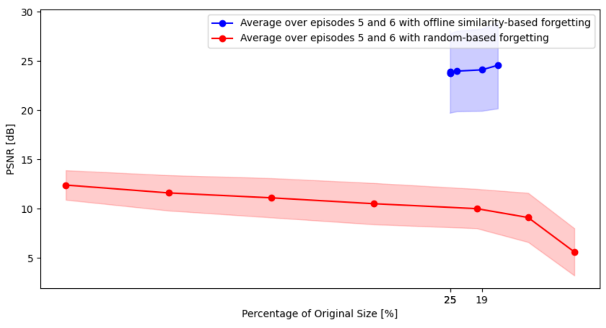
with the thresholds $\delta_{wait} \in \{100\,\text{ms}, 250\,\text{ms}, 500\,\text{ms}, 1000\,\text{ms}, 2000\,\text{ms}, 5000\,\text{ms}\}$. Similar to the evaluation done on the 20BN dataset, we use random-based forgetting as a reference model. As shown in Figure 5.10, reductions of up to 80% of the episodic memory size using small thresholds causes no significant decrease in average PSNR values. For all reductions, frequency-based forgetting performs better than random-based forgetting on the reconstruction and the prediction task.

To evaluate online similarity-based forgetting mechanisms we use the thresholds $d_{min} \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. Average PSNR values for the reconstruction and prediction task are shown in Figure 5.11. No big decrease in average PSNR values for increasing d_{min} was observed. Compared to random-based forgetting, similarity-based forgetting is superior, especially for larger size reductions using $d_{min} \geq 40$ which leads to an average memory size of $\leq 60\%$.

Utilizing the idea of Plewnia et al. (2024) to combine different forgetting mechanisms we evaluate a combination of online forgetting- and similarity-based forgetting. To test combinations of online forgetting algorithms, we evaluate the episodic memories resulting from all 12 combinations of the thresholds $\delta_{min} \in \{100\,\text{ms}, 250\,\text{ms}, 500\,\text{ms}, 1000\,\text{ms}\}$ and $d_{min} \in \{10, 20, 30\}$.



(a)



(b)

Figure 5.13.: Average PSNR values of the reconstruction (a) and prediction task (b) for latent similarity-based forgetting mechanisms on data recorded on the humanoid robot ARMAR-6, compared to random-based forgetting. The first applicable threshold already reduced the size to 25.2% and instantly increased the achieved PSNR values. After a reduction to 15.9% no further reduction was possible.

The results are close to the ones when leveraging online similarity-based forgetting, with the exception of a steep decline for small episodic memory sizes, that resulted from the combination of $\delta_{min} = 1000$ and $d_{min} = 30$ as shown in Figure 5.12. A characteristic shared with similarity-based forgetting mechanisms is the big difference in resulting episodic memory sizes for the same threshold combinations for different episodes. This make the combination of online forgetting mechanisms less useful than leveraging similarity-based solely.

Using the aforedescribed procedure to train two models on different and possibly filtered batches and then use a third batch for testing, we achieve size reductions of up to 75 to 84%. As the difference in calculated dissimilarity-values $D^{latent}(x_i, x_j)$ is rather small for $x_i, x_j \in \mathcal{E}_{enc}$, it is hard to find one threshold that achieves good results for different episodes. We use the threshold parameters $d^{latent}_{min} \in \{0.988, 0.99, 0.992, 0.994, 0.996, 0.998\}$. In our experiments, using $d^{latent}_{min} < 0.988$ led to no size reduction, meaning that the latent distance must exceed a critical limit before a compression effect occurs. Figure 5.13 shows that leveraging latent space representations in a similarity-based forgetting mechanism allows a increase in achievable PSNR values for the reconstruction and prediction task. This observation strengthens the assumption that the distances learned by the model allow more efficient forgetting than the previous online-capable but rule-based methods.

Given the individual results of the forgetting methods, we achieved the best results when using a combination of online frequency-based forgetting with $\delta_{wait} = 250\,\mathrm{ms}$, online similarity-based forgetting with $d_{min} = 10$, and offline similarity-based forgetting with $d^{latent}_{min} = 0.998$. The automated pre-filtering of the data can therefore have a positive effect on the AE's ability to learn. However, in this experiment the pre-filtering may only take place to a very limited extent and the identified threshold values highly depend on the used input. In another experiment, other thresholds might be necessary to observe a positive effect.

5.4. Evaluation of Long-term Memory on Sample Task

We also evaluated the memories ability to only remove redundant or useless data on a sample task. Similar as in the previous section, we utilize online and offline forgetting methods and identify the most promising combination.

5.4.1. Methodology

We conducted an experiment to assess a task that relies on the robot's episodic memory. Specifically, we recorded and analyzed the episodic memory of two humanoid robots: ARMAR-III (in simulation as shown in Figure 5.14) and ARMAR-6 (in real-world experiments). The experimental scenario involved typical tasks, such as object search and navigation. Various filtering methods and parameterization techniques were applied.

A fundamental challenge for both humans and robots is the ability to recall the last known location of an object. To investigate this, we introduced multiple objects into the robot's field of view, even though these objects were not directly manipulated during the assigned task. In the simulated experiments, we placed 40 objects in the environment, whereas the real-world experiments involved 23 objects. Our approach aimed to emulate human-like recall by searching the episodic memory for previously observed objects.

To evaluate our model, we performed a structured search through the episodic memory. The search was conducted by iterating backward in time through consolidated entity snapshots of the robot's visual perception until the target object was identified. The robot is doing a retrospective object search on memorized data. Once the object was found, we extracted the robot's state from the episodic memory, including its configuration and global position, to determine the exact moment and location of the recorded entity snapshot. Both visual and state information were processed using different forgetting mechanisms to optimize memory utilization.

The experiments focused on the most frequently accessed, most commonly used, and most storage-intensive data sources in ArmarX– proprioception data and visual data. While we primarily examined these modalities, our proposed memory and forgetting model is adaptable to incorporate additional sensory modalities as needed.

To systematically assess the efficacy of different memory management strategies, we quantified the following metrics:

1. Object Localization Accuracy (Task performance): We evaluated the number of objects the robot correctly recognized in the task scenario, comparing performance with and without forgetting mechanisms.
2. Episodic Memory Size Reduction: We analyzed the degree to which episodic memory size decreased under different forgetting mechanisms. To ensure comparability, we

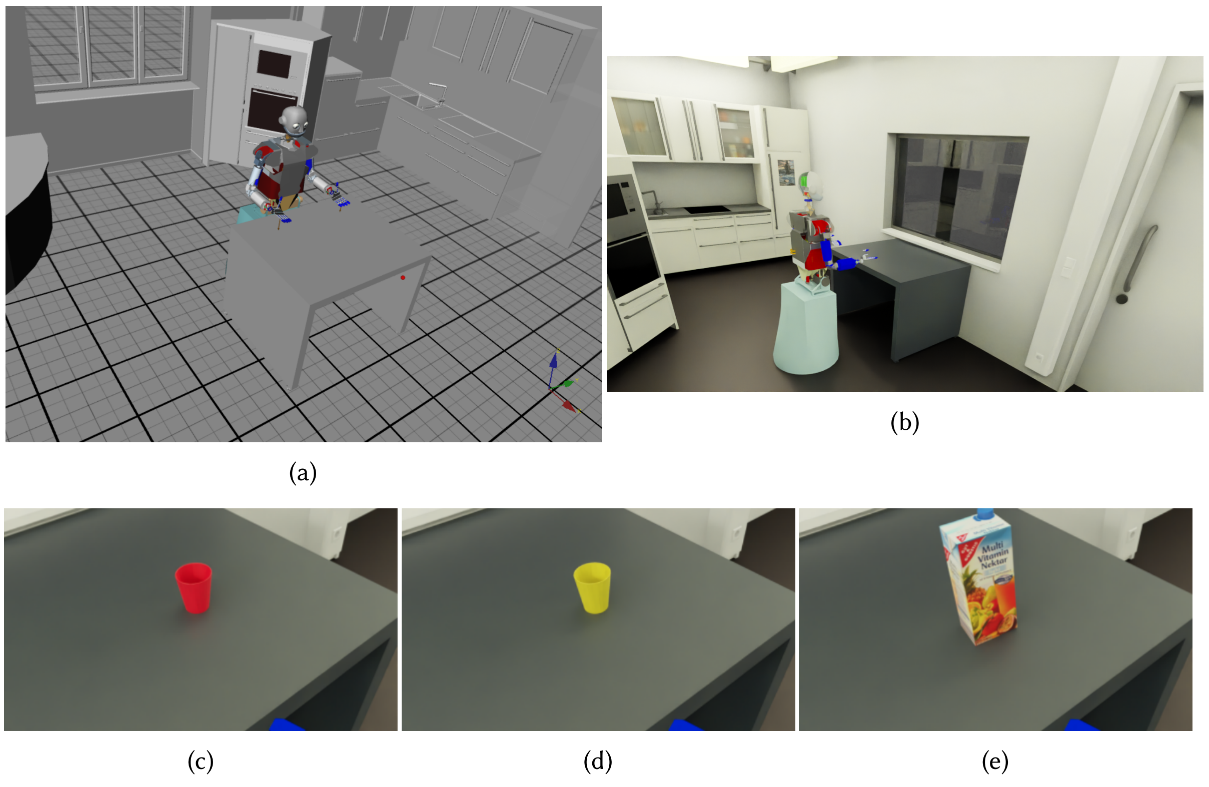


Figure 5.14.: Simulation environment (a) where data of the humanoid robot ARMAR-III was recorded. Since objects and robots in our simulation are not represented with much variety, the calculated visual impressions of the robot offer few features. This makes learning in deep EM more difficult. For this reason, the data can be recreated in the robot's memory in a more realistic environment (b). In this environment, new visual impressions could even be generated, for example by exchanging environments or objects with which the robot has interacted in an experience, as shown in (c), (d) and (e). ©Elsevier (Peller-Konrad et al., 2023), partially

assessed the number of stored snapshots across different mechanisms, including online and offline forgetting (applied before or after latent compression).

3. Computational Time Requirements: We measured the processing overhead associated with online forgetting (executed during task execution) and offline forgetting (performed when the robot is inactive).

Since we recorded data on the target system, the recordings exactly match the target scenarios in terms of embodiment, duration or continuity of the data.

Similar as before, all experiments were conducted on a computational setup comprising an AMD Ryzen 9 3900X 12-Core Processor, an NVIDIA GeForce RTX 3060 GPU, and $80\,\mathrm{GB}$ of RAM . A systematic comparison of various forgetting mechanisms and their combinations was performed using identical experimental episodes. The parameter settings that resulted in optimal performance were subsequently reported.

	Frequency	Similarity			Time-based Decay		Latent Similarity		Additional Time		Size	
	δ_{wait}	D	d_{min}	n	type	a_{min}	D_C	a_{min}	online	offline	best	average
(1)	200 ms	-	-	-	-	-	-	-	<0.01 s	0 s	15.28 %	15.38 %
(2)	-	MSE	100.0	2	-	-	-	-	8.2 s	0 s	12.31 %	12.43 %
(3)	-	Cosine	0.1	2	-	-	-	-	287 s	0 s	14.10 %	14.42 %
(4)	-	-	-	-	Exp.	1.679	-	-	0 s	382 s	46.60 %	50.77 %
(5)	-	-	-	-	-	-	Cosine	0.3	0 s	578 s	**4.62 %**	4.75 %
(6)	200 ms	MSE	50.0	-	-	-	-	-	8.3 s	0 s	7.75 %	7.82 %
(7)	200 ms	-	-	-	Exp.	1.679	-	-	<0.01 s	383 s	7.12 %	7.48 %
(8)	200 ms	-	-	-	-	-	Cosine	0.1	<0.01 s	579 s	6.31 %	6.37 %
(9)	200 ms	MSE	50.0	-	Exp.	1.679	-	-	8.3 s	402 s	3.76 %	3.98 %
(10)	200 ms	MSE	50.0	-	-	-	Cosine	0.1	8.3 s	384 s	3.14 %	3.21 %
(11)	200 ms	-	-	-	Exp.	1.679	Cosine	0.1	<0.01 s	962 s	4.50 %	4.63 %
(12)	-	MSE	100.0	2	Exp.	1.679	-	-	8.2 s	380 s	5.72 %	6.04 %
(13)	-	MSE	100.0	2	-	-	Cosine	0.1	8.1 s	580 s	3.25 %	3.31 %
(14)	-	MSE	100.0	2	Exp.	1.679	Cosine	0.1	8.0 s	800 s	3.07 %	3.14 %
(15)	-	-	-	-	Exp.	1.679	Cosine	0.1	0 s	961 s	6.95 %	7.17 %
(16)	200 ms	MSE	100.0	2	Exp.	1.679	Cosine	0.1	10.2 s	421 s	**3.03 %**	3.09 %

Table 5.1.: The table presents the optimal size reductions achieved while preserving 100% correct object localization on the simulation scenario, detailing results for individual mechanisms (rows 1–5) as well as for various combinations of mechanisms (rows 6–16). All mechanisms were evaluated on the same episode to ensure consistency in the experimental conditions. Additionally, we provide the average size of episodic memories, which were constructed from $n = 3$ episodes, each with an average duration of 8 minutes and 32 seconds. ©IEEE (Plewnia et al., 2024), modified

5.4.2. Results

Similar to the evaluation presented in Section 5.3.2, we analyze frequency-based and similarity-based online forgetting, as well as time-based and similarity-based offline forgetting. Table 5.1 summarizes the most significant size reductions achieved by each forgetting mechanism and their combinations that maintained 100% accuracy in object localization. The first three rows represent online consolidation from WM to LTM, while rows four and five illustrate offline consolidation using time-based decay and latent similarity measures. Subsequent rows depict combinations of these approaches.

In our experiments, frequency-based mechanisms (1) reduced episodic memory size by up to 84.72%. However, similarity-based mechanisms (2–3) slightly outperformed frequency-based methods, achieving reductions of up to 87.69%. This improvement arises from the repetitive nature of robot observations in similar environments, which leads to redundancy in stored data. Unlike our experiments using the 20BN dataset, where frequency-based mechanisms performed competitively, similarity-based approaches were more effective in reducing unnecessary information without impairing task performance, even when employing simple similarity measures such as MSE. Despite their advantages, similarity-based mechanisms require greater computational resources. While frequency-based approaches incurred a negligible computational overhead ($< 0.01\%$), the use of cosine similarity increased processing time by 57.06%. Among similarity-based methods, MSE (2) achieved the best balance between memory reduction and computational efficiency.

For offline forgetting, our experiments demonstrated that time-based decay (4) led to additional memory reductions of up to 55.40% without degrading task performance. The highest memory reduction achieved by a single mechanism while maintaining 100% object localization accuracy was 95.38%, obtained through latent similarity measures (5).

We also investigated whether combining different forgetting methods could enhance memory reduction while preserving task accuracy. Our findings indicate that integrating latent space similarity mechanisms with other approaches consistently improved memory reduction. Specifically, online similarity mechanisms should be combined with latent space similarity mechanisms (10,13,14) to ensure 100% object localization accuracy. Additionally, frequency-based methods could be paired with time-based decay (9,11,16) to optimize memory size reductions.

A combination of all methods resulted in a final memory size of 3.03%. However, this approach only yielded 100% task performance in simulation. In both, simulation and real robot experiments, the best balance between memory reduction and task performance was achieved by combining frequency-based, online similarity-based (MSE), and offline similarity-based (cosine similarity) mechanisms. This configuration reduced memory size to 3.31% in simulation and 7.99% in real world experiments of the original while maintaining 100% accuracy in retrospective object recognition tasks.

5.5. Discussion

The evaluations conducted provide a comprehensive analysis of our memory system in different dimensions, including working memory responsiveness, deep episodic memory reconstruction and prediction capabilities, and the effects of forgetting mechanisms. In the following, we discuss key findings, their implications, and potential limitations.

5.5.1. Working Memory

The results of the working memory evaluation reveal a clear trade-off between response time and system complexity. As expected, direct P2P communication exhibited the lowest latency, serving as a theoretical lower bound for data transmission. The PS mechanism, while introducing minimal overhead, lacks the ability to retrieve past information. The memory-based approach, though slower due to data storage and retrieval processes, provides essential cognitive functionalities such as historical data access and query-based retrieval.

Interestingly, while batch processing increased overall latency, it significantly improved efficiency per transaction, demonstrating that memory operations can be optimized by grouping data updates. Additionally, complex data types, particularly those involving images, benefited from optimized serialization formats, indicating that future implementations should incorporate specialized encoding techniques to further minimize processing time.

Despite its longer processing times compared to direct communication methods, our working memory implementation achieves query response rates exceeding $160\,\mathrm{Hz}$, making it suitable for many robotic and cognitive applications. In practical deployments, performance optimizations – such as co-locating producers, memory servers, and consumers on the same machine – can further improve response times.

5.5.2. Long-term Memory

In this chapter, we explored the design and evaluation of forgetting mechanisms in the context of deep episodic memory systems for cognitive robotics. Our research highlights the central importance of data completeness and quality when training compression and reconstruction models for long-term memory systems. While deep episodic memory architectures provide efficient storage solutions, their ability to reconstruct and predict missing or future information is highly sensitive to the volume and variability of their training data.

Through extensive experimentation on the 20BN dataset and the artificial episodic memory of the humanoid ARMAR-6 robot, we have shown that online forgetting mechanisms, particularly those based on frequency and similarity, can achieve substantial reductions in memory size, often up to $80 - 99\%$, without negatively impacting performance on reconstruction and prediction tasks. These mechanisms outperform random-based strategies and prove especially effective in scenarios where bandwidth and storage limitations make memory efficiency a critical concern.

In contrast, offline forgetting mechanisms such as those proposed by Plewnia et al. (2024) demonstrate limitations, especially when applied to episodic memories composed of short and highly diverse episodes. The all-or-nothing nature of their forgetting approach fails to generalize in our setup, highlighting a mismatch between theoretical models and practical implementations for robotic systems. However, our evaluations revealed that offline similarity-based mechanisms, while not improving prediction capability, significantly improve reconstruction performance – even surpassing models trained on full datasets – while reducing the memory requirements by up to 84%.

Interestingly, our findings indicate that traditional evaluation metrics like PSNR, while useful for measuring image quality, do not adequately capture the predictive accuracy of deep episodic memory systems. Especially in qualitative analyses, we observed discrepancies where models generated visually similar outputs that failed to reflect meaningful future changes. This underscores the need for developing task-specific or semantic-aware metrics for better evaluation of memory models in robotics.

Another key insight from this is that reducing episodic memory size does not necessarily compromise robotic performance – in fact, in some cases, it can even enhance it by reducing retrieval times and minimizing irrelevant information. This counters the intuitive assumption that more data always leads to better outcomes, and instead positions intelligent forgetting as a valuable cognitive function in robotic agents.

In conclusion, we demonstrate that thoughtfully designed forgetting mechanisms, particularly when used in an online fashion, are not only compatible with high-performing memory systems but are essential for their scalability and real-world applicability. As cognitive robots increasingly interact with complex, information-rich environments, the ability to dynamically manage memory – forgetting what is unnecessary while retaining what matters – will be fundamental to achieving adaptive, efficient, and intelligent behavior.

CHAPTER 6

Verbalization in Episodic Memory

Disclaimer

Parts of the content presented in this chapter were previously published in

- L. Bärmann, F. Peller-Konrad, S. Constantin, T. Asfour, and A. Waibel (2021). "Deep episodic memory for verbalization of robot experience". In: *IEEE Robotics and Automation Letters (RA-L)* 6.3, pp. 5808–5815

As motivated in Chapter 3, verbalizing episodic memory provides robots with the ability to explain their past experiences, improving transparency, trust, and usability in human-robot interaction. A robot that can answer questions like *"What did you do earlier?"* or *"Why did you fail this task?"* makes collaboration more intuitive and troubleshooting easier.

However, this task is challenging. Robots must encode rich, multi-modal experiences, resolve temporal references, and generate coherent, context-aware responses. Unlike static databases, episodic memory is dynamic and requires efficient storage, retrieval, and forgetting mechanisms. Moreover, translating raw sensor data into meaningful natural language remains an open problem.

In the following, we will explain how we implemented episodic memory verbalization using the deep episodic memory as a real-world example on robots of the ARMAR humanoid robot family.

6.1. Related Work

We first discuss previous work on verbalization systems in robotics and then proceed with the relation to the area of video question answering and task-oriented dialog.

6.1.1. Robot Experience Verbalization

The concept of *verbalization* for autonomous robots was first introduced by Rosenthal et al. (2016). The authors proposed a system enabling the "CoBot" robots to narrate navigation routes within a building using natural language. To ensure adaptability to different contexts, they defined the notion of a *verbalization space*, which is composed of three discrete dimensions: *abstraction*, *locality*, and *specificity*.

At the core of this approach is the variable verbalization algorithm, a rule-based method that takes as input a specified route, map information, and a designated point within the verbalization space. Based on these parameters, the algorithm generates a natural language description of the route that aligns with the given verbalization preferences. In subsequent work, Perera et al. (2016) expanded upon this framework by addressing the automatic determination of the appropriate point in verbalization space based on user queries. For instance, a request such as *"Please tell me exactly how you got here."* is interpreted as requiring a detailed narrative with a semantic abstraction level and a global locality. To achieve this, the authors compiled a corpus of $2,400$ user queries through online crowdsourcing. They then employed machine learning techniques, such as the Naive Bayes classifier, to categorize these queries, achieving a classification accuracy of approximately 70%, a level deemed sufficient given that the model could also refine verbalization parameters dynamically.

While these initial studies laid the foundation for verbalization in autonomous robots, they did not explicitly establish a connection between verbalization and episodic memory systems. Although navigation records could be considered a rudimentary form of episodic memory, they are inherently limited in scope and specificity. Addressing this gap, Zhu et al. (2017) developed an experience-based verbalization system for navigation and manipulation tasks using the ARMAR-III kitchen robot. In this system, episodic memory is implemented through a *Log File Generator*, which systematically records significant events during robot program execution across different levels of abstraction. These levels include *task* (e. g., emptying a dishwasher), *action* (e. g., grabbing a cup), and *primitive action* (e. g., moving a robotic platform).

To facilitate verbalization of past experiences, the rule-based variable verbalization algorithm originally introduced in Rosenthal et al. (2016) was adapted to this scenario. As a result, the system is capable of generating narrations such as *"I picked up the green cup on the sink with my right hand. It took four seconds."* This integration of verbalization with an episodic memory framework represents a significant advancement in enabling autonomous robots to articulate past experiences in a human-understandable manner, thus enhancing human-robot interaction in real-world environments.

6.1.2. Video Question Answering

Video Question Answering (VQA) is closely related to the verbalization of episodic memory, particularly when EM is constructed from video data. In both cases, a video sequence is provided along with a natural language question about an episode, and the model is required to generate the appropriate response. The complexity of VQA tasks varies significantly, ranging from selecting an answer from a predefined multiple-choice set (Lei et al., 2019), to generating a single-word response (Yu et al., 2019), and even to producing open-ended answer sentences using a decoder network (Zhao et al., 2020).

If the episodic memory consists solely of symbolic data, as e. g., in memory network (Weston et al., 2015), similarities emerge with knowledge-retrieval tasks, such as those encountered in restaurant dialogue systems (Wen et al., 2017). While a comprehensive review of Task-oriented Dialogue (TOD) systems is beyond the scope of this discussion, notable works such as Chen et al. (2019); Eric et al. (2017); Wu et al. (2019) utilize differentiable architectures to process natural language queries and retrieve relevant information from symbolic knowledge bases. In contrast to these approaches, EM-based Verbalization (EMV) differs in two key aspects: (1) TOD systems rely on a static, symbolic knowledge base, whereas EMV operates on a temporally structured stream of multimodal episodic recordings, and (2) the timing of input modalities differs, as explained in the context of VQA systems.

While hidden representations in VQA models may be considered implicit episode representations, the fundamental distinction between VQA and EMV lies in the timing of question processing. In VQA, both the video input and the corresponding query are presented to the model simultaneously, enabling mechanisms such as attention networks (Bahdanau et al., 2015) to focus on video segments relevant to the question. In contrast, EMV systems must handle queries that may arise at arbitrary points in the future (e. g., *"What did you do yesterday?"*). Consequently, episodic memory cannot be processed in direct relation to the question at the time of encoding. Instead, explicit episode representations must be constructed and stored in advance, while the original episodic input stream is discarded.

Additionally, the field of language command grounding in Human-Robot Interaction (HRI) (Kollar et al., 2014, 2017) is relevant to this discussion, as it seeks to link natural language expressions with multi-modal representations of a robot's environment. This connection further emphasizes the importance of structured episodic memory in enabling robots to interpret and respond to natural language queries effectively.

6.1.3. Large Language Models

Large Language Models (LLMs) have emerged as pivotal technologies in the realm of robotics, particularly for the verbalization of knowledge and enhancing HRI. These models, grounded in advanced natural language processing capabilities, allow robots to engage in more lifelike conversations, contextual adaptation, and consistent interaction with users, making them invaluable for diverse applications, including healthcare, education, and service industries (Kim et al., 2024; Ye et al., 2023).

The integration of LLMs into robotic systems enhances the interpretative and communicative functions of robots, enabling them to parse natural language commands and translate these into actionable tasks. This capability is crucial for developing systems that efficiently interact with humans in real-world environments. Recent studies highlight the capacity of LLMs to manage high-level planning and decision-making in robotics, drawing on their extensive world knowledge to interpret complex instructions and execute intelligent responses (Birr et al., 2024). The concept of utilizing LLMs for orchestrating robotic behavior aligns with the need for robots to process and engage with multi-modal sensory inputs – such as visual data and auditory signals – thus enriching the interaction experience (Zhao et al., 2023).

Incorporating dynamic memory frameworks into robots leveraging LLMs facilitates an understanding of context and continuity in dialogue, allowing robots to evolve their responses based on prior interactions (Soikao et al., 2024). This development aligns with novel architectures that utilize structured approaches, such as knowledge graphs, to support the contextualization of language, thereby improving the reliability of robotic responses in ongoing conversations (Zafar et al., 2024). Additionally, the iterative feedback loop between human inputs and robot actions ensures that LLMs not only adapt but also enhance their comprehension of tasks, leading to more effective execution of complex commands (Luan et al., 2024).

Research indicates that robots endowed with LLMs demonstrate heightened reasoning abilities, which are particularly beneficial in scenarios requiring task planning and collaborative interaction. For instance, LLMs can facilitate groundbreaking reasoning frameworks that allow robots to make informed decisions as they engage with their environments, thus seamlessly merging cognitive tasks with physical actions (Chalvatzaki et al., 2023). Furthermore, the successful coupling of LLMs with robotics brings forth new paradigms in emotion generation during interactions, enhancing the empathetic capabilities of social robots, which is especially pertinent in healthcare settings where emotional nuances are critical (Mishra et al., 2023; Pashangpour and Nejat, 2024).

In conclusion, knowledge verbalization in robotic systems through LLMs represents a significant advancement in HRI. By incorporating the robust capabilities of LLMs, robots can achieve sophisticated levels of interaction, planning, and execution that were previously unattainable, thus broadening the horizons of robotic applications across a variety of domains.

6.2. Dataset Collection

To develop and evaluate a model capable of verbalizing episodic memory, a large-scale dataset was collected in a simulated environment, similar as shown in Figure 5.14. This dataset consists of Robot Episode Recordings (RERs), capturing symbolic and sub-symbolic data from the robot's interactions, and natural language Question-Answer (QA) pairs, used to train the verbalization model.

6.2.1. Simulated Dataset and Verbalization Grammar

The dataset was generated using the ArmarX robot framework, which provides a high-fidelity simulation of a kitchen environment. This allowed for efficient and large-scale data collection using the robots working and long-term memory system with controlled variations in object placement, robot actions, and environmental factors. The humanoid robot performed everyday tasks such as moving objects, placing items on surfaces, and interacting with the environment. Next to timestamps used for temporal reasoning and retrieval, each recorded episode includes symbolic data like task goals, actions performed, action arguments, and execution status, along with sub-symbolic data such as camera images, object detections, robot configuration, human presence, and platform position.

To efficiently capture relevant data without overwhelming storage, only key frames were extracted whenever a new action was initiated, an execution status changed, or a predefined amount of seconds passed without significant changes (see Section 4.5.1). Each episode forms a temporally ordered collection of key frames, encapsulating a single planner goal.

To enhance the dataset diversity and model generalization, several augmentation techniques were applied. Episodes were assigned multiple random timestamps to vary time-dependent queries. Noise was introduced into object positions and robot states to simulate sensor imperfections. Multi-episode histories were constructed by combining episodes into longer sequences, helping the model learn long-term memory retrieval and sequential reasoning.

To train the verbalization model, a large number of question-answer pairs were generated using a hand-crafted grammar. This ensured structured and diverse question types, including basic action queries (e. g., *"What did you do?"*), temporal questions (e. g., *"What did you do before [time]?"*), failure explanations (e. g., *"Why didn't you move the object?"*), and duration-based questions (e. g., *"How long did it take to pick up the object?"*). Answers were constructed from recorded events in episodic memory, following predefined templates referencing symbolic and sub-symbolic data. By employing this approach, the dataset maintained consistency while covering a broad range of interactions.

6.2.2. Real-world Dataset and Questionnaire

To evaluate generalization beyond simulation, a dataset of real-world robot experiences was collected. The humanoid robot ARMAR-III was deployed in a real kitchen environment, performing similar tasks as in the simulation. Each episode adhered to the same episodic memory

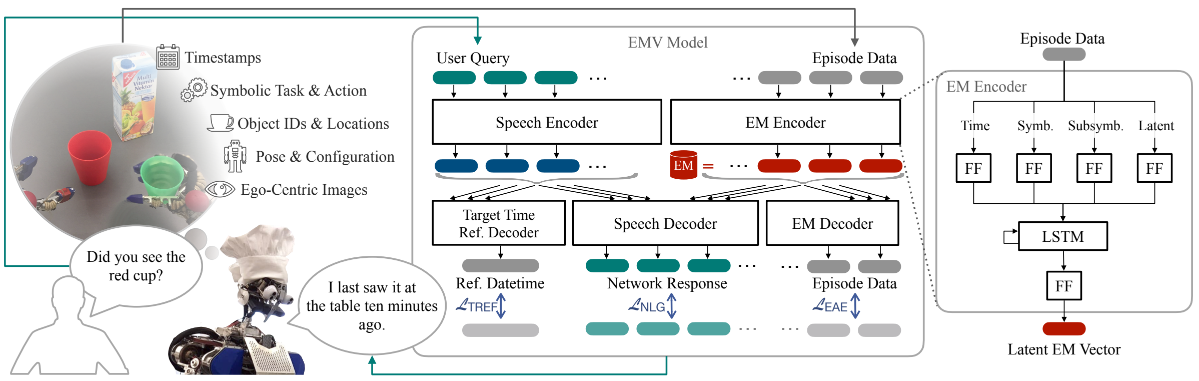


Figure 6.1.: The EMV model operates by continuously processing episode data from multi-modal sensory streams. These data streams are structured as a sequence of key frames, which are subsequently passed through the EM encoder network. This encoder constructs a latent representation of each episode, which is then stored in the episodic memory as a collection of encoded experiences. At a later point, when a user queries the robot about past events, the question—along with the timestamp of the inquiry—is processed by the speech encoder. The decoder then utilizes both the encoded query and the stored latent EM representations to generate a natural language response. This response is subsequently converted into speech using the robot's Text-to-Speech (TTS) system. The EMV model is trained end-to-end, incorporating two auxiliary decoders that contribute additional supervision signals to enhance overall learning and improve response generation accuracy. ©IEEE (Bärmann et al., 2021)

structure as the simulation, containing symbolic data (task goals, actions, and execution status), sub-symbolic data (camera images, detected objects, robot configuration), and timestamps.

Unlike simulation, real-world data introduced additional challenges such as sensor noise, occlusions, and unexpected human interactions. These aspects make real-world recordings more complex but also more representative of practical applications.

To introduce natural variation in language, human annotators generated questions and answers about the robot's experiences using a web-based interface. Annotators viewed video playback of recorded episodes, along with timestamps of key events and structured lists of symbolic and sub-symbolic data. They formulated realistic and natural questions, provided contextually appropriate answers, and used diverse phrasing to improve language generalization.

For example, a human-annotated QA pair might include:

- Question: "What were you doing just before you grabbed the cup?"
- Answer: "I was moving a plate to the counter."

The human-annotated dataset serves as a crucial evaluation set, assessing how well the model generalizes to naturally phrased user questions, which may differ from structured grammar-generated queries.

6.3. Network Architecture

The network architecture (see Figure 6.1) for EMV is designed to process natural language queries, retrieve relevant episodic information, and generate coherent responses. It consists of multiple components, including a Transformer-based speech encoder and decoder, supplementary decoders for enhanced processing, and a multi-stage training pipeline. This section outlines the architecture and its key functionalities.

6.3.1. Episodic Memory

Given the inherently multi-modal nature of robot perceptions, data is first converted into vector representations and embedded into a unified latent space before being processed. We employ the deep episodic memory architecture as introduced in Section 4.5.3. The aforementioned modalities are used as follows:

1. **Timestamps**: The temporal aspect of each key frame is encoded as a seven-dimensional vector, capturing the following elements: year, month, day, day-of-week, hour, minute, and second. This structured representation ensures that chronological relationships between episodes are preserved within the learned model.
2. **Symbolic information**: Symbolic data is encoded using a predefined dictionary constructed during preprocessing. Each entry in this vector corresponds to key semantic elements, including
   - Task goal as determined by the planner.
   - Task execution status (success, failure, or abort).
   - Executed action and its associated arguments.
   - Action execution status (initiated, success, or failure).
   - Object name (ID) and detection status.

   Unfilled vector entries are assigned a special padding token to maintain consistency in input dimensions.
3. **Sub-symbolic information**: The sub-symbolic data consists of numeric values that are concatenated to form a high-dimensional feature vector. These values include:
   - Detected object positions for up to two objects (expressed in the robot's root coordinate system).
   - Platform position (represented in the world coordinate system).
   - Human pose estimations derived from OpenPose (Cao et al., 2018) (expressed in the camera coordinate system).
   - Kinematic data of the robot's joints, including angles, velocities, and currents.

   To enhance training stability, each dimension of this vector is normalized based on the mean and standard deviation computed from the training dataset.

4. **Pretrained images**: A latent representation of the image associated with each key frame is obtained using a previously trained deep EM network as described in Rothfuss et al. (2018). This autoencoder model is trained as per the original methodology and fine-tuned using our dataset. Importantly, this network is not trained jointly with the EMV model; rather, the precomputed latent vectors serve as direct input features for our model.

   Each of the four feature components described above is independently passed through a fully connected layer. Symbolic data undergoes dimensional expansion, while sub-symbolic data is reduced in complexity. The resulting embeddings are then concatenated and used as input for the episode encoder. The hidden states of this encoder are subsequently processed through another fully connected layer, producing a latent vector for each time step (see the right part of Figure 6.1).

   As explained before, the episodic memory consists of the collection of all latent representations generated by the Auto-encoder from a sequence of input key frames. While the model functions as an end-to-end system during training and batched evaluation, deployment on a real robot introduces a fundamental distinction. In real-world scenarios, there may be an indefinite time gap between an experience and a subsequent user query. Thus, latent vectors must be persistently stored and retrieved on demand. Rather than directly constructing a large episodic memory from training data, our approach focuses on developing a learned model capable of dynamically generating latent EM representations when deployed in a robotic system.

6.3.2. Speech Encoder and Decoder

The natural language processing component of the EMV model is built upon the Transformer architecture (Vaswani et al., 2017). This sequence-to-sequence framework relies on self-attention mechanisms to iteratively refine its hidden state representations.

Specifically, we utilize the “small” variant of the *T5* model (Raffel et al., 2020). This model is pretrained on both an unsupervised fill-in-the-gap task and various supervised sequence-to-sequence language tasks, such as translation.

In the EMV model, the speech encoder responsible for processing user queries is directly adapted from the original T5 encoder. The resulting activations, referred to as the query encoding, are then passed to the decoder. Additionally, the decoder receives the sequence of latent vectors in EM, enabling it to attend to both the query and the episodic memory representations simultaneously. The T5 decoder remains unchanged from its original implementation in Raffel et al. (2020).

Finally, the decoder generates a probability distribution over the vocabulary, producing the response sequentially, word by word, to construct a natural language answer.

6.3.3. Supplementary Decoders and Loss Function

With the components described in the previous sections, the EMV model could be trained using a standard cross-entropy loss applied to the Natural Language Generation (NLG) output, as is common in sequence-to-sequence tasks (Raffel et al., 2020). However, to improve learning performance and provide additional analytical insights into the EMV task, we incorporate auxiliary decoders with additional loss terms.

First, we employ the reconstruction loss used for deep episodic memory. This serves as an auxiliary task to encourage the model to preserve rich episode details.

The episode decoder is implemented using four independent linear layers, each processing the latent EM vectors step-by-step:

1. Date-Time Reconstruction – One linear layer maps the latent vectors to a probability distribution over all possible date-time values, enabling the model to reconstruct temporal aspects of an episode.
2. Symbolic Data Reconstruction – Another linear layer maps to a probability distribution over symbolic episode tokens (e. g., task goal, action labels), treating this as a classification task trained with cross-entropy loss.
3. Sub-symbolic Data Reconstruction – A third linear layer directly regresses the numeric values of detected object positions, robot joint states, and human pose data using MSE loss.
4. Latent Representation Reconstruction – The final linear layer reconstructs the latent encoding of visual data using L1 loss.

We experiment with two training strategies: (1) jointly optimizing the model with both NLG and episode auto-encoder loss terms, and (2) pretraining the episode auto-encoder separately before integrating it into the full EMV model.

Second, we introduce an auxiliary time-reference decoder to assess the model's ability to correctly extract temporal references from user queries. This component attends to the hidden states of the query encoder using an attention mechanism, similar to the approach in Bahdanau et al. (2015). Its goal is to output the date-time referenced in the query string. Since our training dialogue dataset was generated synthetically (as detailed in Section 6.2.1), ground-truth timestamp references are available, enabling direct supervision for this task.

Finally, the complete loss function $\mathcal{L}$ used for training the EMV model is defined as:

$$\mathcal{L} = \alpha_1 \mathcal{L}_{NLG} + \alpha_2 \mathcal{L}_{EAE} + \alpha_3 \mathcal{L}_{TREF}$$

where:

- $\mathcal{L}_{NLG}$ represents the loss for natural language generation,
- $\mathcal{L}_{EAE}$ corresponds to the episode auto-encoder reconstruction loss, and
- $\mathcal{L}_{TREF}$ is the loss for time-reference extraction.

The episode auto-encoder loss $\mathcal{L}_{EAE}$ is further decomposed as follows:

$$\mathcal{L}_{EAE} = \beta_1 \mathcal{L}_{DT} + \beta_2 \mathcal{L}_{Sym} + \beta_3 \mathcal{L}_{Sub}$$

where:

- $\mathcal{L}_{DT}$ is the classification loss for reconstructing the date-time values,
- $\mathcal{L}_{Sym}$ is the classification loss for symbolic episode information, and
- $\mathcal{L}_{Sub}$ is the regression loss for subsymbolic key frame content.

The hyperparameters α_i and β_i control the relative weighting of each loss component, ensuring that $\sum_i \alpha_i = \sum_i \beta_i = 1$. This formulation allows for a balanced training process, where auxiliary tasks contribute to improving the model's ability to store and verbalize episodic memory while ensuring accurate response generation.

6.3.4. Training Procedure

The training process for the EMV model follows a structured three-phase approach to ensure effective representation learning and mitigate overfitting. This approach incorporates dimensionality reduction, episodic auto-encoding, and curriculum-based full model training.

The first phase focuses on learning a compact representation for the latent image vectors produced by the deep EM. To achieve this, we employ a non-linear neural network with a single hidden bottleneck layer, trained as an auto-encoder. This dimensionality reduction step ensures that the latent image representations are more manageable while retaining essential information. After training, all image vectors in the dataset are transformed into this reduced-dimensional space and subsequently treated as standard input and target data for further training stages.

In the second phase, the rest of the deep EM is trained, optimizing only the loss function $\mathcal{L}_{EAE}$. To enhance generalization and prevent overfitting, episode data is regenerated periodically throughout this phase. Specifically, after every few training epochs, new random date-time values are assigned to episodes, ensuring that the model is exposed to a diverse set of temporal inputs. This strategy broadens the range of date-time references the model encounters, improving its ability to generalize beyond the training data.

The final phase involves training the complete EMV model while leveraging the pretrained weights from the episode auto-encoder. To improve convergence and model stability, we employ a curriculum learning strategy, progressively increasing task complexity. Initially, the model is trained on single-episode histories, gradually scaling up to sequences containing up to five episodes. This staged approach allows the model to incrementally learn how to process and verbalize extended episodic memories.

Given the extensive number of parameters in the pretrained T5 language model, we selectively update only the speech encoder and the first layer of the decoder during this phase. Notably, we freeze the fully connected output layer of the T5 decoder, as it constitutes the largest portion of the model's parameters. This prevents catastrophic forgetting while ensuring that the model effectively integrates the episodic memory representations into its natural language generation process.

6.4. Results

6.4.1. Evaluation Metrics

Evaluating the performance of the EMV model presents several challenges, particularly due to the variability in how correct answers can be phrased in natural language and the inherent ambiguity of some questions. To systematically assess the model's output quality, we define the following response categories:

- Correct: The response is fully correct and contains no extraneous information.
- Too-much-information (correct): The response is correct but includes additional, relevant information beyond what was strictly required.
- Too-much-information (wrong): The response contains all necessary information but also introduces additional, incorrect details.
- Partially correct: Some aspects of the response match the expected answer, but certain details are either missing or incorrect.
- Partially correct, Only Action: The response lacks most required details, but correctly identifies the action performed by the robot. This category is specifically introduced due to the limited number of distinct actions in the simulated dataset.
- Wrong: The response conveys an incorrect answer while still maintaining the correct intent.
- Inappropriate: The response is entirely unrelated to the question (e. g., *"What did you do?"* $\rightarrow$ *"It is Tuesday."*).

Beyond categorical classification, we quantify the number of factual elements in each utterance. For instance, the sentence *"I moved the green cup to the sink."* consists of three distinct facts: move, green cup, and sink. By comparing the facts present in the model-generated response against those in the target utterance, we compute information precision and information recall as additional evaluation metrics.

For test datasets generated using the grammar-based dialog framework, the expected structure and content of both questions and responses are predefined. This allows for the development of a heuristic evaluation script that automatically categorizes model responses and computes precision and recall scores. The validity of this script was ensured through manual verification across all possible types of grammar-generated questions, leveraging the deterministic nature of the grammar.

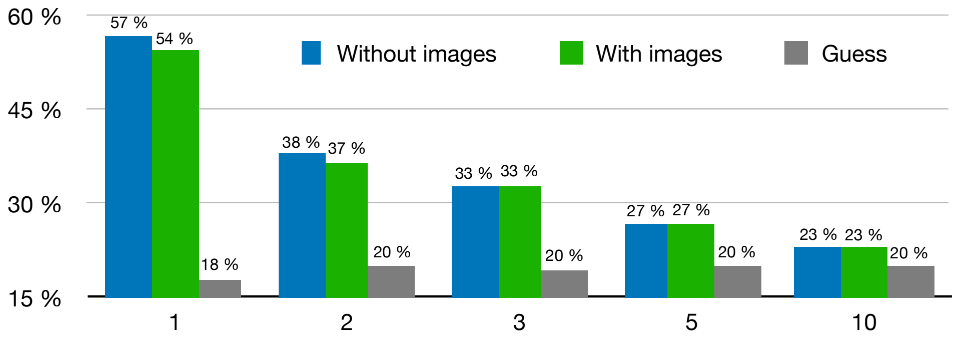


Figure 6.2.: Percentage of correct answers by history length. ©IEEE (Bärmann et al., 2021)

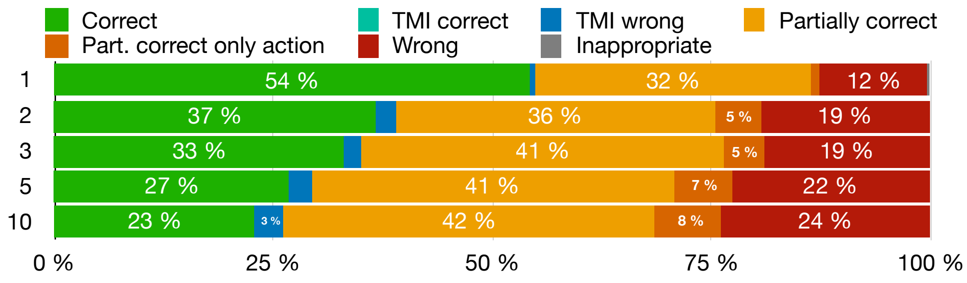


Figure 6.3.: Classification of answers by history length. ©IEEE (Bärmann et al., 2021)

In contrast, for human-annotated QA datasets, the underlying question structure is no longer strictly defined. As a result, automated heuristic evaluation is not applicable, and model performance on these test sets is assessed through human evaluation. This manual assessment follows the same response categories outlined above to ensure consistency across different evaluation methodologies.

6.4.2. Experimental Results

To assess the model's generalization capability across different history lengths, we evaluated its performance on simulated-robot-grammar-generated test sets containing varying number of episodes. This included histories of up to 10 episodes, which is twice the maximum length seen during training.

The results, presented in Figure 6.2, illustrate the percentage of correct answers as a function of history length. We compare the EMV models trained with and without images to a guessing baseline model, which is identical in architecture but trained without episode data. The baseline model relies solely on the input question and must infer the answer without episodic memory. This comparison is essential, as the highly structured nature of the generated dataset makes text-based guessing surprisingly effective, yielding approximately 20% correct answers.

Our findings indicate that EMV models incorporating episodic data significantly outperform the guessing baseline. However, performance declines as history length increases, highlighting the inherent difficulty of resolving date-time references and retrieving the correct entry from episodic memory. For a discussion of performance differences between models trained with and without images, refer to Section 6.4.3.

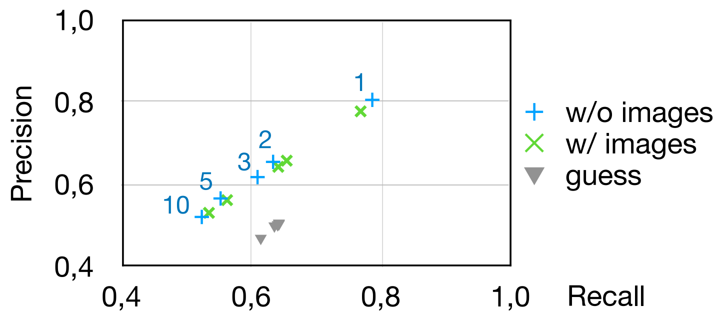


Figure 6.4.: Precision-Recall curve, referring to the information content of the utterances, as defined in Section 6.4.1. The different data points per model are the evaluation on histories of different lengths, as indicated by the numbers on the "w/o images" points. ©IEEE (Bärmann et al., 2021)

A more detailed classification of model responses is shown in Figure 6.3, which depicts results for the EMV model trained with images. While the decline in accuracy with increasing history length remains evident, the data further reveal that approximately 75% of the responses are at least partially correct, with only a minimal fraction classified as inappropriate.

For comparison, the guessing baseline model achieves:

- 20% correct answers,
- 4% too-much-information (wrong) answers,
- 52% partially correct answers, and
- 20% incorrect answers,
- Less than 0.1% inappropriate answers, independent of history length.

This suggests that while the structured dataset allows for relatively strong performance even without episodic memory, the EMV model's ability to incorporate past episode data provides a significant advantage.

Figure 6.4 presents the precision-recall curve, evaluating the information content of model responses. Several key observations emerge:

1. All EMV models exhibit higher precision than the guessing baseline.
2. Guessing achieves an artificially high recall, which can be attributed to the training strategy favoring longer outputs that include multiple facts.

An analysis of the guessing model's output reveals that 59% of all responses are identical, typically repeating a generic sentence such as: *"I tried to but failed to move the multivitamin juice and the red cup to the countertop."* This repetition leads to high recall but low precision. In contrast, the episode-informed EMV models avoid such overfitting, providing more contextually relevant answers.

To overcome the limitations of grammar-generated QA data, we additionally evaluated performance on simulated episodes annotated with human questions. Figure 6.5 presents results from this evaluation.

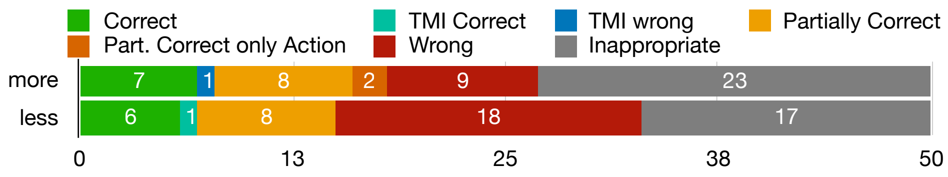


Figure 6.5.: Human evaluation results on the test set with human-annotated questions (absolute numbers). The two lines differ with respect to the number of trained parameters in the T5 speech network, with the lower one having less trained parameters. ©IEEE (Bärmann et al., 2021)

The findings indicate that the model struggles to generalize to human-annotated questions. This suggests that, despite leveraging a pretrained Transformer-based language model, the natural language capabilities do not fully transfer, leading to a failure in handling previously unseen utterances.

We also conducted evaluations on real-world robot recordings, comparing performance on:

- Real-robot-grammar-generated questions, and
- Real-robot-human-annotated questions.

For the real-robot-grammar-generated dataset, the model achieved:

- 39.3% correct answers,
- 29.9% partially correct answers,
- 28.7% incorrect answers, and
- 0.4% inappropriate answers, for episode histories of length one.

However, in the real-robot-human-annotated dataset, despite a small sample size, the model's generalization issues persisted, producing:

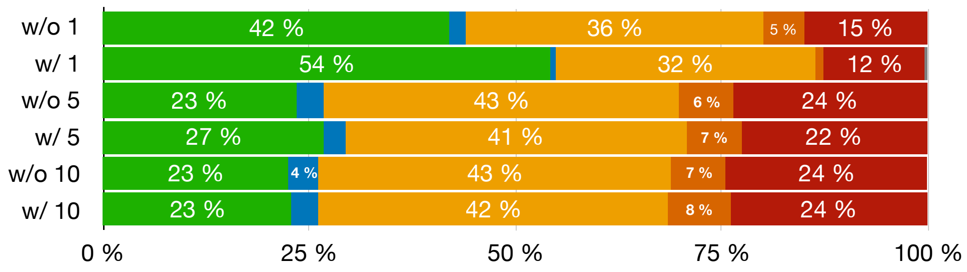
- Two correct answers,
- one too-much-information (correct) answer,
- one incorrect answer, and
- nine inappropriate answers.

These results indicate that while the model effectively handles structured, grammar-generated queries, it fails to generalize to more natural human-generated questions. However, using real-world rather than simulated episodes does not negatively impact performance.

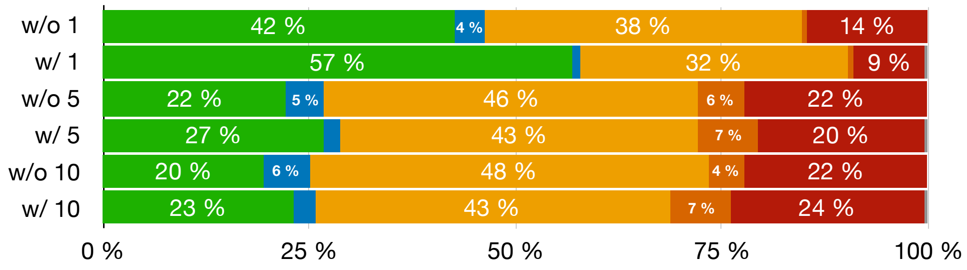
In summary, the EMV model demonstrates strong performance on structured, grammar-generated datasets but struggles with natural human queries, emphasizing the need for further improvements in robustness and generalization.

6.4.3. Ablation Study

To gain deeper insights into the model's behavior, we conducted additional ablation experiments. Figure 6.2 already presents a comparison between a model trained with episode image data and one trained without it. The results indicate that incorporating visual information does



a) Ablation study: additional loss functions (all modalities)



b) Ablation study: curriculum learning (without images)

Figure 6.6.: Results of different ablation experiments. Each experiment compares the answer classification results on the *simulated-robot-grammar-generated* test set between a model without (w/o) and with (w/) the corresponding option. The numbers for each row (1, 5, 10) indicate the history length used for evaluation. ©IEEE (Bärmann et al., 2021)

not improve performance and, for certain history lengths, even degrades it. This phenomenon can be attributed to the nature of the generated dialog training data, which does not effectively utilize image-based information. As a result, the model likely learns to ignore this modality.

Since the model struggles to leverage multi-modal data due to limitations in the training set, we further analyzed other aspects of the EMV model training strategy. The results of different ablation studies are summarized in Figure 6.6.

A key question concerns the utility of the complex loss function used in EMV model training. To investigate this, we trained two versions of the EMV model (both receiving all episodic modalities):

1. One using the full loss function $\mathcal{L}$.
2. One using only the NLG loss $\mathcal{L}_{NLG}$, effectively setting $\alpha_2 = \alpha_3 = 0$.

Experimental results show that including additional loss terms improves performance for history lengths up to five episodes (the maximum length seen during training). However, this benefit diminishes for histories of length ten, where both models converge in performance, approaching the guessing baseline.

Another crucial design element of EMV model training is the use of curriculum learning to gradually increase the number of episodes in training histories. Figure 6.6 compares two models:

- One trained directly on histories of up to five episodes.
- One trained using curriculum learning, progressively increasing history length.

The results confirm that curriculum learning enhances model performance, both on histories within the training distribution and in generalization to longer histories. This demonstrates that a structured, gradual training approach contributes to improved episodic memory retrieval.

To evaluate the impact of reducing the number of trainable parameters in the T5 speech encoder, we conducted additional experiments. While this modification did not improve performance on the simulated-robot-grammar-generated test set, it yielded a slight improvement on human-annotated utterances in the simulated-robot-human-annotated dataset. However, as shown in Figure 6.5, this improvement was marginal, mainly shifting some "inappropriate" responses into the "wrong" category rather than producing more accurate answers.

Our ablation experiments highlight several key findings:

- Multi-modal data (images) does not improve performance, likely due to training data limitations.
- The additional loss functions contribute positively to performance on training-length histories but provide no advantage for longer histories.
- Curriculum learning enhances both performance and generalization, reinforcing its effectiveness as a training strategy.
- Reducing trained parameters in T5 has minimal impact, with only a slight improvement in human-annotated test sets.

Overall, these results suggest that while certain design choices, such as curriculum learning and loss function design, positively impact model performance, there remain challenges in leveraging multi-modal inputs and generalizing to human-annotated queries.

6.5. Discussion

The proposed approach enables natural language responses to queries about past events and activities based on stored episodic data. We first constructed a comprehensive dataset comprising episodic recordings from both simulated and real-world robot executions. To enrich this data, we generated a diverse set of question–answer pairs using a combination of grammar-based generation and human annotations. Secondly, we introduced the EMV neural architecture, which integrates multiple learning modules to enable verbalization of robot experiences.

While our results demonstrate the viability of episodic memory-based verbalization, this work represents only an initial step. Recent advances in LLMs have shown their ability to generate context-aware, coherent narratives grounded in vast language corpora. Building on this, we developed `H-EMV`, a system that leverages LLMs to enhance the expressiveness and contextualization of verbalized robot experiences across long-term deployments (Bärmann et al., 2024a). `H-EMV` employs a hybrid memory structure that fuses high-level human-interpretable summaries with low-level, fine-grained episodic data. This structure enables LLMs to generate natural language reports that are not only faithful to the robot's experiences but also appropriately tailored to the user's informational needs. The system utilizes prompt engineering and memory abstraction techniques to retrieve relevant events and contextual cues, allowing it to produce narratives that reflect temporal dependencies, causal relationships, and semantic coherence. As demonstrated in multiple use-case evaluations, H-EMV supports robust verbalizations in dynamic environments, making it a promising approach for scalable, user-aligned robot communication.

A critical next step involves adapting verbalization to individual users. As discussed in Chapter 3, the robot must consider personal preferences and user-specific knowledge when verbalizing memory content. In another work, we extended the robot's EM with explicit user profiles and experience-based personalization (Weberruß et al., 2025). Two complementary methods were explored: (i) structured key-value profiles storing attributes such as preferences and restrictions, and (ii) incremental interaction-based learning, where user-specific feedback dynamically updates memory content. Experiments showed that while explicit profiles improved preference recall (sp = 92%), interaction-based learning achieved higher overall task success (st > 80%) in diverse scenarios. These findings suggest a hybrid approach is required to balance contextual adaptability with preference fidelity.

Personalization inevitably raises important challenges around data security and user privacy. When a robot retains and verbalizes interaction histories or user-specific knowledge, it effectively handles sensitive information. As emphasized in Bayreuther et al. (2022), such capabilities necessitate strict privacy-preserving frameworks. We argue that any robotic system capable of storing and externalizing information – particularly through natural language – must include mechanisms for access control, data minimization, and explicit consent. These mechanisms must also distinguish between public and private information, particularly when multiple users interact with the same system, possibly in shared environments.

While our current system does not yet implement these safeguards, we envision integrating them as essential components in future iterations. This includes user-specific data governance policies, dynamic access rules, and interaction-aware verbalization filters to ensure ethical, context-sensitive use of memory-based verbal communication.

CHAPTER 7

Experience-based Planning

Disclaimer

Parts of the content presented in this chapter were previously published in

- F. Peller, M. Wächter, M. Grotz, P. Kaiser, and T. Asfour (2018). "Temporal concurrent planning with stressed actions". In: *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pp. 901–908

We motivated that remembering previous experiences is crucial for decision-making and adapting to dynamic changes (Section 3.3). The way in which a problem has been addressed in the past significantly influences how similar or identical problems will be approached in the future. This is particularly evident in cases where a prior execution has failed or was not fulfilled in accordance with the required constraints. Often, the action models employed in robotic systems are insufficiently precise or even change over time due to wear, failures or decreasing power levels, leading to discrepancies between predicted and actual outcomes in real-world scenarios. If a robot can identify the reasons behind a previously failed action, it can adapt its strategy to either avoid or modify the action in similar future situations, thereby reducing the likelihood of failure.

For instance, humans can perform well under temporal stress by estimating action durations and adjusting execution speed, though this increases the risk of failure. This tradeoff between acceleration coming from stress and risk motivates experience-based planning, where past experiences guide decision-making under time constraints. The exact relationship between stress, acceleration and risk however is unknown, which can lead to errors in time-critical tasks. Only with increasing experience and thus increasing training can the relationship between these variables be better approximated so that, when new constraints – such as time or cost limitations – are introduced in future tasks, leveraging past execution data can enhance the accuracy of action models, enabling more reliable planning and decision-making.

To effectively utilize past knowledge for improving task execution, the following key questions must be addressed:

- Has a similar problem been encountered in the past? What does similar mean?
- If so, was the execution successful? Did it meet the expectations, or did unforeseen issues arise? What were the associated costs, in terms of time or resources?

By systematically analyzing past execution data stored in Episodic Memory, robots can refine their decision-making processes, optimize resource allocation, and enhance overall task efficiency. In the following, we explain how we take the above questions into account when planning the robot's actions based on experience using EM. We focus on updating cost functions and assume that the initial models were inadequately or incorrectly defined.

7.1. Related Work

7.1.1. Symbolic Planning and Domain Definition

Symbolic planning in robotics provides an essential framework for decomposing arbitrary complex tasks into manageable sub-actions, usually by using high-level representations that abstract away continuous details. In embodied systems, this approach facilitates a clear separation between the logic of task execution and the underlying geometric and kinematic constraints, thereby allowing the system to reason about actions in a domain-independent manner symbolically. Coming from classical non-embodied symbolic planning, domain definition languages such as Planning Domain Definition Language (PDDL) have emerged as standard tools for specifying the instantaneous actions, predicates, initial states, goal states and planning constraints that drive symbolic planning (Ghallab et al., 1998). The resulting domain and problem description is used by automated planners that generate a sequence of actions an agent must take to achieve the specified goal from a given initial state, considering defined constraints and available actions. By employing PDDL, researchers and engineers can describe robot capabilities and environmental states in a unified formal language that both algorithms and planning engines can readily interpret. Automated planners are widely used in robotics, AI, and logistics to enable intelligent decision-making in complex, dynamic environments.

Search algorithms constitute the backbone of many symbolic planners and are critical for efficiently navigating the combinatorial space of possible action sequences. Graph-search techniques, including variants of A-star (A*) (Hart et al., 1968) and its parallelized extensions (Aine et al., 2015), have been widely exploited to find optimal or near-optimal plans. These algorithms benefit from heuristic functions that estimate the cost-to-go and thereby accelerate convergence to a solution by pruning away bad decisions and focusing on promising ones. Given the right heuristic function, functions like A* are guaranteed to find the optimal path.

The incorporation of a domain-specific language like PDDL in symbolic planning not only enhances the modularity and maintainability of planning systems but also enables compatibility with a variety of planning engines and search algorithms.

7.1.2. Time-aware Planning

Time-aware or temporal planning is a generalization of classical automated planning paradigms. It is crucial, particularly in robotics where actions not only have instantaneous effects and preconditions, but also explicit durations and specific temporal constraints. In robot applications, it is essential to know when such a durative action should start, how long it will take and when its effects will take place. Motivated by the interval algebra for temporal reasoning by Allen (1983), extensions to planning languages, such as PDDL 2.1, have been developed to incorporate durative actions with conditions and effects that can be annotated as “at start”, “over all”, or “at end” (Fox and Long, 2003). This temporal augmentation enables planners to model and solve problems where both the sequence of possibly concurrent actions and their precise timings relative to each other matter, ensuring feasibility in real-world execution. Similar to classical planners, temporal planners make use of heuristic functions to estimate the remaining duration of the plan until the goal is reached. Given the right heuristic function, temporal planners are guaranteed to find the optimal solution, although temporal planning is more complex than classical planning because it not only determines which actions to execute but also when to execute them while considering time constraints, action durations, and concurrency. The duration of the final plan is called *makespan*.

7.1.3. Success-Failure Prediction

Considering learning from experience and experience-based planning, predicting success/-failure is an increasingly important part of robot planning. Ideally, predictive models that incorporate historical data are used to guide the planning process. By leveraging machine learning and statistical methods, these approaches can forecast the outcome of actions under varying conditions. The integration of success-failure prediction into planning is further enhanced when linked with sensorimotor experience, as outlined by Hegemann et al. (2022); Wächter et al. (2018). Wächter et al. (2018) emphasizes the importance of mapping symbolic plan representations to perceptual inputs during plan execution. This mapping allows for real-time monitoring and predictive evaluation of each action’s outcome. By incorporating such bottom-up feedback, planning engines can update their expectations dynamically, enabling a more resilient response to unforeseen circumstances, and effectively incorporating failure prediction into a continuous learning cycle. Similarly, Hegemann et al. (2022) propose an approach for failure detection during the execution of grasping and mobile manipulation tasks by a humanoid robot, combining multi-modal sensory information to learn task models from multiple successful task executions. Symbolic action predicates were learned based on the multi-modal sensory information to allow high-level state estimation based on action-specific decision trees.

Within the context of experience-based planning (Mokhtari et al., 2016), success-failure prediction mechanisms serve as a critical evaluation step, assessing whether the conditions that led to previous failures have been mitigated or persist in new scenarios. Thus, these predictive models not only help in avoiding errors in future task executions but also contribute to a more refined knowledge base that can be leveraged for iterative plan improvement.

Not only focused on success-failure prediction models, we believe that experience-based planning is an approach where past experiences of a system are used to inform and refine future decision-making processes. This also includes action model refinements regarding e. g., preconditions, effects or cost functions. At the heart of this approach lies the integration of episodic memory, which serves as a repository for rich contextual information and allows cognitive architectures to learn from, adapt to, and predict future events based on previous encounters (Vernon et al., 2007).

7.2. Planning With Stressed Actions

We motivated that robotic systems operating in dynamic environments often face scenarios in which actions are not only time-constrained but also subject to stress due to deadline pressures or environmental uncertainties. Unlike humans, who can adjust their action speed to meet deadlines – albeit with an increased risk of failure – robots typically lack this flexibility (Cushing et al., 2007). We introduced the concept of *stressed actions* (Peller et al., 2018), which are actions whose execution may be adversely affected by external temporal pressures or dynamic conditions. Stressed actions extend the traditional formulation by explicitly accounting for variations in execution times and the potential degradation of performance as deadlines approach. This approach enables robotic systems to incorporate adaptive strategies that either mitigate the stress impacts or replan actions dynamically to conform with the temporal limitations imposed by the operating environment. In time-critical robotic tasks, such as emergency response or precision manufacturing, considering stressed actions ensures that the planning system accounts for both nominal durations and variable execution conditions, thus enhancing robustness and reliability in real-world applications.

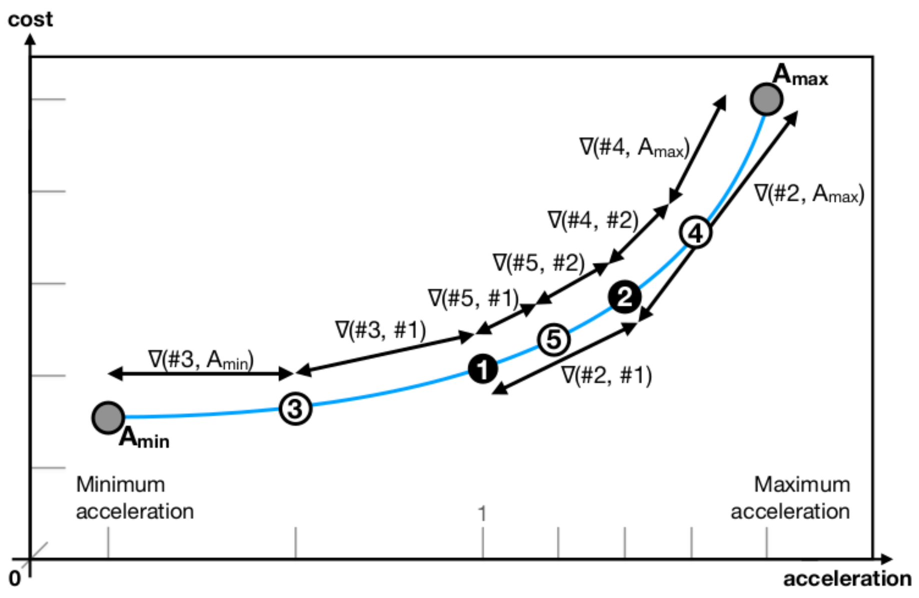


Figure 7.1.: Since the cost functions are continuous, there is an infinite number of possible action accelerations. Hence, TSFD discretizes stressable actions following a binary approach concentrating on the steepest part of the cost graph. Black filled circles depict already accepted accelerations and white filled circles show open candidates. Gray filled circles picture the minimum and maximum acceleration. The arrows above each candidate show the gradient to the next accepted acceleration or boundary. The blue graph depicts the cost function from its minimum acceleration up to its maximum acceleration. ©IEEE (Peller et al., 2018)

7.2.1. Planning with Stressed Actions

For being able to plan with stressed actions, we present Temporal Stressing Fast Downward (TSFD), an extension of the Temporal Fast Downward (TFD) (Eyerich et al., 2012) planner designed to handle temporally constrained problems. TSFD allows the modification of action durations within predefined acceleration bounds. Each action consists of:

- A default execution time (unstressed).
- A minimum and maximum acceleration factor (a_{min}, a_{max}).
- A manually defined cost function $f(a)$ that expresses the trade-off between speed and risk.

Instead of assuming a fixed duration for each step, TSFD allows actions to be executed faster or slower, depending on the urgency of the task. However, this flexibility comes at a cost, modeled through a manually defined cost function that quantifies the trade-off between speed and “risk”. This “risk” can represent failure probability, energy consumption, or execution uncertainty, depending on the domain. For example, the approach movement of the hand can be accelerated when grasping, but this is associated with a higher risk of collision.

To make these adaptations computationally feasible, TSFD discretizes the acceleration space. An example is shown in Figure 7.1. Instead of treating acceleration as a continuous variable, the planner selects from a finite set of acceleration levels that provide the highest information

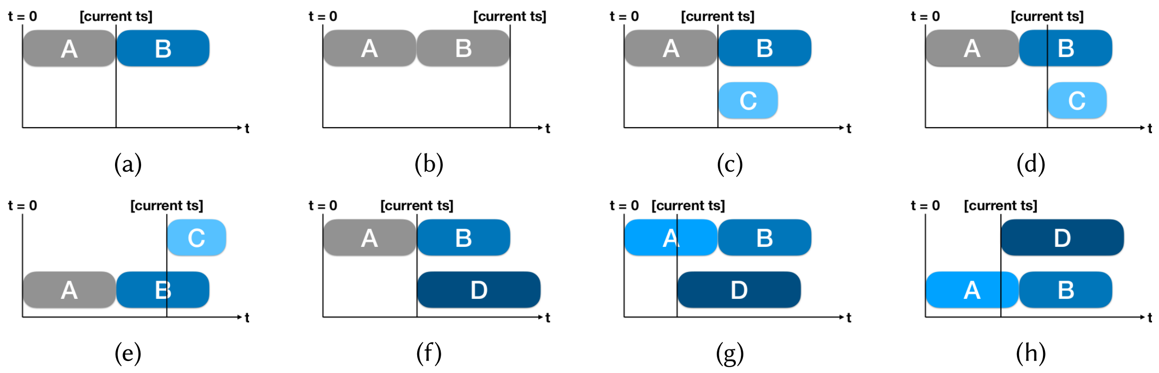


Figure 7.2.: All derived new states after scheduling a new action C or a new action D to a state s with an already scheduled action B and executed action A. Here $duration(C) < \Delta_{DE}$ and $duration(D) > \Delta_{DE}$ where Δ_{DE} denotes the distance of the current timestamp to the next decision epoch (i. e., closest end or begin event of a scheduled action in the future of s). current t_s denotes the timestamp of the generated state. Actions with gray colors depict already executed actions. Figure 7.2(a) presents the assumed state before any new action is scheduled. Figure 7.2(b) represents the the state after executing the scheduled action B. The state after scheduling C so that it begins with B is shown in Figure 7.2(c). That C ends shortly before or after B is shown in Figure 7.2(d) and 7.2(e). Figure 7.2(f) presents the state after scheduling D begin with B. As before, that D ends shortly before or after B are shown in Figure 7.2(g) and 7.2(h). ©IEEE (Peller et al., 2018)

gain about the cost-duration trade-off. This enables the system to balance speed and risk without excessive computational overhead.

TSFD modifies the standard search method used in TFD, improving its ability to handle temporally expressive problems. In conventional search, new actions can only be scheduled at the start of the problem or when a previously scheduled action completes. This rigid structure limits the ability to find feasible solutions in time-sensitive scenarios.

To overcome these limitations, TSFD introduces a more flexible scheduling approach. If there is an already scheduled action B in the current state s, actions that have a shorter duration than B can be scheduled in the future, allowing the planner to consider alternative execution timings, or (if the action has a longer duration than B) scheduled in the past, enabling actions to finish at the same time as others to maximize concurrency. Also, no further scheduling and increasing the current timestamp of s to the next decision epoch, as described in Figure 7.2, is possible. If no action is scheduled in s, we can only schedule a new action. The planning procedure is complete and optimal as described in more detail in Peller et al. (2018). This flexibility increases the likelihood of finding feasible solutions in time-restricted environments.

Additionally, TSFD integrates two heuristic functions to improve search efficiency. The first, Context-enhanced Additive Heuristic (HCEA), estimates the remaining makespan and guides the search towards shorter solutions. However, HCEA struggles with required concurrency. To address this, TSFD introduces a second heuristic based on the Relaxed Temporal Planning Graph, which retains information about action durations and improves the handling of temporally expressive states. By integrating these improvements, TSFD produces higher-quality plans that better account for time constraints and concurrent execution.

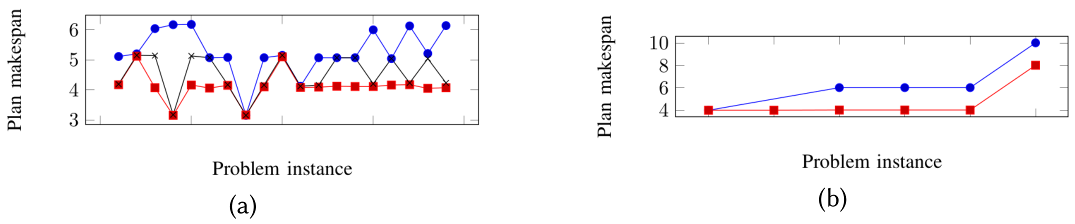


Figure 7.3.: Results of TSFD (red squares) in comparison to TFD (blue circles) and the best solution so far (black crosses) on the parking challenge (Figure 7.3(a)) and on temporally expressive domains (Figure 7.3(b)). A missing node indicates that the planner was not able to find a solution. ©IEEE (Peller et al., 2018)

7.2.2. Results

TSFD was evaluated on standard planning benchmarks, service robotics tasks, and real-world experiments with the humanoid robot ARMAR-III.

In benchmark tests, TSFD consistently produced shorter plans than TFD, especially in scenarios with strict deadlines as shown in Figure 7.3. It successfully solved problems that TFD failed to address, particularly those requiring concurrent execution. In domains where previous planners struggled to find optimal schedules, TSFD demonstrated a higher success rate by leveraging stressed actions and improved scheduling flexibility.

The planner was further tested in service robotics tasks, including object manipulation and surface wiping. These experiments confirmed that TSFD outperformed traditional planning approaches, particularly in time-sensitive situations.

To validate its real-world applicability, TSFD was deployed on ARMAR-III in a cooking scenario. The robot was tasked with preparing scrambled eggs within ten minutes, a constraint that standard planning methods could not satisfy. Without stressed actions, the generated plan exceeded the deadline. By accelerating key actions, TSFD enabled the robot to complete the task on time, demonstrating that stressed actions can enhance robotic performance in practical settings.

7.3. Learning Cost Functions from Experience

While TSFD successfully introduces stressed actions, it relies on manually defined cost functions that do not adapt to changing environments or execution history. In real-world applications, the relationship between acceleration, risk, and efficiency is not fixed – it depends on task complexity, environmental factors, and past execution data.

The next step is to replace static cost functions with experience-based learning. Instead of predefining cost models, the system should learn from execution history, refining its cost estimates from similar actions of the past dynamically.

7.3.1. Similarity Measurement

To effectively update the cost function of an action based on past executions, it is crucial to determine which historical execution data is relevant. This requires defining a similarity measurement that identifies past executions comparable to the current execution. The selection of relevant past executions for an action X forms the dataset T_X, which is used for cost estimation and anomaly detection.

The most straightforward similarity criterion is complete equality. Under this approach, an execution is considered relevant only if all aspects of the action match exactly:

- The action name must be identical (and thus its parameters, preconditions and effects as we assume unique action names)
- All parameters used in the action must have the same values.
- The world state at the time of execution must be identical.

While this approach ensures that only perfectly matching executions are used for updating the cost function, it is impractically restrictive. Even minor variations in the world state, such as other objects being present in a different but irrelevant location, would exclude past executions from consideration.

A more practical approach involves relaxing the equality constraint by considering only relevant features of the environment and execution. This means ignoring aspects that do not significantly impact execution time or success probability. For example:

- In a grasping task, only the objects close to the target object's location are relevant, while other objects in the scene can be ignored.
- In a navigation task, the presence of obstacles in the direct path is crucial, whereas objects outside the robot's trajectory are irrelevant.

By defining task-specific relevance criteria, this method increases the number of comparable past executions while maintaining meaningful similarity.

Also, machine learning techniques can be used to group similar executions through clustering, nearest neighbor search or neural embeddings, as described in Section 4.5.3. In this work, we use a mixture of task-specific relevance criteria and automatically generated ones from the Domain Transition Graph (DTG) (Helmert, 2004) of the problem domain definition. A domain transition graph represents how a single planning variable evolves over time. It consists of:

- Nodes: Represent possible values of a state variable.
- Edges: Represent state transitions caused by actions.

For example, in a pick-and-place task, a causal graph might look like:

$$\text{object\_at}(O, Table) \xrightarrow{\text{pick}(O)} \text{robot\_holding}(O) \xrightarrow{\text{place}(O, Shelf)} \text{object\_at}(O, Shelf)$$

Each edge represents an action that moves the variable from one state to another.

To identify which past executions are relevant for updating an action's cost, we analyze the DTG to determine how similar past state transitions are to the current execution as follows:

1. Each action primarily influences one or more state variables. By extracting the affected variables from the DTGs, we focus only on past executions where this variable underwent a similar transition.
2. We then compare the current execution to past executions by measuring how similar the state transitions are: If a past execution involved the exact same state transitions, it is considered fully relevant. If a past execution only partially involves the state transitions, the amount of similar transitions is used as a weight factor.
3. Additionally, we take predefined criteria into account, although not directly affected by the action and thus not part of the DTG. E. g., all objects at the grasp location are taken into account for a grasp action.

Once similarity is computed, past executions are filtered based on a threshold. If the transitions match exactly, the past execution is included in T_X. If the transitions are similar but not identical, the execution is included with a similarity weighting. If the transitions differ significantly, it is ignored.

7.3.2. Anomaly Detection

Anomalies in action execution, characterized by deviations in execution costs that significantly diverge from expected values, provide valuable insights for learning and adaptation in robotic systems. The recognition of anomalies goes hand in hand with the recognition of similar data, but one step further. In Section 4.5.3, similarity-based forgetting techniques were introduced as a strategy for balancing training data. These methods facilitate the selective retention of relevant information by filtering out redundant or non-informative samples, ensuring that the dataset used for learning remains representative and diverse. The objective of this approach is to retain as many unique and informative instances as possible, particularly those representing anomalies, which often hold critical information about unexpected variations in task execution. Sometimes, however, the exact opposite is the case. One tries to eliminate outliers in the data in order to better approximate the real data. This is also the case with cost approximation.

In our scenarios, we use anomaly detection primarily to avoid unnecessary or incorrect updates that were performed because of outliers. To systematically filter and detect anomalies in our experiments, we employ the following methodologies:

- Z-Score Analysis: The z-score, quantifies the deviation of an observed execution cost from the expected mean, assuming a normal distribution of action execution costs. It is computed by:
$$z_X = \frac{c_X^{\text{real}} - \mu_X}{\sigma_X}$$
where c_X^{real} denotes the real and during execution measured cost of an action X, μ_X represents the measured mean cost, and σ_X is the standard deviation. The z-score

provides a standardized measure of deviation, where a value of $z = 0$ indicates that the execution cost aligns with the mean. A higher absolute z-score signifies greater deviation, with extreme values indicating potential anomalies. Following the thresholding approach proposed by Andrade (2021), an execution is classified as an outlier if its z-score falls outside the interval $[-2, 2]$. This thresholding method is based on empirical observations that suggest values beyond this range are indicative of significant deviations from expected behavior.

- Baseline Model (no anomaly detection): We include an approach where no data is discarded and everything is considered being relevant for learning. In this setting, after each execution phase, all action models are updated without any filtering or anomaly detection mechanisms. This approach serves as a comparative baseline to assess the effectiveness of forgetting-based strategies in improving anomaly detection and model efficiency.

7.3.3. Cost Update

Once anomalies have been detected in the execution of an action, it is necessary to update the model associated with that action. This ensures that future predictions accurately reflect variations in execution cost and improve the system's ability to adapt to dynamic environments. Various approaches can be employed to update cost estimates, each differing in how past data is weighted and incorporated into future predictions.

To systematically evaluate different update strategies, we analyze the following methods:

- Average Forecasting: A straightforward yet effective approach involves computing the mean of all previously observed execution durations to estimate future action costs. This method assumes that all past executions contribute equally to the forecasted cost and that no specific execution carries greater significance than others. The cost estimate c_X given T_X is defined by:

$$c_X \mid T_X = \frac{1}{|T_X| + 1}\left(\sum_{t \in T_X} c_X^{\text{real}}(t) + c_X\right)$$

  where $c_X^{\text{real}}(t)$ represents the recorded execution costs of an action X at execution t. The initial estimate of the costs c_X is also taken into account. While this approach is computationally efficient and robust to noise, it does not account for temporal variations or trends in execution costs, potentially leading to suboptimal predictions when execution costs exhibit significant temporal dependencies.
- Simple Exponential Smoothing: To address the limitations of equal weighting in the averaging approach, Simple Exponential Smoothing (SES) introduces a weighting mechanism where more recent observations contribute more heavily to the updated cost estimation. Inspired by time-based decay models (see Section 4.5.3), this method applies

exponentially decreasing weights to past execution costs, ensuring that more recent costs have a greater influence on future estimates. The cost estimate is computed as follows:

$$c_X \mid T_X = \sum_{i=0}^{|T_X|} \alpha(1-\alpha)^i T_X(|T_X|-1-i) + (1-\alpha)^{|T_X|} c_X$$

where $T_X(i)$ denotes the i-th element of T_X and $0 < \alpha < 1$ is the smoothing parameter, controlling the rate at which past observations lose influence over time. Again, the initial estimate of the costs c_X is also taken into account. As suggested by Gardner and McKenzie (1985), a commonly used value is $\alpha = 0.30$, striking a balance between responsiveness to recent changes and stability in cost estimation. Higher values lead to greater sensitivity to recent executions, making the model more adaptive but potentially more susceptible to noise.

- Seasonal and Trend Decomposition using Locally Estimated Scatter-plot Smoothing (LOESS): To further refine cost estimation, particularly in scenarios where execution durations exhibit cyclical patterns or long-term trends, Seasonal and Trend Decomposition using LOESS can be employed. Originally introduced by Cleveland (1990), this method decomposes a time series into three key components:

$$Y_t = Seasonal_t + Trend_t + Remainder_t$$

where Y_t represents the observed execution cost at time t, $Seasonal_t$ denotes the seasonal component (capturing periodic fluctuations), $Trend_t$ represents the trend-cycle component (capturing long-term variations), and $Remainder_t$ corresponds to the remainder component (representing noise and irregular variations). This decomposition allows for flexible and robust analysis of execution costs by isolating predictable patterns and trends, making it particularly suitable for environments where execution costs vary periodically due to external factors such as task complexity, environmental changes, or system wear. The cost update can then be derived in different ways, depending on the emphasis placed on each component. Two possible update strategies are:

  - Using the Trend Component for Long-Term Adaptation: If the goal is to capture gradual changes in execution costs while filtering out noise, the trend component $Trend_t$ can be used to update the predicted cost for the next execution:

$$c_X \mid T_X = Trend_{|T_X|}$$

    This approach is particularly useful when execution costs evolve due to external factors like hardware degradation or environmental changes.
  - Combining Trend and Seasonality for Periodic Adjustments: For tasks exhibiting periodic execution cost patterns, the predicted cost can incorporate both the trend

and seasonal components:

$$c_X \mid T_X = Trend_{|T_X|} + Seasonal_{|T_X|+1}$$

In our experiments, we use the latter version as it should be able to better predict costs. Here, $Seasonal_{|T_X|+1}$ is the predicted seasonal adjustment for the next execution, based on past seasonal patterns. This approach helps account for recurring variations (e. g., peak execution times during specific phases of a task).

Each of these methods provides a different perspective on updating execution costs, ranging from simple averaging to more sophisticated trend-aware models. By comparing their effectiveness in different experimental settings, we aim to determine the most suitable approach for maintaining accurate and adaptive cost estimates in robotic action execution.

7.3.4. Results

To develop and evaluate a model capable of refining action models based on experience, we conducted experiments using a pick-and-place task, as described in Chapter 3. We use *FastDownward* (Helmert, 2006) as a planning engine and *AIPlan4EU* (Micheli et al., 2025) as a planning framework. The problem domain is formally described in PDDL syntax, as exemplified in Listing 7.1.

Each action is associated with a non-uniform cost value, denoted as c_{grasp}, c_{move}, and $c_{\text{put\_down}}$. These cost values could represent various factors, such as energy consumption, duration[1], or operational risk. In our case, we chose action duration as cost value. However, these assigned costs do not necessarily correspond to the real execution costs $c^{\text{real}}_{\text{grasp}}$, $c^{\text{real}}_{\text{move}}$, and $c^{\text{real}}_{\text{put\_down}}$, but rather serve as initial estimates. Moreover, the real costs may vary dynamically during task execution.

In our simulated experiment, a robot was tasked with repeating the same task multiple times. The task consists of several subtasks and each subtask consists of one or more actions. Between sequential subtasks, the robot has the opportunity to adjust its action models based on the actions it has previously executed.

The user wants the robot to bring all the ingredients for breakfast to the table. The repetitions of this task can be seen as repetitions on different days. Initially, the objects *Soymilk* and *Multivitamin Juice* are located in the refrigerator, while *Nesquik* and *Vitalis Cereals* are placed on the countertop. The robot starts at the center of the room.

The following goals are defined for the tasks:

1. First the user asks to place the soymilk on the table (`object_at(soymilk, onTable)`).

[1]Although PDDL 2.1 introduced `durative-actions` (as employed in Peller et al. (2018)) to model concurrent actions, we assume a sequential execution model for the purposes of this study.

Listing 7.1: PDDL domain of simple pick-and-place-task

```
( define ( domain simple_pick_and_place )
    ( :requirements :strips :typing :negative-preconditions
   :action-costs )
    ( :types navigation_location item item_location )
    ( :predicates
        ( robot_at ?loc - navigation_location )
        ( object_at ?i - item ?loc - item_location )
        ( connected ?loc_0 - navigation_location ?loc_1 item_location )
        ( in_left_hand ?i - item )
        ( in_right_hand ?i - item )
        ( left_hand_empty )
        ( right_hand_empty )
    ( :functions ( total-cost ))
    ( :action move_to
        :parameters (
            ?l_from - navigation_location
            ?l_to - navigation_location )
        :precondition (and
            ( robot_at ?l_from ))
        :effect (and
            ( not ( robot_at ?l_from ))
            ( robot_at ?l_to )
            ( increase ( total-cost ) 50 )
        )
    )
    ( :action grasp_left
        :parameters (
            ?r_loc - navigation_location
            ?item - item
            ?i_loc - item_location )
        :precondition (and
            ( robot_at ?r_loc )
            ( object_at ?i ?i_loc )
            ( connected ?r_loc ?i_loc )
            ( left_hand_empty ))
        :effect (and
            ( not ( object_at ?item ?i_loc ))
            ( in_left_hand ?i )
            ( not ( left_hand_empty ))
            ( increase ( total-cost ) 10 )
        )
    )
    ( :action putdown_left
        ...
    )
    ...
)
```

2. Then, the nesquik should be placed on the table (`object_at(nesquik, onTable)`).
3. Then, the vitalis should be placed on the table (`object_at(vitalis, onTable)`).
4. Finally and after having breakfast, the user wants the robot to tidy up again. All objects should be returned to their original locations (`object_at(soymilk, inFridge)` $\wedge$ `object_at(nesquik, onCountertop)` $\wedge$ `object_at(vitalis, onCountertop)`).

This task is combined with different trials for simulating different behaviors of real-world execution costs.

1. In the first trial, only a small variability in real execution costs is introduced by sampling the real costs from a normal distribution. However, apart from this stochastic variation, the real costs remain unchanged across repeated executions.
2. In the second trial, the real action costs $c^{\text{real}}_{\text{grasp}}$, $c^{\text{real}}_{\text{move}}$, and $c^{\text{real}}_{\text{put\_down}}$ gradually increase over time and are reset to their normal values after each task.
3. In the third trial, actions may fail sporadically with a probability of 5%. This trial accounts for occasional execution failures, such as unstable grasps or navigation failures, leading to significantly increased costs (e. g., due to timeout-induced delays). We assume that help from the user is required after the high costs were incurred, which is why the task can be continued by the robot. If the same or a similar action is performed again in the future, this action can succeed normally again.
4. In the fourth trial, during the second iteration, a failure is introduced where one of the robot's arms becomes unusable, preventing the execution of teh grasping and putdown actions. Similarly to the third trial, if the robot attempts the failed action, it incurs significantly increased costs. Again, we assume that help from the user was required after the high costs were incurred. If the same or a similar action is carried out again, this action will again incur high costs.

Then, each trial is evaluated using a combination of the proposed anomaly detection and cost update methods. E. g., the results of using the baseline anomaly detection method and average forcasting as cost update method is shown in Figure 7.4. After each task the robot compares the estimated duration to the real measured duration of similar actions of the past to refine its models. We assume a robot that can move cheap (around $10\,\text{sec}$ between two locations) but grasping and placing objects takes much longer (around $20\,\text{sec}$ per action). To show the behavior of the proposed methods under worst-case conditions, we assume that the initially estimated costs c_{grasp}, c_{move}, and $c_{\text{put\_down}}$ significantly deviate from the real execution costs $c^{\text{real}}_{\text{grasp}}$, $c^{\text{real}}_{\text{move}}$, and $c^{\text{real}}_{\text{put\_down}}$. However, after multiple executions, the learned cost values should converge toward the real execution costs, enabling more accurate planning and execution.

First, a combination of the baseline anomaly detection (no forgetting) and average forecasting is used. The results of using the combination of the baseline anomaly detection and average forecasting on all trials is shown in Figure 7.5. The plots show that for the first and last trials the model quickly adapts to changes, although the initial values do not match the real values. However, it can also be seen that averaging does not deliver good results with seasonally dependent data. The model always lags behind the real costs up to the point where the costs are hardly ever updated and only reflect the average of the real data, which leads to regular over- or underestimates. Also, trial 4 is special. After the robot's left arm becomes unusable, the model is motivated not to use it at all if possible. Instead, additional actions are taken in order to use only the right arm when e. g., returning the Nesquik and the Vitalis, for example. For this reason, high costs only occur once in the plot (once for grasp and once for putdown).

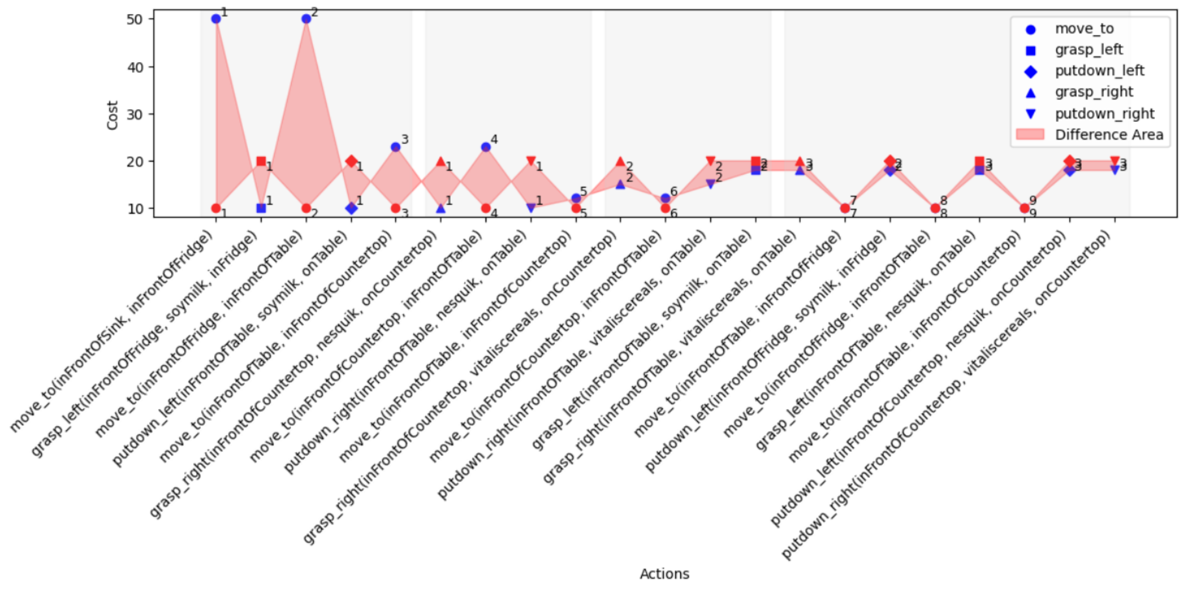
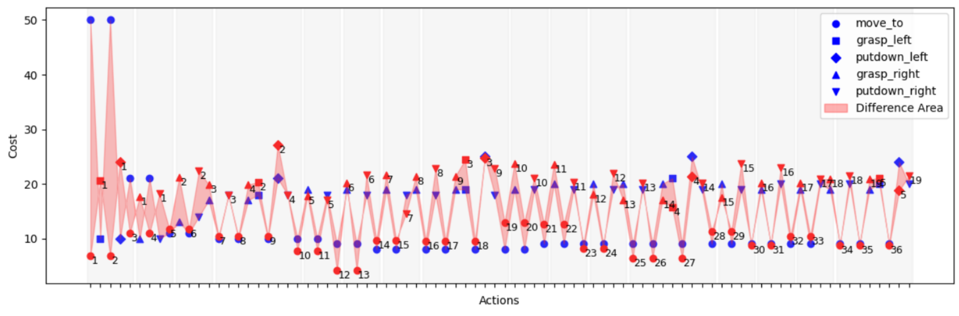


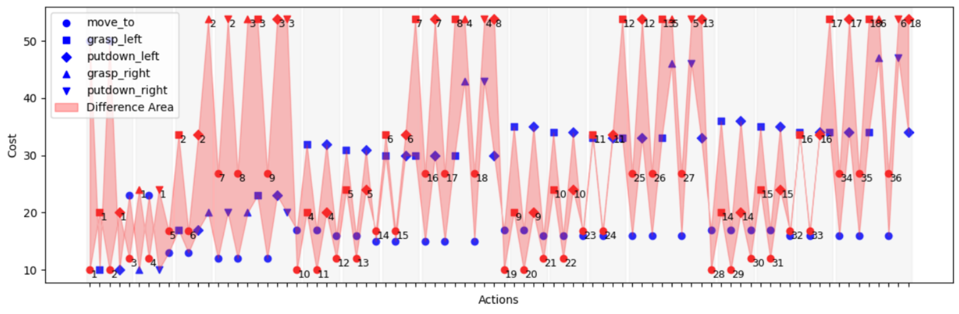
Figure 7.4.: Comparison of real and estimated cost using no forgetting and average forecasting. Each vertical gray bar represents one task, after which the action models may be updated. Only one repetition of the task is shown. Below the plot, the used actions and their parameterizations are shown. The red area between the red points (real costs that were measured after performing an action) and the blue points (estimated cost of that action before executing it) represents the difference between the two. The numbers next to the points denote the number of similar actions that occurred in the past, including the current action.

Second, a combination of the baseline anomaly detection and exponential smoothing is used. The results of all trials are shown in Figure 7.6. The observations are very similar to those when average forecasting is used as cost update method with the difference that average forecasting reacts more quickly to large differences between c_X^{real} and c_X. On the other hand, exponential smoothing does not suffer as much from outliers that lie far in the past as shown in Figure 7.6(c) compared to Figure 7.5(c). As in the experiment with average forecasting, the planner avoids the incorrect arm after the cost update by exponential smoothing and accepts extra actions for this.

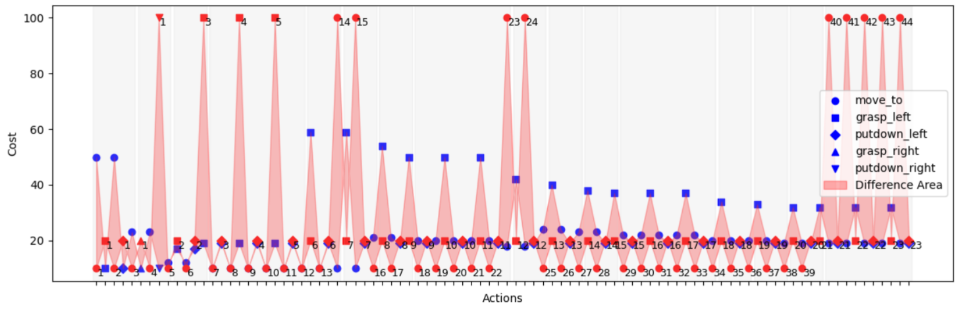
Third, Seasonal Trend Decomosition using LOESS (STL) is used as cost update method. STL expects a periodic pattern to be recognizable, hence it does not update its model during the first two executions of the task. Further, STL expects the period to be known. Unfortunately, the amount of elements per period can vary due to newly developed strategies based on the updated action models when repeating a task. For this reason, we use Fast Fourier Transformation (FFT) to estimate the periodicity and use STL to analyze trend, seasonal and remainder. The results of all trials is shown in Figure 7.7. Due to the inability to update during the first two tasks, this method cannot be compared with the previous methods. Surprisingly, however, STL performs well in the tasks where average forecasting and exponential smoothing deliver poor results. Figure 7.7(b) shows that after a few repetitions, the repeating pattern is recognized and the costs can be estimated with little error. Figure 7.7(c) also shows that – after collecting data from several tasks – the estimated costs are successfully adjusted to real ones. STL ignores the random failures that do not fit the pattern and therefore approximates



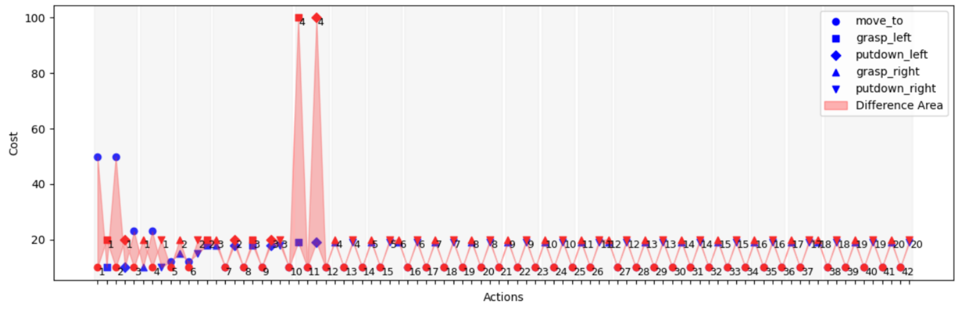
(a)



(b)



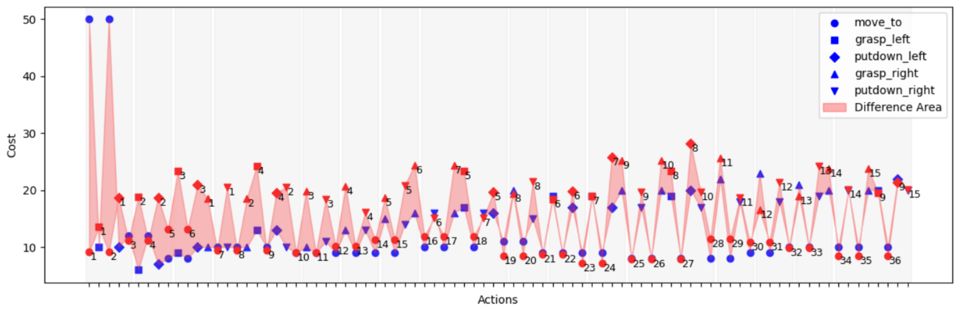
(c)



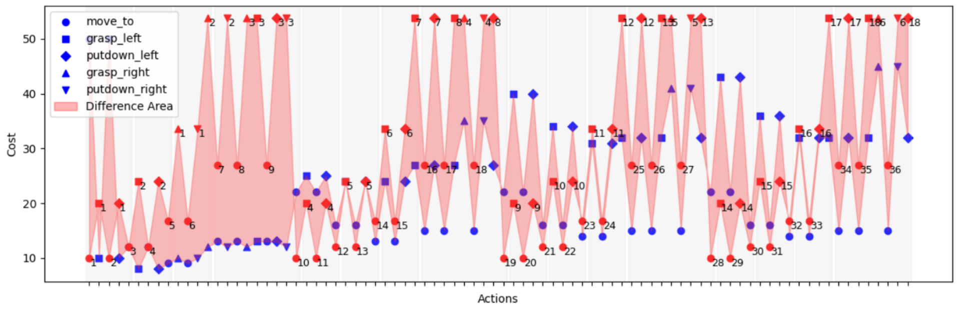
(d)

Figure 7.5.: Comparison of real and estimated cost using no forgetting and average forecasting. Figure 7.5(a), Figure 7.5(b), Figure 7.5(c) and Figure 7.5(d) show the results of the aforementioned methods on the first, second, third and forth trials.

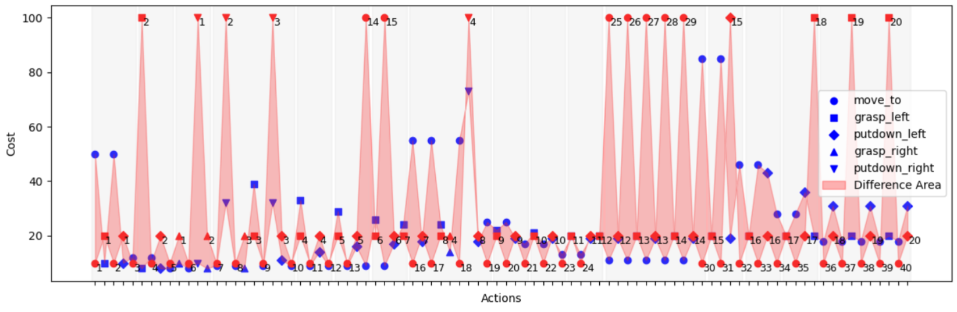
the costs better than the previous methods. In Figure 7.7(d), however, the permanent change in costs yield that no pattern can be recognized, which means that the model for the actions for the malfunctioning arm cannot be updated. For this reason, the robot continues to try to use this arm, resulting in large errors between measured and estimated action execution time.



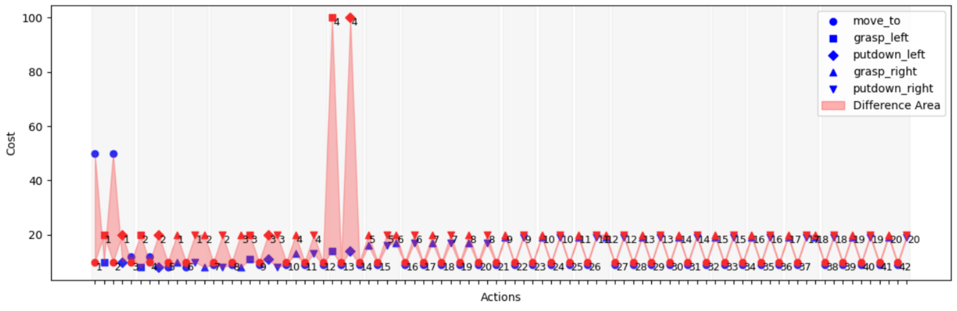
(a)



(b)



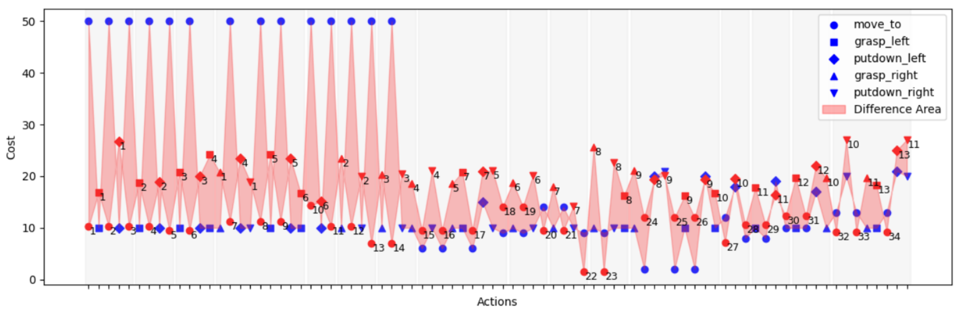
(c)



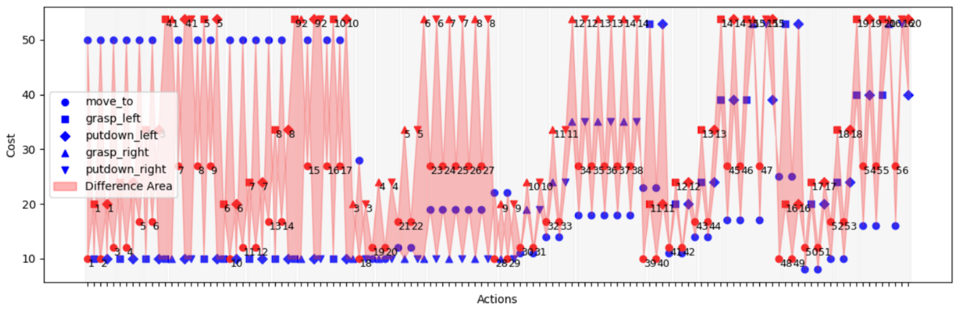
(d)

Figure 7.6.: Comparison of real and estimated cost using no forgetting and exponential smoothing.

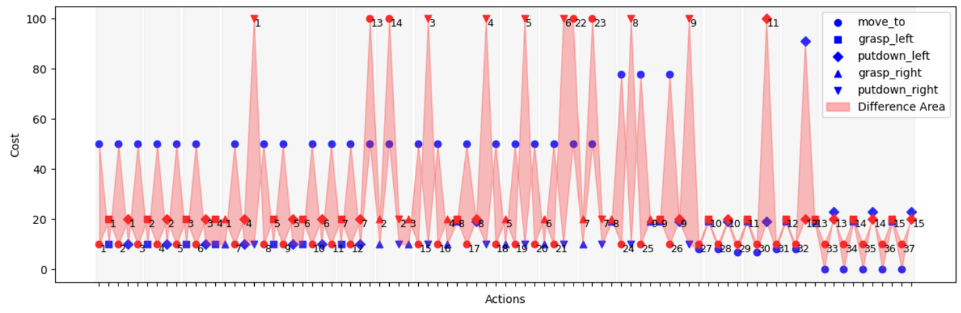
The plots when using the z-score metric to filter out surprising data and outliers are shown in Figure A.1, A.2 and A.3. Updates with negative effects on the overall performance can be avoided, especially in the case of significant outliers, as in trial 3. The numbers next to the data points, which symbolize the number of updates including the current date, illustrate this. Overall, around 10% of updates could be avoided by filtering. The accuracy of the data (i. e., the area between the red and blue data points) has only improved marginally.



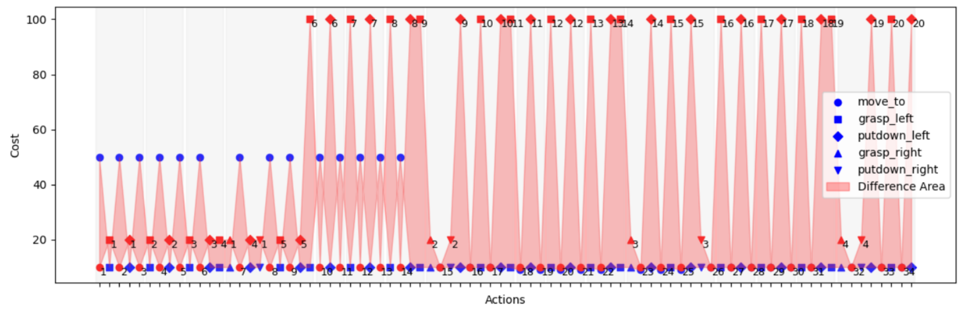
(a)



(b)



(c)



(d)

Figure 7.7.: Comparison of real and estimated cost using no forgetting and STL.

Action	①	②	③	④
`move_to`	5 s	∼11.25 s	12 s	12 s
`grasp_left`	10 s	∼27.1 s	21 s	22 s
`grasp_right`	20 s	∼29.7 s	20 s	25 s
`handover_left`	10 s	∼12.66 s	12 s	11 s
`handover_right`	20 s	∼14.9 s	20 s	18 s

Table 7.2.: Initial (①), measured (from real executions ②) and updated durations (③ after 1st plan execution and ④ after 2nd plan execution) for actions in the real robot experiment using the humanoid robot ARMAR-7. The robot uses the initial estimate and the measured real duration of an action to update its internal model of the action duration using average forecasting so that the estimate better approximates the real cost on the next execution.

In a real-world robotic experiment, the humanoid robot ARMAR-7 was assigned the task of delivering multiple objects to a human subject. Analogous to the simulated experiments, the initially predicted action durations deviated significantly from the observed execution times. Moreover, the preliminary model assumed that the robot's right arm would operate at a slower pace, thereby attributing higher costs to any actions involving the right arm. However, empirical evidence gathered during the experiment contradicted this assumption, prompting the robot to iteratively refine its suboptimal action model based on accrued experience. The initial and updated durations for various actions are summarized in Table 7.2[2]. In this study, we employed an average forecasting model for cost updating.

Initially, the robot was positioned at the center of the room close but not in front of the kitchen counter, with objects arranged on a countertop (also shown in Figure 3.2). To transfer an object to the human, the robot was required to navigate to a specified location. With the initial cost assumptions, the planner generated the following sequence of actions for handing over two objects from the countertop to the human:

```
move_to(countertop) → grasp_left(soymilk) → move_to(human) → handover_left(soymilk)
                    → move_to(countertop) → grasp_left(juice) → move_to(human)
                    → handover_left(juice)
```

Because of the presumed higher costs associated with the right arm, the planner exclusively utilized the left arm. The estimated duration of this plan was 60 s; however, the actual execution required 141 s, thereby indicating a significant underestimation in the initial cost model. Consequently, the cost parameters were updated after the first execution, as detailed in Table 7.2.

[2]Please note that the actual execution times depend on various factors, such as the distance to the target pose, the object's graspability or the human reaction. The values given here are averaged over all executions.

Subsequently, when the robot was tasked with an identical operation after further experiential learning, it employed updated action models to compute an alternative plan:

$$\texttt{move\_to(countertop)} \rightarrow \texttt{grasp\_left(soymilk)} \rightarrow \texttt{grasp\_right(juice)} \rightarrow \texttt{move\_to(human)}$$
$$\rightarrow \texttt{hand\_over(soymilk)} \rightarrow \texttt{hand\_over(juice)}$$

In this revised plan, the increased costs associated with left-arm actions and the navigation action (`move_to`) resulted in the inclusion of the right arm, yielding an estimated makespan of $97\,\mathrm{s}$. With further experience, the action models were updated once more, finally converging to an optimal plan that more accurately approximated the actual execution duration. This iterative process of model refinement underscores the efficacy of leveraging experiential learning to enhance planning quality and reduce the discrepancy between predicted and observed performance.

7.4. Discussion

In robotics, the embodiment of agents and their interaction with the physical world lead to variations in action durations, even for the same symbolic action. These variations arise due to factors such as sensor noise, actuator wear, environmental disturbances, and dynamic task requirements. Consequently, it becomes necessary to adapt action cost models dynamically based on experience. Failure to do so can lead to inefficient planning, suboptimal decision-making, and an increased risk of execution failures. To address this challenge, various methods for cost estimation were introduced and evaluated across different scenarios, each representing distinct cost behaviors in real-world conditions.

The results indicate that all proposed models improve upon the initial worst-case assumptions, albeit with varying strengths and weaknesses. Simple methods such as averaging and exponential smoothing perform well when the actual cost of an action remains relatively stable over time. Outlier filtering techniques can further enhance their reliability by eliminating erroneous cost fluctuations caused by external disturbances or sensor inaccuracies. However, these methods struggle in environments where costs fluctuate dynamically due to changing task demands or environmental conditions, failing to provide accurate predictions.

In contrast, approaches that decompose cost data into seasonal components offer superior predictive capabilities in such cases. These methods leverage periodic patterns in the data, allowing the system to anticipate cost variations over time. However, they require sufficient historical data to identify recurring trends, which introduces a delay before meaningful predictions can be made. Furthermore, if no discernible pattern exists, these methods fail entirely, making them unsuitable for highly unpredictable environments.

A hybrid approach emerges as a promising solution to overcome these limitations. Initially, outliers are removed from the dataset after verifying cost consistency. Subsequently, continuous analysis determines which method would have best approximated past data. This enables the system to rely on averaging techniques during stable phases, while switching to more sophisticated forecasting methods when a seasonal pattern becomes evident. Also, considering a time window of past data may be beneficial. Such an adaptive strategy ensures robust cost estimation across a wide range of operational conditions.

However, this approach represents only a preliminary step toward estimating the dynamically changing real-world costs of an action. In complex environments, cost estimation should not rely solely on historical action data. Instead, environmental factors, task context, and execution constraints play a crucial role in predicting the success or failure of an action. Ideally, the system learns that certain situations require specific actions or parameterizations over others, depending on prior success or failure.

Also, action costs may depend on multiple interacting parameters. As demonstrated in Peller et al. (2018), the cost function can be influenced by both the action itself and an associated acceleration factor. Notably, the relationship between cost and acceleration may be nonlinear (as in Figure 7.1), further complicating cost estimation.

To identify such interdependencies and inform plan adaptation, it becomes necessary to incorporate a broader, more integrated understanding of the environment and task dynamics into the planning process. LLMs are a compelling alternative planning engine for this purpose (Bärmann et al., 2024b; Birr et al., 2024). Due to their extensive pretraining on diverse textual data, LLMs encapsulate a rich body of commonsense and procedural knowledge about the world, including how different actions tend to affect outcomes under varying conditions. This makes them particularly well suited for learning from experience – especially from interactive, language-based feedback such as corrections from a human operator as demonstrated in Bärmann et al. (2024b).

This work has demonstrated that LLMs can be employed not just as language understanding modules but as central planners in a closed-loop system that orchestrates robot behavior. By generating code-level commands (e. g., Python) in response to high-level instructions, and incorporating feedback from both perception and execution, LLMs can iteratively refine their behavior over time. Importantly, when an LLM-controlled robot misinterprets a command or performs a task incorrectly, a secondary LLM can be prompted to revise the generated behavior based on human feedback. This corrected plan is then stored in episodic memory and retrieved in the future when semantically similar situations arise. In this sense, incremental learning of plans becomes a form of experience-based planning, where past successes and failures influence future decisions.

Compared to symbolic approaches such as PDDL, which require explicitly modeled domain and cost functions, LLMs offer a more flexible and scalable way to incorporate nuanced contextual knowledge and adapt behavior through interaction. Moreover, they can generate context-sensitive plans even in the absence of explicit models, and their behavior can evolve over time as more interaction data is accumulated. While symbolic planners excel at deterministic reasoning over well-defined domains, LLMs bring the ability to generalize across ambiguous, open-ended, and linguistically specified tasks—especially when grounded in real-world feedback.

Thus, integrating LLM-based planning into experience-driven robotic systems opens up new directions for cost-sensitive, adaptive decision-making. It bridges low-level execution realities with high-level behavioral abstraction, and complements traditional cost adaptation strategies by enabling a more holistic understanding of when, why, and how certain plans should change.

At this point, however, it should also be noted that at the time of writing this paper, LLMs are not able to generate correct plans with a long time horizon. They are particularly suitable for planning the next step, not for planning a long sequence of actions. For this reason, LLMs are often only used as an interim solution, e. g., to generate target descriptions from natural language or to subsequently analyze action executions (Birr et al., 2024). Classical algorithms are then used for planning, which guarantee to find a plan if one exists.

Next to the employment of LLMs, future research could explore other sophisticated machine learning models, such as Bayesian inference or reinforcement learning, to capture complex dependencies and improve the adaptability of cost models in real-world robotics applications.

CHAPTER 8

Conclusion

This thesis has explored the design, implementation, and evaluation of a memory system for robot cognitive architectures, focusing on its role as a mediator between high-level cognitive functions and low-level sensorimotor processes. By addressing key research questions, this work contributes significantly to the understanding and development of memory mechanisms in cognitive robotics. The following sections summarize the scientific contributions, discuss existing limitations, and outline directions for future research and improvements.

8.1. Scientific Contributions

By identifying key characteristics, implementing an extensible and efficient memory architecture, and rigorously evaluating its performance in real-world scenarios, this research establishes a solid foundation for advancing robotic cognition. The following sections summarize the primary scientific contributions of this work.

8.1.1. Identification of Key Characteristics for a Memory System in a Cognitive Control Architecture

One of the major scientific contributions of this work is the identification of essential properties required for an effective memory system in cognitive robotics as outlined in Peller-Konrad et al. (2023). Unlike conventional approaches that view memory as a simple storage mechanism, we have demonstrated that a memory system in a robotic cognitive architecture must be *active*, *episodic*, *multi-modal*, *associative*, and *introspectivev* to support higher-level cognition. These properties ensure that memory is not just a passive repository but a dynamic component of the robot's decision-making and learning capabilities.

Additionally, the system requires key technical requirements: a *distributed* architecture, *efficiency* in retrieval, and *long-term* retention capabilities. A distributed architecture ensures robustness and scalability, while efficiency is maintained through structured memory indexing and different retrieval techniques. Long-term retention is made possible through sophisticated memory consolidation and forgetting mechanisms, which allow the system to manage data over extended operational periods effectively.

By integrating these characteristics, this research establishes a foundation for memory systems that enhance adaptability, learning, and reasoning in robotic cognitive architectures. The proposed framework enables robots to process and leverage past experiences, thereby improving their ability to interact with humans and dynamic environments in a meaningful way.

8.1.2. Implementation of Memory System and Knowledge Representation Framework

Building on the conceptual framework, we introduced a memory-centric cognitive architecture implemented within the ArmarX framework (Peller-Konrad et al., 2023). To meet the defined requirements, a novel knowledge representation was introduced, enabling the memory system to unify and analyze knowledge during runtime. This knowledge representation, built upon variants, is highly extensible and optimized for ease of use by developers. Despite the dynamic nature of the variant type, it supports static type checking, auto completion, and other features, facilitated through code generation.

The developed memory-based architecture integrates Working Memory and Long-term Memory in a seamless manner, providing structured data management and efficient memory retrieval. Arbitrary knowledge is stored in this distributed memory in a structured manner, with all data treated to be episodic. This approach, combined with the aforementioned standardized knowledge representation, enables the memory to be constructed in the most flexible and modular way possible, as all data types can be stored and managed uniformly.

Forgetting mechanisms in LTM ensure the system can efficiently manage large volumes of data. The knowledge is regularly assessed for redundancy or irrelevance and undergoes filtering. Several methods have been proposed to address this. Specifically, long-term memory knowledge can be generalized, with a data-driven approach utilizing autoencoders introduced for this purpose. The architecture and training of these models not only facilitate knowledge reconstruction for reuse but also enable the prediction of new knowledge based on prior experience.

8.1.3. Evaluation of Efficiency and Usability of Memory System in Verbalization and Experience-based Planning

The effectiveness of the proposed memory system was evaluated through two core applications:

Verbalization of Past Experiences (Bärmann et al., 2021): We demonstrated a system able to recall and articulate past experiences from the deep episodic memory. This capability is crucial for explainability and trust in human-robot interactions, as it enables robots to provide explanations for their decisions and actions. To achieve this, a data-driven approach was employed, where various tasks were automatically carried out by a robot within the simulation of the robot software framework ArmarX and stored in the robot's episodic memory. Question-Answer (QA) pairs were then generated for these tasks and used to train the verbalization network. A key feature of this approach is that the model only receives the question, not the corresponding episode, as input at the time of querying. The experiential knowledge has already been encoded in a separate step beforehand. Through the use of attention mechanisms, the model learns to map the input to the correct entries in the episodic memory. The results of this work demonstrate that the generated questions are successfully answered with a short history. However, the model struggles with longer histories or when faced with real, human-asked questions. As a potential improvement, LLM-based verbalization was explored.

Experience-based Symbolic Planning (Peller et al., 2018): The memory system was utilized to enhance planning capabilities, allowing robots to leverage past experiences for improved task execution. This capability was tested in dynamic scenarios where robots were required to adapt their behaviors based on prior experiences, showcasing the importance of episodic memory in robotic cognition. A task was presented in which the robot was instructed to perform time-critical tasks within a specified time frame. The robot had the option to accelerate or decelerate its actions, which directly influenced the risk of failure associated with those actions. We introduced TSFD, a temporal planner capable of finding optimal plans for temporally constrained tasks involving such stressed actions, delivering better quality results than other temporal planners. Initially, the relationship between action acceleration and the risk of failure is defined manually, though it is expected to be learned from experience over time. We identified three key contributors to this process: similarity measurement (identifying similar data in episodic memory), anomaly detection (filtering outliers), and cost update strategies (determining how to update the existing model of an action). In an evaluation study, we compared different methods for each of these contributors, and the results indicated that a combination of various methods tailored to different use cases produced the best outcomes.

8.2. Outlook

The memory framework presented in this work has evolved into a core system within our robotic platforms. Beyond the two use cases discussed in detail, the memory system is employed in nearly all of our research activities, including domains such as navigation and mobile manipulation (Pohl et al., 2024). This widespread adoption is reflected in the extensive and continuously growing number of supported data types and memory servers (see Section A.2 and A.3).

A key factor in the success and adoption of the framework is that its design prioritized not only innovation but also usability and scalability. These principles were fundamental from the beginning of the conceptualization of the framework. The introduction of the memory system has led to a more structured and unified software architecture, where components previously depended on tightly coupled, ad hoc communication mechanisms, the memory now provides a clear separation and a standardized approach to data storage and retrieval.

In addition to this architectural unification, the framework includes a range of practical features such as import/export capabilities, management of time series of data, data type generation, and predictive functionality. These have significantly streamlined the development process and enhanced the flexibility of the system in both research and deployment settings.

Nevertheless, the development of the memory system and thus the cognitive architecture is still in its early stages. The development of a cognitive system is an ongoing process that typically spans several years up to decades (Kotseruba and Tsotsos, 2018), and there are numerous areas where the memory system can be enhanced to better meet the requirements of complex cognitive architectures, particularly as the demands of robotic applications continue to evolve.

One of the key areas for future development is expanding the memory's ability to handle and retrieve information in a more sophisticated manner. Currently, the memory system supports temporal structures and temporal-based queries, reflecting how knowledge is managed in a linear sequence. However, human episodic memory is not solely temporal – it also organizes knowledge spatially and contextually, as done in Peters et al. (2001). Implementing spatio-temporal and contextual access to stored information would significantly improve the efficiency and flexibility of data retrieval, enabling the system to better mimic human cognitive processes. This would also open the door to more advanced memory architectures capable of handling richer and more complex types of knowledge.

Also, uncertainty and probabilities are not explicitly modeled, even though handling uncertainty is essential for robust cognitive systems (Paulius and Sun, 2019). Incorporating probabilistic reasoning would enable the memory system to evaluate the reliability and relevance of stored information, make informed decisions under ambiguity, and dynamically update beliefs based on new evidence. This approach would enhance the system's ability to handle noisy or incomplete data, prioritize information retrieval based on confidence levels or expected utility, and align more closely with human cognitive processes. While integrating probabilistic methods poses challenges, such as computational complexity, advancements in probabilistic programming and approximate inference techniques offer promising solutions for making the memory system more adaptive and resilient in complex environments.

An important aspect of the future development of the memory system is the employment of attention mechanisms. Attention plays a crucial role in efficiently processing and retrieving relevant information from episodic memory. Currently, the system employs no method to focus on the most salient information apart from simple statistical methods. By implementing complex attention models, the system could prioritize specific memories based on their relevance to the task at hand, the context, or the novelty of the experience. This would allow the robot to dynamically adapt its memory retrieval process, ensuring that it draws on the most pertinent and useful knowledge, rather than overloading its cognitive resources with irrelevant information. Moreover, attention mechanisms could be used to facilitate more sophisticated filtering and refinement processes, improving overall system performance in complex, time-sensitive scenarios.

Related to attention, another concept to explore is the role of surprise in the memory system. Surprise occurs when an experience deviates significantly from expectations, signaling that something novel or unexpected has occurred. Incorporating a surprise-driven mechanism into the memory system would allow the robot to prioritize and focus on such anomalous experiences, as these may offer valuable insights for learning and adaptation. Experiences marked by surprise could trigger the updating of models, the refinement of memory representations, or even the generation of new knowledge structures. This would enable the system to be more responsive to changes in its environment, continually adapting its knowledge base to reflect novel and unexpected events. By integrating surprise as a key component, the robot could develop a more dynamic and flexible memory system capable of handling complex, unpredictable tasks and environments with greater efficiency.

Further, the integration of augmentation techniques into the memory system could be promising, as presented in Figure 5.14 During offline phases, when the robot is not engaged in real-time tasks, it could leverage its knowledge stored in episodic memory alongside specialized augmentation engines to modify or enhance its stored experiences. These augmentation engines could simulate hypothetical scenarios, explore alternative actions, or introduce new variables to the experiences, allowing the system to explore a broader range of possibilities and improve its decision-making in future tasks. For instance, the robot could simulate the outcomes of different strategies or refine its actions based on prior experiences, creating enriched or diverse knowledge representations. This process of augmenting existing memories in LTM could result in a more flexible and adaptive memory system, capable of not only recalling past experiences but also refining and improving its knowledge base for better performance in dynamic and unpredictable real-world environments.

Another area that requires attention is the refinement of models within the LTM. Currently, the system generates new models for each work cycle or discards old models and retrains a new one by concatenating previously encoded experiences with the current data as described in Algorithm 4. While this approach is space-efficient, it carries the risk of introducing errors during decoding and then training, which can affect the accuracy of future models. Incremental learning approaches would allow for the continuous refinement of models without the need to discard valuable prior knowledge, ensuring that the memory system becomes more efficient and reliable over time.

The evaluation and learning capabilities of the memory system also need to be enhanced. One promising direction is the ability to compare and learn from the memories of multiple robots. Sharing knowledge between robots – much like the human exchange of experiences – would improve the overall efficiency of learning and task performance. This could be further extended to allow for the sharing of memories with humans, enabling collaborative and adaptive learning environments. However, such sharing of memories also raises important concerns related to security and privacy, which must be addressed at the memory system level. To that end, we have already begun exploring privacy-aware memory systems and cognitive architectures to ensure that the exchange of data does not compromise sensitive information.

These outlined improvements represent just a small portion of the possible advancements for the memory system within our cognitive architecture. We believe that the architecture we have designed is both general enough to support a wide range of cognitive applications and specific enough to meet the complex demands of AI-driven robotics. As we continue to develop and refine this memory system and the cognitive functions using it, we aim to create a robust foundation for building intelligent, cognition-enabled robotic systems, particularly those in complex applications such as humanoid robots.

Appendices

APPENDIX A

Supplementary Material

A.1. Detailed Description of Cognitive Architectures

The individual related cognitive architectures referenced in Chapter 2 are discussed in more detail below. In addition to the work of the respective authors, reviews of cognitive architectures are used as sources of information (Kingdon, 2008; Kotseruba and Tsotsos, 2018; Langley et al., 2009; Vernon, 2014a; Vernon et al., 2007; Ye et al., 2018).

A.1.1. Executive-Process/Interactive Control (EPIC)

The EPIC (Kieras and Meyer, 1997) cognitive architecture is a computational framework designed to model human cognitive, perceptual, and motor performance in real-time tasks. It emphasizes parallel processing, enabling simultaneous execution of cognitive, perceptual, and motor operations. EPIC's architecture is grounded in the principles of cognitive psychology and focuses on understanding and predicting human behavior.

EPIC's memory system cannot be described as active. The architecture features working memory and long-term memory, with working memory handling short-term, task-specific information and long-term memory storing procedural and declarative knowledge. Working memory acts as a simple buffer that supports task execution by maintaining information needed for immediate processing. The architecture's executive process controls the flow of information, dynamically scheduling tasks and prioritizing actions based on goals.

Episodic memory is not explicitly implemented in EPIC. The architecture lacks a dedicated component for storing and recalling specific sequences of past events. Instead, EPIC's memory model revolves around working memory and long-term memory, which are designed to support immediate task requirements and procedural skills. While EPIC can simulate task performance over time and track the state of cognitive, perceptual, and motor processes, it does not store experiences as coherent, retrievable episodes. This absence reflects the architecture's emphasis on task modeling rather than capturing rich, temporally structured experiences.

The multi-modal memory characteristic is also not fulfilled in EPIC. The architecture is designed to handle perceptual input from different sensory modalities, such as vision and audition, and can integrate these inputs to perform tasks. However, EPIC's multi-modal integration focuses on processing perceptual data in real-time for task execution rather than forming persistent, multi-modal memory representations. Perceptual processing is encapsulated in specialized modules that interpret sensory data and update working memory. The architecture can associate sensory inputs with task goals and responses, but this integration is limited to the immediate context. As a result, EPIC cannot maintain rich, multi-modal memories over extended periods.

Associative memory in EPIC is limited and implicit. The architecture does not feature a dedicated associative memory component but can create task-specific associations through working memory and long-term memory. During task execution, EPIC associates perceptual inputs with responses and goals, enabling context-sensitive behavior. For example, visual input can trigger appropriate motor actions based on stored procedures. However, these associations are strictly task-oriented and are not stored as enduring relationships.

EPIC's introspective capabilities are present through pure symbolic representations. The architecture can monitor the state of cognitive, perceptual, and motor processes to adjust task execution dynamically. The executive process allocates resources, tracks progress, and resolves conflicts between tasks. This form of introspection is task-focused and functional, enabling efficient performance rather than self-reflection. Also, EPIC does not engage in meta-cognition in the sense of reasoning about its own cognitive processes or assessing its own performance beyond task requirements. The architecture lacks mechanisms for self-awareness, reflection, or evaluation of strategies. This constraint arises from the emphasis on modeling cognitive control and multitasking rather than exploring deep self-awareness.

Learning in EPIC is highly constrained. The architecture does not support learning in the traditional sense, as it is designed to simulate performance rather than adapt or learn from experience. Knowledge in EPIC is typically pre-defined, reflecting procedural and declarative aspects of human cognition.

In summary, EPIC lacks dedicated episodic and associative memory components, and multi-modal integration is limited to immediate task execution. Introspection is present but task-focused, and genuine learning capabilities are absent.

A.1.2. ICARUS

The ICARUS cognitive architecture (Langley and Choi, 2006) is a symbolic, goal-driven framework designed to support intelligent agents capable of perceiving, reasoning, learning, and acting in dynamic environments. It integrates cognition and perception through hierarchical representations and emphasizes the use of long-term memory for planning and decision-making.

ICARUS can be considered active in its use of memory, as it dynamically leverages stored knowledge to support reasoning, problem-solving, and decision-making (Kotseruba and Tsotsos, 2018). The architecture is structured around two main types of long-term memory: concept memory and skill memory. Concept memory encodes declarative knowledge about objects, properties, and relationships in the environment, while skill memory stores procedural knowledge for achieving goals. During execution, ICARUS continuously monitors the environment, applies stored skills, and adapts behavior based on context. This active use of memory enables real-time, goal-directed behavior, as the architecture can reason about goals, evaluate alternative actions, and select appropriate plans. Memory retrieval is tightly coupled with decision-making, allowing ICARUS to operate reactively and deliberately.

Episodic memory in ICARUS is notably absent as a dedicated memory component. The architecture does not explicitly store temporally ordered sequences of past experiences. Instead, it relies on conceptual and procedural memories to handle reasoning and learning. This reflects ICARUS's focus on problem-solving and task execution rather than detailed recall of specific past events. However, the architecture can indirectly mimic episodic memory by encoding environmental states and decision contexts within skill memory. When an agent learns a new skill from experience, it implicitly captures the context of prior actions. This approach lacks the richness of true episodic memory found in biological systems, where individual experiences are stored and retrievable as coherent episodes.

The multi-modal memory characteristic is not fulfilled in ICARUS. The architecture processes perceptual input to recognize objects and their properties, forming structured representations in concept memory. These representations are symbolic rather than multi-modal in the sense of fusing rich sensory data. ICARUS can distinguish objects based on visual and spatial attributes but does not integrate multiple sensory modalities (e. g., touch, sound) in a holistic manner. Perception is treated as a preprocessing step rather than a dynamic, multi-modal memory. This focus on symbolic perception limits the architecture's ability to handle raw sensory data and reduces the richness of perceptual experiences compared to biological cognition.

Associative memory in ICARUS is implicitly present through its hierarchical structure and skill learning. The architecture stores relationships between objects, properties, and actions, enabling associative reasoning. For instance, ICARUS can recognize that approaching a door and applying force typically results in the door opening, representing this relationship in skill memory (Tenorth et al., 2010). However, these associations are prespecified or learned through task execution rather than autonomously formed. The architecture does not create novel or abstract associations beyond the scope of predefined concepts and skills. This constraint reflects the challenge of enabling flexible, emergent associations in symbolic cognitive architectures.

The introspective property in ICARUS is possible, through pure symbolic representations, but is limited. The architecture can reason about its own goals and progress through hierarchical decomposition of tasks. It tracks goal achievement and detects failures, leading to re-evaluation and replanning. However, this form of introspection is focused on action monitoring and adaptation rather than self-reflection or deep meta-cognition. ICARUS lacks the ability to reflect on its own cognitive processes, evaluate alternative strategies at a meta-level, or reason about its own reasoning.

ICARUS excels in symbolic learning but does not support sub-symbolic learning. The architecture can acquire new skills through experience by observing outcomes, storing procedures in skill memory, and refining decision policies. This symbolic approach supports incremental learning and adaptation in dynamic environments. However, ICARUS does not natively incorporate sub-symbolic methods such as neural networks or statistical learning.

In summary, ICARUS fulfills the active and associative memory characteristics effectively, albeit with limitations in dynamic association formation. The architecture lacks a dedicated episodic memory and does not support introspection and multi-modality due to its symbolic nature.

A.1.3. Deep Episodic Memory (deep EM)

The deep EM (Bärmann et al., 2021; Rothfuss et al., 2018) system is a cognitive architecture designed to model human episodic memory by capturing, storing, and retrieving temporally structured experiences. It integrates deep learning techniques with cognitive memory principles, focusing on the formation and recall of rich, context-dependent episodes.

Deep EM's memory system can be characterized as active due to its dynamic management of experiences during encoding, storage, and retrieval. The architecture actively captures perceptual inputs, agent actions, and environmental changes, storing them as coherent episodes. During retrieval, deep EM can recall relevant episodes based on cues, enabling context-sensitive reasoning and decision-making. This active retrieval process leverages associative recall mechanisms, allowing partial or noisy cues to retrieve complete episodes.

The episodic memory characteristic is implemented in deep EM, which is explicitly designed to model representation learning and episodic memory prediction. The architecture captures experiences in a temporally structured manner, recording detailed sequences of perceptual inputs, actions, and outcomes. Deep EM's episodic memory enables the recall of specific past experiences, supporting reasoning about similar scenarios. This capability is central to the architecture's design, providing a robust mechanism for learning from experience. The system leverages deep neural networks to encode complex patterns in episodes, enabling it to handle noisy and incomplete inputs. Unlike traditional symbolic approaches, deep EM's episodic memory benefits from sub-symbolic processing, allowing for flexible, context-dependent recall.

deep EM effectively handles multi-modal memory by integrating sensory information from various modalities into comprehensive episodes. The architecture can store and retrieve experiences containing visual, auditory, and contextual data. Also, the associative memory property is strongly supported in deep EM, which uses deep neural networks to learn associations between inputs, actions, and outcomes.

deep EM's introspective capabilities are poor, as the learned representations are not explainable. It does not support symbolic learning. It lacks mechanisms for explicit rule formation or symbolic reasoning. The architecture's strength lies in pattern recognition and associative recall rather than logical deduction or symbolic manipulation. This focus aligns with the goal of modeling episodic memory rather than emulating higher-level cognitive functions.

In summary, deep EM effectively fulfills the active, episodic, multi-modal, and associative memory characteristics.

A.1.4. Multilevel Darwinist Brain (MDB)

The MDB (Bellas et al., 2010) cognitive architecture, inspired by evolutionary principles, integrates multiple levels of selection and adaptation to generate intelligent behavior.

MDB's memory system is active due to its evolutionary adaptation mechanisms. The architecture continuously adapts its behavior through multi-level evolutionary processes, where competing strategies are evaluated, selected, and refined over time.

The episodic memory characteristic in MDB is not explicitly implemented in the sense of retaining past experiences as rich, temporal narratives. Instead, it encodes behavioral strategies and evolutionary adaptations that emerge from historical interactions.

MDB's multi-modal memory is partially realized as it does not include symbolic representations The architecture handles sensory inputs from multiple sources, integrating information to guide behavior, but it does not maintain a centralized, structured multi-modal representation.

The associative memory property is strongly supported in MDB. The architecture forms associations between environmental stimuli, actions, and fitness outcomes.

MDB's introspective capabilities are very poorly implemented. The architecture does not engage in explicit meta-cognitive reasoning but demonstrates a form of implicit introspection. The multi-level selection process enables MDB to evaluate and refine behavioral strategies based on performance. This self-monitoring allows the architecture to adjust its behavior over time.

MDB's learning capabilities are sub-symbolic and emergent. Learning occurs through evolutionary adaptation, selection, and variation, with strategies gradually improving over generations. This population-based learning mechanism refines behaviors in a way that mirrors biological evolution. Symbolic learning is not directly supported.

In summary, MDB effectively fulfills the active and associative memory characteristics, with episodic, multi-modal, and introspective properties only partially implemented.

A.1.5. Intelligent Soft Arm Control (ISAC)

The ISAC (Kawamura et al., 2008) cognitive architecture is a hybrid system designed for autonomous robots, emphasizing perceptual processing, cognitive control, and adaptive behavior. It integrates memory systems to support complex, real-time decision-making in dynamic environments.

ISAC's memory system can be considered active due to its dynamic management of perceptual, procedural, and cognitive information during task execution. The architecture consists of three primary memory components: Working memory, perceptual memory, and episodic memory. Working memory temporarily holds information relevant to current tasks and interactions, enabling real-time decision-making and adaptive behavior. Perceptual memory stores sensory input from the environment, supporting recognition and context awareness. Episodic memory captures sequences of events, enabling ISAC to reflect on past experiences. The active nature of ISAC's memory system is evident in its continuous updating and retrieval of information based on the agent's goals and environmental state. This enables ISAC to monitor its surroundings, make context-sensitive decisions, and adjust actions in real time.

ISAC's episodic memory characteristic is poorly implemented. The architecture's episodic memory records experiences as sequences of perceptions, actions, and results, providing a structured representation of events. This allows ISAC to recall and utilize past episodes during decision-making, supporting adaptive behavior. However, ISAC's episodic memory is limited compared to biological systems, as it primarily captures task-relevant information rather than rich, temporally structured narratives. The episodic memory focuses on practical recall for action selection and problem-solving.

The multi-modal memory characteristic is implemented in ISAC for sub-symbolic representations. The architecture integrates sensory data from multiple modalities, including vision, touch, and proprioception. Perceptual Memory provides a unified representation of the environment, facilitating recognition, reasoning, and decision-making. During task execution, ISAC can access multi-modal perceptions from both current inputs and stored experiences, allowing it to react adaptively to complex scenarios.

ISAC's associative memory is strongly supported through its cognitive control mechanisms. The architecture can form associations between perceptions, actions, and outcomes, enabling context-sensitive reasoning and decision-making. Associative recall enables ISAC to anticipate the outcomes of actions, recognize familiar situations, and select appropriate responses. The architecture can dynamically create and update associations over time, enhancing adaptability.

The introspective capabilities of ISAC are partially implemented. The architecture features a Self-Monitoring System that observes and evaluates its own cognitive processes, monitoring task performance and identifying errors or deviations from expected outcomes. This self-monitoring enables ISAC to adaptively refine its behavior and plans. Introspection in ISAC is functional and task-oriented, focusing on error detection and goal fulfillment rather than deep self-reflection. The architecture does not engage in meta-cognition in the sense of reasoning about its own cognitive processes or exploring alternative strategies.

ISAC's learning capabilities are strong on the sub-symbolic level and limited on the symbolic level. The architecture supports sub-symbolic learning through adaptation and reinforcement mechanisms, enabling it to improve performance over time based on feedback from the environment. This learning is task-specific, focusing on refining skills and behavior for adaptive control. However, ISAC's symbolic learning capabilities are limited, as the architecture lacks mechanisms for rule-based reasoning, symbolic manipulation, or explicit concept formation. Symbolic knowledge is typically pre-defined, reflecting task goals rather than being autonomously generated. This constraint arises from ISAC's focus on real-time control and dynamic adaptation rather than high-level cognitive reasoning.

In summary, ISAC effectively supports active, and associative memory characteristics. Its multi-modality, episodic memory and introspective capabilities are partially implemented, focusing on practical task execution and adaptation.

A.1.6. State, Operator Apply Result (Soar)

The Soar cognitive architecture (Laird, 2019, 2022b; Laird et al., 1987) is a comprehensive, general-purpose framework designed to model human cognition and support intelligent agents in complex environments. It integrates problem-solving, decision-making, and learning capabilities using a unified approach.

Soar's memory system (Derbinsky and Laird, 2009) is active in nature due to its continuous use of long-term knowledge to drive decision-making and problem-solving. The architecture features working memory, long-term semantic memory, procedural memory, and episodic memory. Working memory holds the current state of the problem, including goals, perceptions, and intermediate results. Procedural memory stores rules that trigger actions based on conditions in working memory, facilitating goal-directed behavior. The architecture operates through production cycles where rules match against working memory, generating actions or sub-goals to resolve impasses. This active use of memory enables Soar to dynamically apply knowledge, create new sub-goals, and manage complex tasks efficiently.

Soar's episodic memory characteristic is partially implemented. The architecture includes a dedicated episodic memory component that records snapshots of working memory over time. These snapshots represent states encountered during problem-solving, enabling Soar to recall past experiences. The episodic memory in Soar is designed to support reasoning, learning, and adaptation by providing historical context.

The multi-modal memory characteristic in Soar is limited. The architecture can integrate perceptual inputs into working memory, enabling it to reason about sensory data in a symbolic manner. However, Soar does not natively handle multi-modal sensory integration in the sense of fusing raw sensory inputs from vision, sound, and other modalities into a holistic percept. Instead, perceptual information must be abstracted into symbolic representations before being used by the architecture.

The associative memory property is implemented in Soar through its production rule system. The architecture can create associations between conditions and actions, allowing context-sensitive responses. Associative learning occurs when Soar encounters impasses—situations where no rules apply – and resolves them by creating new productions.

Soar's supports introspective capabilities. The architecture can reason about decisions, evaluate strategies, and refine behavior. Introspection is enabled by representing internal states and decisions symbolically, allowing Soar to understand and manipulate its own reasoning processes.

In summary, Soar fulfills the active, associative, and introspective memory characteristics, with robust symbolic learning capabilities. Its multi-modality and episodic memory are partially implemented, offering useful recall and generalization on symbolic level. But, Soar is one of the most prominent architectures and has therefore been adapted and extended for many different systems. Specific implementations of Soar could therefore be more focused on episodic memory or sub-symbolic representations.

A.1.7. Adaptive Control of Thought – Rational (ACT-R)

The ACT-R cognitive architecture (Anderson et al., 2004; Anderson et al., 1997; Laird, 2022a) is a theory-driven model of human cognition that simulates how humans perceive, think, and act. It is designed to represent the structure and processes of the human mind, emphasizing the integration of symbolic and sub-symbolic mechanisms.

ACT-R's memory system is active due to its dynamic management of information. The architecture comprises declarative memory and procedural memory, along with buffers for sensory and motor processing. Declarative memory stores facts and experiences as chunks, while procedural memory consists of production rules that define actions based on conditions. During cognition, production rules are matched against the contents of the buffers, and the most relevant rule is selected and executed. This cycle enables ACT-R to make decisions, solve problems, and adapt behavior based on current goals and environmental stimuli.

The episodic memory characteristic in ACT-R is partially implemented. It captures past experiences as declarative chunks but lacks a rich, detailed temporal. Recall is context-sensitive, with retrieval influenced by the similarity and recency of stored chunks. However, the system does not autonomously encode or recall events in a continuous or narrative form.

ACT-R's multi-modal memory is implemented through specialized modules that process inputs from different sensory modalities. However, the architecture focuses on symbolic representations rather than raw sensory fusion, and multi-modal integration is task-specific. This approach provides functionality for perceptual and motor tasks but lacks the seamless sensory fusion seen in biological cognition.

The associative memory property is strongly supported in ACT-R. Declarative memory stores information as chunks, which can be retrieved based on associative cues. Retrieval strength depends on factors such as frequency, recency, and context. Production rules in procedural memory also create associations between conditions and actions, enabling adaptive behavior.

ACT-R's introspective capabilities are partially implemented. The architecture can simulate certain aspects of meta-cognition by setting internal goals and reflecting on its own cognitive processes. For example, ACT-R can represent uncertainty, evaluate decisions, and revise strategies. However, introspection is limited to task-driven monitoring and evaluation.

ACT-R's learning capabilities are robust at the symbolic level but limited at the sub-symbolic level. In summary, ACT-R fulfills the active and associative memory characteristics, with strong symbolic learning. Multi-modality, episodic memory and introspection are partially implemented.

A.1.8. Learning Intelligent Decision Agent (LIDA)

The LIDA cognitive architecture (Franklin et al., 2013) is designed to model human cognition by integrating perception, memory, decision-making, and learning in a unified framework. Its architecture consists of multiple interacting components that simulate how humans perceive, understand, and respond to their environment.

LIDA's memory system is inherently active due to its use of cognitive cycles, which continuously drive perception, decision-making, and action selection. During each cycle, information from the environment is processed through sensory memory, perceptual memory, and working memory, with relevant content being broadcast to the Global Workspace (Newell, 1990). This broadcast mechanism mimics conscious awareness, where significant information is globally available to all cognitive processes.

The episodic memory characteristic in LIDA is functionally implemented and central to the architecture's design. LIDA's episodic memory stores detailed records of past events, including perceptual content, actions, and context. These episodes, although not explicitly encoded, are structured as temporally ordered sequences, capturing the flow of experiences over time.

The multi-modal memory characteristic is effectively implemented in LIDA. The architecture processes sensory inputs from multiple modalities, including visual, auditory, and tactile information. Perceptual memory then identifies patterns and creates percepts, which are unified into a coherent scene.

The associative memory property is supported in LIDA's declarative memory. Associations between percepts, actions, and contexts are formed automatically as experiences are encoded. Retrieval is context-sensitive, with cues triggering related episodes or semantic knowledge. This associative recall enables LIDA to apply past knowledge to new situations, facilitating analogical reasoning and decision-making. Associations can be strengthened or weakened over time, reflecting experience and relevance. This dynamic, context-driven associative memory closely resembles human memory capabilities, supporting adaptive behavior in changing environments.

LIDA's introspective capabilities are well-developed. The architecture supports a form of meta-cognition by evaluating cognitive processes during each cognitive cycle. The Global Workspace acts as a conscious spotlight, making specific information globally accessible for decision-making. LIDA can evaluate the importance of information, focus on relevant details, and adjust behavior accordingly. This introspection enables LIDA to monitor its own decision-making and reasoning processes, reflecting on successes and failures.

For learning, the architecture uses procedural memory for skill acquisition and reinforcement learning, allowing adaptive behavior based on past actions and outcomes. LIDA's hybrid approach allows it to handle both structured, symbolic knowledge and continuous, sub-symbolic information, supporting robust adaptation.

A.1.9. A Distributed Adaptive Control Theory (ADAPT)

The ADAPT cognitive architecture (Benjamin et al., 2004) models adaptive, autonomous behavior in dynamic environments by integrating perception, memory, learning, and decision-making. It is inspired by biological systems and emphasizes the role of embodied cognition.

ADAPT's memory system is active through its distributed control structure. The architecture operates based on adaptive control loops that process sensory input, generate motor actions, and adjust behavior based on environmental feedback. These control loops allow ADAPT to continuously interact with its environment, adapting actions to achieve goals.

The episodic memory characteristic in ADAPT is partially implemented. The architecture stores information about interactions with the environment, including sensory inputs, actions, and outcomes. However, this memory is typically represented in a distributed and implicit form rather than as detailed, sequential episodes. ADAPT can use past interactions to guide future behavior, but it lacks the explicit, narrative-based recall of episodic memory.

ADAPT's multi-modal memory is effectively implemented for sub-symbolic information. The architecture integrates inputs from multiple sensory modalities, enabling coordinated responses in complex environments. Different control loops can simultaneously process sensory information, allowing the system to respond adaptively based on multi-modal data.

The associative memory property is strongly supported in ADAPT. The architecture forms associations through sensorimotor learning, where repeated interactions strengthen connections between stimuli, actions, and outcomes.

ADAPT's introspective capabilities are not present. The architecture does not explicitly monitor or evaluate its own cognitive processes.

ADAPT's learning capabilities are primarily sub-symbolic. The architecture leverages adaptive control and reinforcement learning to modify behavior over time. Learning occurs through interactions with the environment, where successful actions are reinforced and ineffective actions are suppressed.

In summary, ADAPT effectively fulfills the active and associative memory characteristics, with strong sub-symbolic learning capabilities. Multi-modality, including symbolic representations, episodic memory and introspection are not or only partially implemented.

A.1.10. Self-Aware Self-Effecting (SASE)

The SASE cognitive architecture (Weng, 2002) is designed to achieve autonomous, adaptive, and intelligent behavior by leveraging self-awareness, self-monitoring, and self-adaptation. It aims to create systems that can introspectively evaluate their own states and processes, enabling flexible and adaptive behavior in dynamic environments.

SASE's memory system is active due to its self-adaptive mechanisms that continuously monitor and modify internal states and behaviors in response to environmental changes. The architecture's self-awareness enables it to recognize the need for adaptation, initiate internal adjustments, and execute appropriate responses.

The episodic memory characteristic in SASE is partially implemented. The architecture can store and recall information about past experiences, including internal states, actions, and outcomes. This enables the system to reason about past events and use them to inform future decisions. However, episodic memory in SASE is not as rich or detailed as in other architectures.

SASE's multi-modal memory is effectively implemented by integrating diverse sensory inputs, enabling coordinated responses in complex, dynamic environments. The architecture handles multiple modalities, processing and combining inputs from various sources. This multi-modal integration enhances perceptual awareness and supports adaptive decision-making. The system's ability to handle and relate different types of sensory data allows it to make informed decisions based on a holistic understanding of the environment. However, the focus remains on goal-oriented processing rather than long-term sensory fusion.

The associative memory property is strongly supported in SASE. The architecture forms associations between sensory inputs, internal states, actions, and outcomes over time. These associations allow SASE to recognize patterns, make predictions, and adapt behavior based on learned experiences.

SASE's introspective capabilities are comprehensively implemented. The architecture continuously monitors its internal states, goals, and performance. It can evaluate the effectiveness of its actions, identify errors, and adjust strategies accordingly. This meta-cognitive functionality enables SASE to reason about its own cognitive processes.

SASE's learning capabilities are robust at both symbolic and sub-symbolic levels. The architecture employs reinforcement learning, associative learning as well as symbolic learning.

A.1.11. Cognitive Robot Abstract Machine (CRAM)

The CRAM (Beetz et al., 2010) cognitive architecture is designed to enable autonomous robots to perform complex tasks in unstructured environments. It emphasizes high-level reasoning, planning, and the dynamic adaptation of actions based on sensory input. Memory in CRAM is a central component that facilitates the storage, retrieval, and utilization of information during task execution.

CRAM's memory system can be considered active due to its integration with plan-based control mechanisms, which enable continuous monitoring, reasoning, and adaptation during task execution. The architecture is able to encode complex plans that can be dynamically adapted based on sensory input and internal state evaluations. This allows CRAM to respond proactively to changes in the environment and handle unexpected events. The memory system does not merely store information passively but actively contributes to decision-making processes by accessing and manipulating stored information during execution.

Episodes in CRAM' episodic memory consist of temporally structured experiences, capturing sensory inputs, executed actions, and observed effects, enriched by a narrative (Beetz et al., 2018). These episodes can be recalled to analyze past experiences, evaluate the success of actions, and adjust future plans. While CRAM can store detailed task-related information, it lacks the depth of context, personal significance, and emotional valence associated with human episodic memory. The representation of episodes in CRAM is limited to task-relevant information, and the architecture does not naturally capture subjective aspects of experiences. Also, continuous update, representation learning and gradually forgetting in episodic memory are not implemented.

CRAM effectively handles multi-modal memory by integrating various sensory modalities, including visual, tactile, and proprioceptive data. These sensory streams are associated with specific actions and plans, enabling the robot to understand environmental contexts and respond appropriately. The system forms comprehensive episodes by correlating multiple sensory inputs, allowing for robust perception and decision-making. However, the degree of sensory fusion is restricted by hardware limitations and the computational cost of processing rich sensory data in real time. Multi-modal integration is pragmatic rather than deeply semantic, focusing on task-relevant information.

Associative memory in CRAM is realized through the semantic reasoning capabilities of KnowRob (Beetz et al., 2018; Tenorth and Beetz, 2013). The system stores relationships between objects, actions, and contexts, allowing the robot to perform reasoning based on associations. For example, the architecture can infer that a graspable object can be picked up or that a door marked as open is passable. These associations enable flexible reasoning and support adaptive behavior in dynamic environments. However, the associations are limited to predefined rules and relationships. CRAM does not autonomously generate novel or abstract associations beyond its knowledge base, reflecting the constraints of rule-based semantic reasoning.

CRAM's memory system exhibits introspective qualities by monitoring ongoing plans, goals, and predicted states. This self-monitoring capability allows the robot to assess its performance, detect failures, and replan if necessary. Introspection in CRAM is achieved through the evaluation of internal states during task execution, providing a degree of self-awareness. Nevertheless, this introspective ability is restricted to action and plan monitoring. The system lacks deeper meta-cognitive capabilities, such as reflective thought or reasoning about its own cognitive processes (Nolte et al., 2025). Introspection in CRAM is functional but not self-reflective.

The architecture's learning capabilities are robust on the symbolic level but limited on the sub-symbolic level. CRAM supports symbolic learning by updating its knowledge base based on episodic memories and refining its plans over time. This symbolic adaptation enables the robot to perform better in repeated scenarios. However, sub-symbolic learning, such as neural network-based pattern recognition, is not a primary focus.

In summary, CRAM effectively supports active, multi-modal, and associative memory characteristics. Episodic and introspective memory are implemented but only partially, reflecting limitations in capturing the depth and richness of human cognition. The architecture excels in symbolic learning but remains limited in sub-symbolic learning and deep meta-cognitive capabilities. These constraints are inherent to the focus on practical task execution and the difficulty of replicating complex cognitive processes.

A.2. Detailed Description of ARON Datatypes used in ArmarX

In ArmarX we use a wide variety of different datatypes and modalities. These do not have to be redefined in every package or library, but are part of the framework. The most important data types used in our software are listed in Table A.1. Some of the types are motivated by prominent frameworks in robotics such as *Simox* (Vahrenkamp et al., 2012), *Eigen3* (Guennebaud, Jacob, et al., 2010), *OpenCV* (*The OpenCV Reference Manual* 2014) or *PCL* (Rusu and Cousins, 2011) an can automatically be converted. In our recent work, these type automated conversions have also been implemented for ROS-2 (Macenski et al., 2022).

Type Name	Description
`Vector2f`	A 2D vector composed of two 32-bit floating point numbers (x, y). Commonly used to represent positions or directions in a 2D plane.
`Vector3f` (aka `Position`)	A 3D vector with float32 components x, y, and z. Typically used to represent spatial positions or displacements in 3D space.
`Vector2d`, `Vector3d`	Double-precision variants of the above vectors, using float64 for increased accuracy in numerical computations.
`Matrix2f`	A 2x2 matrix with float32 entries, useful for 2D linear transformations such as rotation, scaling, or shearing.
`Matrix3f`, `Matrix4f` (aka `Pose`)	3x3 and 4x4 matrices with float32 values. Used for 3D transformations, combining rotation, translation, and optionally scaling or perspective transformations.
`Matrix2d`, `Matrix3d`, `Matrix4d`	Higher precision equivalents of the above matrices using float64.
`Quaternionf` (aka `Orientation`), `Quaterniond`	A quaternion representing 3D orientation, composed of four float components: x, y, z, and w. Float32 and float64 versions are provided. Automatically convertible to and from Eigen or tf types.
`FrameID`	Describes a reference frame within a multi-agent system using two strings: the frame name and the associated agent name.
`FramedPosition`, `FramedPose`, `FramedOrientation`	Combines a spatial element (position, pose, or orientation) with a FrameID to express its reference frame. Enables consistent transformations between coordinate systems.

`LaserScanHeader`	Metadata accompanying a laser scan, including the emitting agent, the reference frame, and the timestamp of the measurement.
`LaserScan`	Full laser scan message including a header, a global timestamp, and raw scan data as an NDArray.
`ColorRGBA`, `ColorHSV`	Represent colors either as RGBA (red, green, blue, alpha) or HSV (hue, saturation, value) components, each stored as 32-bit integers.
`ImageRGB` (aka `Image`)	RGB image data stored in an NDArray with shape $\{w, h, 3, 1\}$. Intended for color images; supports conversion to OpenCV for processing.
`ImageDepth`	A depth image with float32 pixel values. Stored as an NDArray of shape $\{w, h, 1, 4\}$, indicating width, height, single channel, and 4 bytes per value. Conversion to OpenCV is supported.
`PointXYZ, PointXYZRGB, PointXYZRGBA, ...`	Represents 3D points with optional color information. Variants include RGB and RGBA channels. Used in point cloud representations and compatible with PCL.
`PointCloudXYZ, PointCloudXYZRGB, ...`	Collections of the above point types to represent complete 3D point clouds. Can be seamlessly converted to PCL data structures for processing and visualization.
`ClockType`	Enumeration indicating the type of time source used: realtime (wall clock), monotonic (non-decreasing), virtual (simulated), or unknown.
`Duration`	Time duration stored as a 64-bit integer representing microseconds. Used across the framework for temporal measurements.
`Frequency`	Represents a frequency value based on a Duration. Enables intuitive definition of rates and periodic processes.
`DateTime`	Comprehensive timestamp format including time since epoch (Duration), associated ClockType, and originating hostname. Converts to Simox for compatibility.

`TaskSpaceElement`	A single point in a task space trajectory. Contains a timestamp (float32) and a Pose, describing the robot's spatial configuration.
`JointSpaceElement`	Similar to TaskSpaceElement, but instead of a pose, contains a list of joint angles (float32) for the robot. Used for joint-level trajectory planning.
`TaskSpaceTrajectory`, `JointSpaceTrajectory`	Sequences of task or joint space elements forming full trajectories. Used in motion planning and execution.
`BoundingBox`	An axis-aligned 3D box defined by its center and extents, both as Vector3f. Useful for collision checking and object fitting.
`OrientedBox`	Similar to BoundingBox but includes an orientation, allowing the box to be arbitrarily rotated in space.
`ObjectClass`	Defines static knowledge about an object type: includes object ID, names, parent class, mesh, bounding box, features, and joints (for articulated models). Typically sourced from PK.
`ObjectInstance`	A specific realization of an object class. Includes the object's pose (global or framed), instance ID, class reference, and additional metadata such as the creator.
`KnownGrasp`	A predefined grasp associated with a known object, including grasp ID, pose relative to the object, handedness, optional pre-grasp pose, and a quality metric.
`KnownGraspSet`	A collection of KnownGrasp entries that describe all available grasps for a specific object.
`GraspCandidate`	A dynamically generated grasp proposal. Includes generation context, reference object, quality estimates, and optional planning metadata.
`JointInfo`	Represents real-time feedback and status from a single robot joint, including position, velocity, torque, effort, and thermal state.
`PlatformInfo`	Provides movement-related information of the robot base, such as linear and angular velocity or acceleration.

`Proprioception`	Aggregates all JointInfo and PlatformInfo to describe the robot's internal sensorimotor state.
`RobotDescription`	Structural and semantic description of a robot. Includes joints, names, and hierarchy. Often statically defined in PK.
`RobotState`	Captures the full robot state at a particular point in time: global pose, proprioception, timestamp, and RobotDescription reference.
`SkillDescription`	Describes a robotic skill, including its symbolic name, parameters, execution proxy, result type, timeout, and event expectations.
`SkillParameterization`	A specific assignment of parameters used to configure and invoke a SkillDescription.
`SkillExecutionRequest`	Stores execution metadata such as which agent requested a skill, when, and with what parameters.
`SkillExecutionResult`	Logs the results of a skill execution, including success state, outputs, and intermediate statuses from different execution stages.
`Human`	Static information about a known human, such as the name, face features or profile information.
`HumanPose`	A description of the human at a particular point in time. The human is composed of keypoints, their position in local and global coordinates and a reference to the recognition component. An id, which is assigned to the human if the human was recognized, is optional.
`Affordance`	Semantic description of an object's interactive property. Includes a FramedPose, textual description, and label such as graspable, navigable, etc.
`SymbolicSceneDescription`	Symbolic, high-level representation of a scene, constructed by combining object, robot, and skill memory using knowledge about affordances and relations.

`PackagePath`	Identifies a resource location within a software package. Contains the package name and a relative path to the resource. Used for modular software referencing in ArmarX.

Table A.1.: A collection of compound datatypes and type aliases defined within the ARON knowledge representation framework and used throughout the ArmarX robot software framework. ARON is designed to be highly extensible, and its type system is regularly expanded to accommodate new application needs. To ensure a natural and intuitive experience for developers, many of the defined types draw inspiration from established libraries such as *Simox*, *Eigen3*, *OpenCV*, and *PCL*. Types optimized for efficient network transmission are internally represented as `NDArray` structures. This table provides an overview of commonly used general-purpose types in ArmarX; it excludes domain-specific or transient types used for intermediate computations (e. g., process results) or specialized functionality (e. g., natural language understanding, navigation, verbalization, etc.).

A.3. Detailed Description of Memory Servers used in ArmarX

Memory Server Name	Description
`RobotStateMemory`	Stores comprehensive information about all known robots and their current states. This includes robot descriptions (e.g., structure and configuration), proprioception data (joint positions, velocities, forces), localization information, and motor command values. As the core memory for representing the robot itself, it is widely used across modules that require an up-to-date model of the robot for tasks such as control, simulation, planning, and visualization. Its contents are rendered in the *ArViz* 3D visualization tool for purposes such as collision checking, path planning, and state inspection.

`ObjectMemory`	Contains symbolic and geometric information about objects in the robot's environment. This includes static object classes, dynamic object instances, and models for known, similar, or unknown objects. It also stores the environment model and semantic annotations. As a frequently accessed memory, it is typically located on the machine running high-level cognitive modules. Object information is visualized in *ArViz* for reasoning, manipulation planning, and scene interpretation.
`VisionMemory`	Captures and stores perceptual data generated by the robot's vision system, including RGB images, depth images, laser scans, and other sensor data. This memory enables asynchronous access to raw sensor information and supports vision-based algorithms for detection, tracking, segmentation, and reconstruction. It is usually hosted on the same machine as the sensors to reduce network latency and bandwidth usage.
`SkillsMemory`	Holds symbolic and executable knowledge about robotic skills. This includes skill descriptions, parameterizations, logs of past executions, and result data. It serves as the central repository for managing the lifecycle of robotic capabilities—from definition to execution and monitoring.
`GraspMemory`	Stores grasp-related knowledge for objects, including known grasps (predefined, high-quality grasps), grasp candidates (generated on the fly), and associated metadata such as grasp quality, pregrasp poses, and handedness. It facilitates grasp planning, selection, and evaluation, and its contents are visualized in *ArViz* for debugging and applicability analysis.
`SystemStateMemory`	A low-level memory that records system resource usage and runtime statistics, such as CPU load, memory usage, and active processes. This information can be leveraged to monitor the system's health, enable adaptive load balancing, or support debugging and diagnostics.

`HumanMemory`	Manages symbolic and instance-level information about detected humans in the scene. This includes recognition results, identity assignments, and tracked poses. It enables human-aware interaction and is rendered in *ArViz* to provide situational awareness during human-robot collaboration.
`IndexMemory`	Serves as an optimization layer that stores precomputed indices or lookup tables for efficient access to other memory entries. This can significantly speed up query resolution in large-scale knowledge bases by avoiding repeated full scans.
`MotionPrimitiveMemory`	Stores motion primitive (MP) knowledge, including symbolic representations and parametrizations of reusable low-level motions. These primitives are typically used in motion planning, manipulation, and complex skill execution.
`NavigationMemory`	Contains all symbolic and metric information related to navigation. This includes known locations, room and building layouts, navigation graphs, and costmaps. It supports motion planning and localization and provides a spatial understanding of the environment. All navigational data is visualized in *ArViz* for inspection and planning purposes.
`SymbolicSceneMemory`	Maintains a symbolic abstraction of the current scene, integrating data from various sources such as object memory, robot state, and skill execution. It includes high-level representations such as object affordances, location types, and spatial relations, and is visualized in *ArViz* for semantic interpretation and planning.
`AudioMemory`	Stores audio input data as temporal chunks, which can be used for sound localization, audio-based interaction, or offline analysis. It forms the basis for voice recognition pipelines and auditory perception.
`ControlMemory`	Logs control-related data such as controller parameterizations, setpoints, and actual control values. This memory supports control tuning, diagnostics, and retrospective analysis of controller performance.

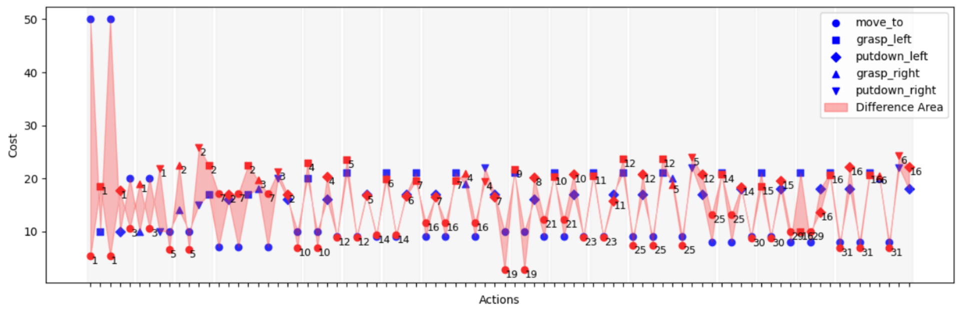
`SpeechMemory`	Captures all verbal communication knowledge, including both recognized speech inputs and robot-generated verbal outputs. It supports natural language interaction and logging of dialogue history.
`ManipulationMemory`	Stores specialized knowledge relevant to robotic manipulation. This includes strategies for grasping, placing, pushing, and tool use, as well as logs of past manipulation attempts and success metrics.
`PlanningMemory`	Augments skill-based representations with symbolic planning information, typically in PDDL format. It is used for generating high-level action plans, enabling goal-oriented behavior and autonomous task execution.
`ViewSelectionMemory`	Manages view targets and camera strategies for perception tasks. It enables reasoning about which viewpoints are optimal for observing specific objects, areas, or interactions. This memory is critical in active perception scenarios and view planning.
`DemonstrationMemory`	Stores demonstrations of actions performed by humans, either recorded through kinesthetic teaching, teleoperation, or visual observation. These demonstrations are used for imitation learning, trajectory generalization, and skill acquisition.

Table A.2.: This table presents a selected subset of the memory servers available within the ArmarX framework. While the listed servers represent those most commonly used in day-to-day development and deployment, numerous additional memory servers exist to support specific research domains. These specialized servers serve to organize and centralize research outcomes and often provide extended functionalities such as unified data export. Owing to the modular and extensible architecture of the memory system, new servers can be integrated or removed with minimal effort, allowing the framework to adapt efficiently to evolving requirements.

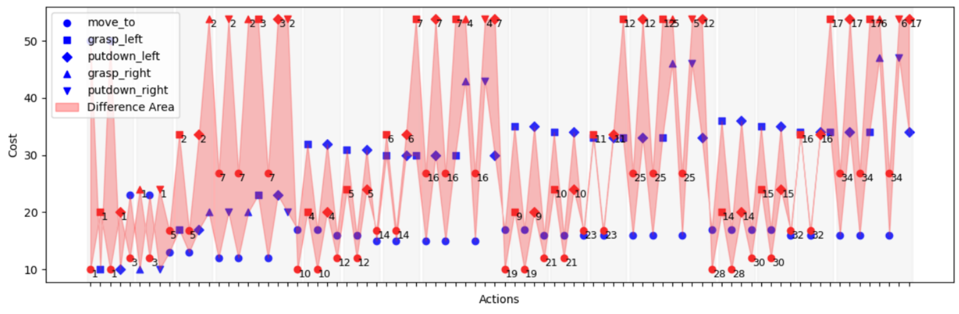
A.4. Cost Adaptions during Household Task with Anomaly Detection based on Z-Score

In order to identify outliers in the experience data of action executions, the z-score for anomaly detection was introduced. In the following, the results of the household task in Section 7.3.4

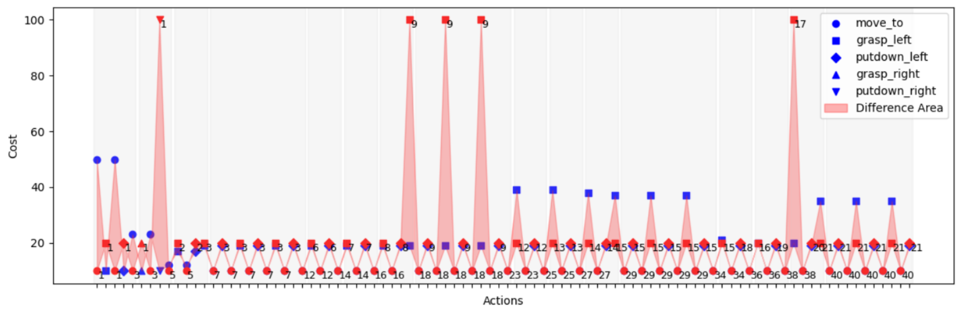
are presented, whereby the input data is also checked for anomalies and corrected. This enables the system to ignore one-off high costs and not adjust the model in this case.



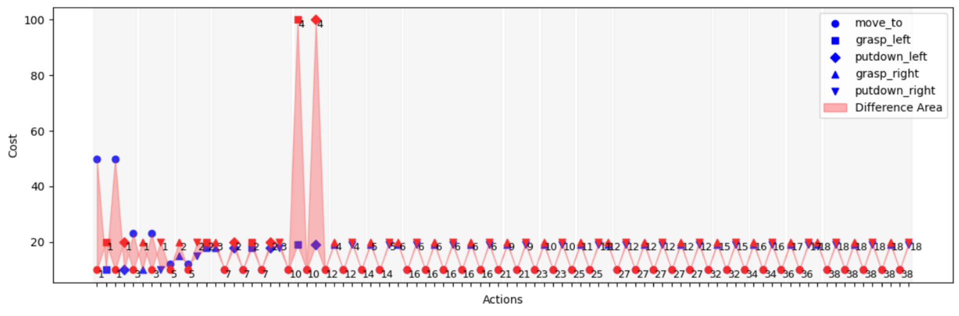
(a)



(b)

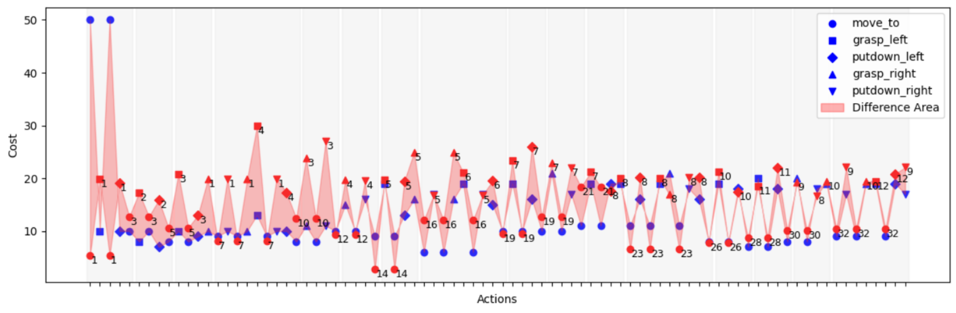


(c)

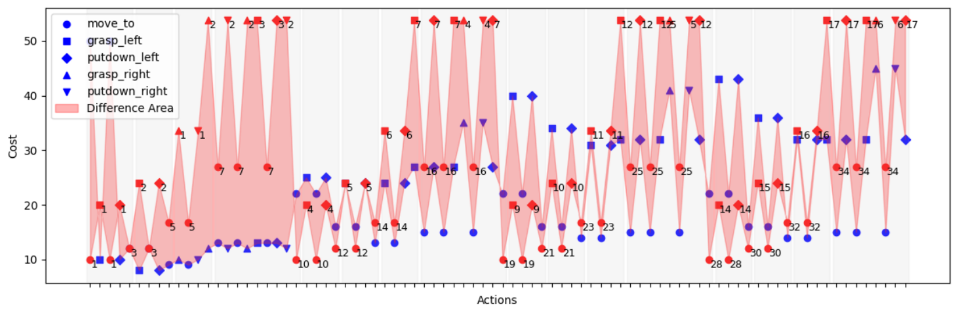


(d)

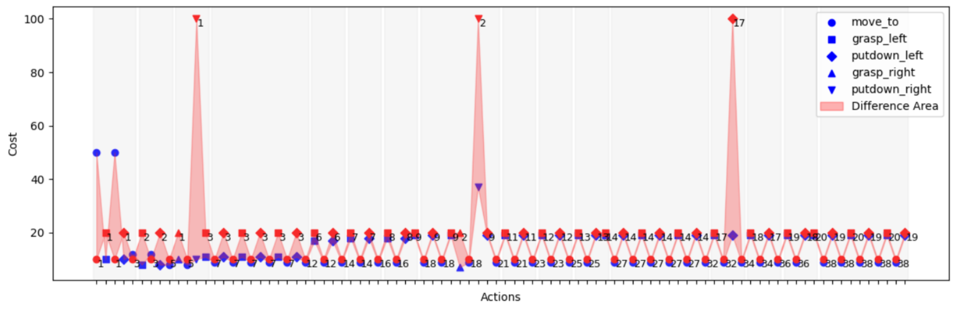
Figure A.1.: Comparison of real and estimated cost using z-score to filter outliers and average forecasting.



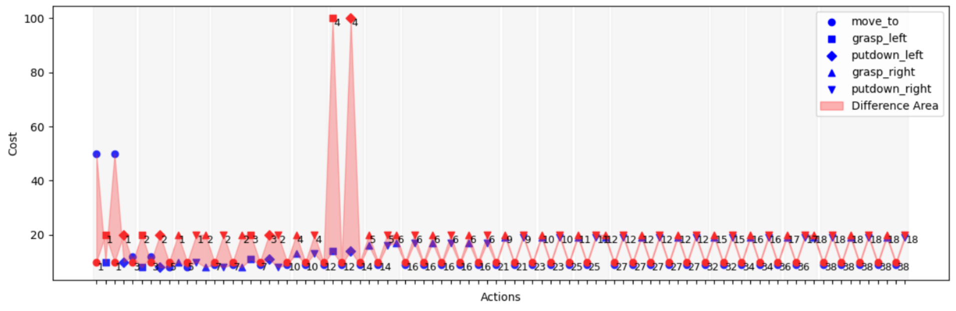
(a)



(b)

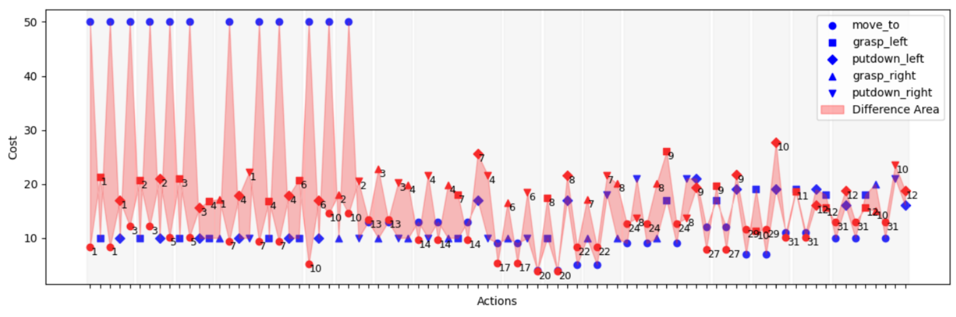


(c)

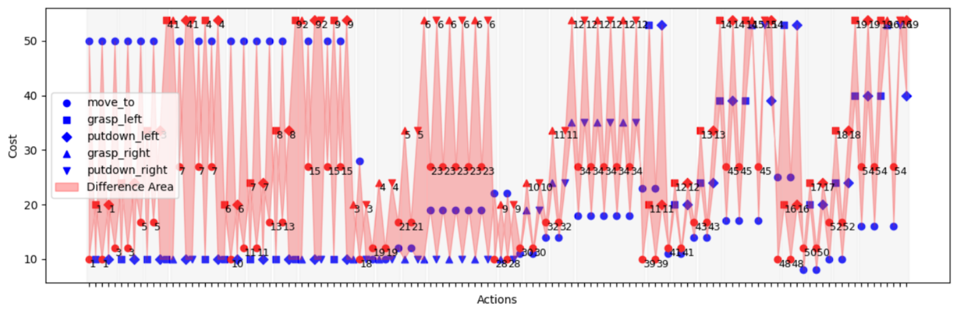


(d)

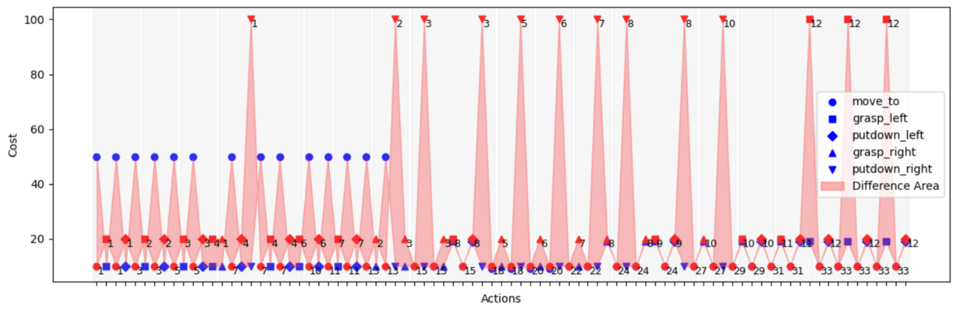
Figure A.2.: Comparison of real and estimated cost using z-score to filter outliers and exponential smoothing.



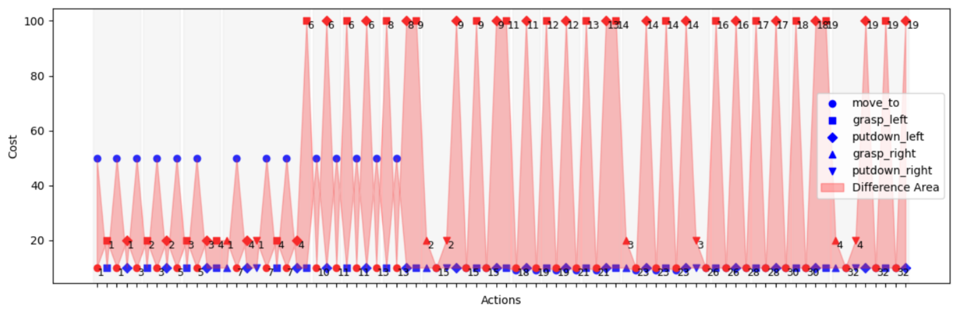
(a)



(b)



(c)



(d)

Figure A.3.: Comparison of real and estimated cost using z-score to filter outliers and STL.

List of Figures

List of Tables

List of Algorithms

Acronyms

20BN 20 Billion Something-Something 84, 85, 87–90, 95, 96, 172

A* A-star 118, 119
ACT-R Adaptive Control of Thought – Rational 33, 154, 155
ADAPT A Distributed Adaptive Control Theory 33, 156, 157
ADL Argument-dependent Lookup 56
AE Auto-encoder 66, 69, 71, 91
AMY Amygdala 18
ARON ArmarX Object Notation 49–62, 70, 71, 75, 82, 164, 171, 173

BDNF Brain-derived Neurotrophic Factor 21
BG Basal Ganglia 18, 20
BO Business Object 58, 82, 83

CLARION Connectionist Learning with Adaptive Rule Induction On-line 22, 23
CRAM Cognitive Robot Abstract Machine 27, 33, 158, 159
CREB Cyclic AMP Response Element-binding Protein 21

deep EM Deep Episodic Memory 8, 33, 66, 69, 70, 72, 84, 85, 93, 106, 108, 150, 151, 171
DTG Domain Transition Graph 124, 125
DTO Data Transfer Object 53, 81, 82

EM Episodic Memory 5, 12, 16, 17, 21–26, 30, 59, 69, 70, 79, 101, 104, 106, 107, 115, 118, 177
EMV EM-based Verbalization 101, 104–114
EPIC Executive-Process/Interactive Control 23, 33, 147, 148

FFT Fast Fourier Transformation 131
FIFO First-in-First-out 66

GUI Graphical User Interface 72, 73, 171

HCEA Context-enhanced Additive Heuristic 122
HPC Hippocampus 13, 14, 18–21, 29
HRI Human-Robot Interaction 101, 102

IDF Introspectable Data Format 49
ISAC Intelligent Soft Arm Control 26, 33, 152, 153

JSON JavaScript Object Notation 49, 55, 58, 68

LIDA Learning Intelligent Decision Agent 27, 33, 155, 156
LLM Large Language Model 101, 102, 114, 137, 138
LOESS Locally Estimated Scatter-plot Smoothing 127, 131, 179
LTM Long-term Memory 7, 8, 12–16, 19–23, 28, 29, 31, 59, 60, 65, 66, 68–71, 74, 78, 95, 140, 143, 171

MAE Mean Average Error 68
MD Mediodorsal Thalamus 20
MDB Multilevel Darwinist Brain 33, 151, 152
MNS Memory Name System 63
MSE Mean Squared Error 68, 72, 95, 107
MTL Medial Temporal Lobe 20

NEEMs Narrative-Enabled Episodic Memories 28
NLG Natural Language Generation 107, 113

OAC Object-Action Complex 2, 74

P2P Peer-to-Peer 77, 83, 96
PDDL Planning Domain Definition Language 118, 119, 128, 137, 167
PFC Prefrontal Cortex 18–21, 29
PK Prior Knowledge 7, 60, 162, 163
PM Procedural Mmemory 12
PS Publish-Subscribe 77, 83, 96
PSNR Peak-Signal-to-Nnoise-Ratio 85–91, 97, 172

QA Question-Answer 103, 104, 110, 111, 141

RER Robot Episode Recording 103
ROS Robot Operating System 58, 81
RPC Remote Procedure Call 58, 65

SASE Self-Aware Self-Effecting 25, 33, 157, 158
SES Sensory Ego-sphere 26, 27

SM Semantic Memory 12
Soar State, Operator Apply Result 22–24, 26, 33, 153, 154
SSIM Structural Similarity Index Metric 85
STL Seasonal Trend Decomosition using LOESS 131, 134, 170, 172

TFD Temporal Fast Downward 121–123, 172
THAL Thalamus 20
TOD Task-oriented Dialogue 101
TSFD Temporal Stressing Fast Downward 121–123, 141, 172
TTS Text-to-Speech 104

UML Unified Modeling Language 50, 52

VAE Variational Auto-encoder 25, 69
VQA Video Question Answering 101

WAE Wasserstein Auto-encoder 69
WM Working Memory 7, 12, 19, 20, 23–26, 28–31, 59–61, 64–66, 68, 70, 80, 95

XML Extensible Markup Language 53, 55–57, 62

ZeroC Ice ZeroC Internet Communication Engine 58, 81, 83

Bibliography

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. (2016). "TensorFlow: a system for large-scale machine learning". In: *USENIX symposium on operating systems design and implementation (OSDI)*, pp. 265–283 (cit. on p. 69).

Ageno, S. and K. Iramina (2024). "Analysis of time-varying brain network activity using functional connectivity and graph theory during memory retrieval". In: *Advanced Biomedical Engineering* 13, pp. 123–133 (cit. on p. 29).

Aine, S., S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev (2015). "Multi-heuristic A*". In: *The International Journal of Robotics Research* 35, pp. 224–243 (cit. on p. 118).

Allen, J. F. (1983). "Maintaining knowledge about temporal intervals". In: *Commununications of the ACM* 26, pp. 832–843 (cit. on p. 119).

Anderson, J., D. Bothell, M. Byrne, S. Douglass, C. Lebiere, and Y. Qin (2004). "An integrated theory of the mind". In: *Psychological Review* 111, pp. 1036–1060 (cit. on pp. 30, 31, 33, 154).

Anderson, J. R., M. Matessa, and C. Lebiere (1997). "ACT-R: a theory of higher level cognition and its relation to visual attention". In: *Human—Computer Interaction* 12.4, pp. 439–462 (cit. on p. 154).

Anderson, M. C. and S. Hanslmayr (2014). "Neural mechanisms of motivated forgetting". In: *Trends in Cognitive Sciences* 18, pp. 279–292 (cit. on p. 29).

Andrade, C. (2021). "Z scores, standard scores, and composite test scores explained". In: *Indian Journal of Psychological Medicine* 43.6, pp. 555–557 (cit. on p. 126).

Al-Ani, M. and F. Awad (2013). "The JPEG image compression algorithm". In: *International Journal of Advances in Engineering & Technology* 6, pp. 1055–1062 (cit. on p. 66).

Asfour, T., K. Regenstein, P. Azad, J. Schröder, N. Vahrenkamp, and R. Dillmann (2006). "ARMAR-III: an integrated humanoid platform for sensory-motor control". In: *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*. Genova, Italy, pp. 169–175 (cit. on p. 38).

Asfour, T., M. Wächter, L. Kaul, S. Rader, P. Weiner, S. Ottenhaus, R. Grimm, Y. Zhou, M. Grotz, and F. Paus (2019). "ARMAR-6: a high-performance humanoid for human-robot collaboration in real world scenarios". In: *IEEE Robotics & Automation Magazine* 26.4, pp. 108–121 (cit. on p. 36).

Atkinson, R. C. and R. M. Shiffrin (1968). "Human memory: a proposed system and its control processes." In: *The Psychology of Learning and Motivation: Advances in Research and Theory* 2, pp. 89–105 (cit. on pp. 14, 59).

Auguste, A., N. Fourcaud-Trocmé, D. Meunier, A. Gros, S. Garcia, B. Messaoudi, M. Thévenet, N. Ravel, and A. Veyrac (2023). "Distinct brain networks for remote episodic memory depending on content and emotional value". In: *Progress in Neurobiology* 223 (cit. on p. 17).

Baddeley, A. (2006). "Chapter 1 - working memory: an overview". In: *Working Memory and Education*. Educational Psychology. Academic Press, pp. 1–31 (cit. on p. 28).

Baddeley, A. D. and G. Hitch (1974). "Working memory". In: *Psychology of Learning and Motivation*. Vol. 8. Academic Press, pp. 47–89 (cit. on pp. 15, 59).

Bahdanau, D., K. Cho, and Y. Bengio (2015). "Neural machine translation by jointly learning to align and translate". In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 101, 107).

Bamler, R. and S. Mandt (2017). "Dynamic word embeddings". In: *International Conference on Machine Learning (ICML)*. PMLR, pp. 380–389 (cit. on p. 71).

Barak, O., M. Tsodyks, and R. Romo (2010). "Neuronal population coding of parametric working memory". In: *Journal of Neuroscience* 30, pp. 9424–9430 (cit. on p. 20).

Bärmann, L., C. DeChant, J. Plewnia, F. Peller-Konrad, D. Bauer, T. Asfour, and A. Waibel (2024a). *Episodic memory verbalization using hierarchical representations of life-long robot experience* (cit. on p. 114).

Bärmann, L., R. Kartmann, F. Peller-Konrad, J. Niehues, A. Waibel, and T. Asfour (2024b). "Incremental learning of humanoid robot behavior from natural interaction and large language models". In: *Frontiers in Robotics and AI* 11, pp. 1–14 (cit. on pp. 25, 47, 80, 137).

Bärmann, L., F. Peller-Konrad, S. Constantin, T. Asfour, and A. Waibel (2021). "Deep episodic memory for verbalization of robot experience". In: *IEEE Robotics and Automation Letters (RA-L)* 6.3, pp. 5808–5815 (cit. on pp. 33, 39, 79, 99, 104, 110–113, 141, 150).

Bayreuther, S., F. Jacob, M. Grotz, R. Kartmann, F. Peller-Konrad, F. Paus, H. Hartenstein, and T. Asfour (2022). "BlueSky: combining task planning and activity-centric access control for assistive humanoid robots". In: *Symposium on Access Control Models and Technologies (SACMAT)*, pp. 185–194 (cit. on p. 115).

Beetz, M. (2024). *THINK REACTOR – Cognitive Robotics – mit Prof. Michael Beetz (Uni Bremen)*. https://www.youtube.com/watch?v=VZ16wwfl9UQ, last accessed 31.05.2025 (cit. on p. 5).

Beetz, M., D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoğlu, and G. Bartels (2018). "Know Rob 2.0 — a 2nd generation knowledge processing framework for cognition-enabled robotic agents". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 512–519 (cit. on pp. 27, 39, 158, 159).

Beetz, M., L. Mösenlechner, and M. Tenorth (2010). “CRAM — a cognitive robot abstract machine for everyday manipulation in human environments”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, pp. 1012–1017 (cit. on pp. 27, 31, 33, 158).

Bekinschtein, P., M. Cammarota, L. M. Igaz, L. R. Bevilaqua, I. Izquierdo, and J. H. Medina (2007). “Persistence of long-term memory storage requires a late protein synthesis- and bdnf- dependent phase in the hippocampus”. In: *Neuron* 53, pp. 261–277 (cit. on p. 21).

Bellas, F., R. J. Duro, A. Faina, and D. Souto (2010). “Multilevel darwinist brain (MDB): artificial evolution in a cognitive architecture for real robots”. In: *IEEE Transactions on Autonomous Mental Development* 2, pp. 340–354 (cit. on pp. 33, 151).

Benjamin, D. P., D. M. Lyons, and D. W. Lonsdale (2004). “ADAPT: a cognitive architecture for robotics”. In: *International Conference on Cognitive Modeling (ICCM)*, pp. 337–338 (cit. on pp. 33, 156).

Bhardwaj, M., B. Sundaralingam, A. Mousavian, N. D. Ratliff, D. Fox, F. Ramos, and B. Boots (2022). “STORM: an integrated framework for fast joint-space model-predictive control for reactive manipulation”. In: *International Conference on Robot Learning (CoRL)*, pp. 750–759 (cit. on p. 83).

Birr, T., C. Pohl, A. Younes, and T. Asfour (2024). “AutoGPT+P: affordance-based task planning with large language models”. In: *Robotics: Science and Systems (RSS).* Vol. 2402.10778. Delft, Netherlands (cit. on pp. 102, 137, 138).

Bisby, J., N. Burgess, and C. R. Brewin (2020). “Reduced memory coherence for negative events and its relationship to posttraumatic stress disorder”. In: *Current Directions in Psychological Science* 29, pp. 267–272 (cit. on p. 14).

Bonsall, M. B. and E. A. Holmes (2023). “Temporal dynamics of trauma memory persistence”. In: *Journal of the Royal Society Interface* 20 (cit. on p. 14).

Boran, E., P. Hilfiker, L. Stieglitz, T. Grünwald, J. Sarnthein, and P. Klaver (2020). “Neuronal firing in the medial temporal lobe reflects human working memory workload, performance and capacity”. In: (cit. on p. 20).

Bremner, P. and U. Leonards (2016). “Iconic gestures for robot avatars, recognition and integration with speech”. In: *Frontiers in Psychology* 7 (cit. on p. 43).

Brightwell, J. J., C. A. Smith, R. A. Countryman, R. L. Neve, and P. J. Colombo (2005). “Hippocampal overexpression of mutant creb blocks long-term, but not short-term memory for a socially transmitted food preference”. In: *Learning & Memory* 12, pp. 12–17 (cit. on p. 21).

Brightwell, J. J., C. A. Smith, R. L. Neve, and P. J. Colombo (2007). “Long-term memory for place learning is facilitated by expression of camp response element-binding protein in the dorsal hippocampus”. In: *Learning & Memory* 14, pp. 195–199 (cit. on p. 21).

Bui, H.-D., Q. Dang, and N. Chong (2019). “Robot social emotional development through memory retrieval”. In: pp. 46–51 (cit. on pp. 30, 46).

Buoncompagni, L. and F. Mastrogiovanni (2019). “A framework inspired by cognitive memory to learn planning domains from demonstrations”. In: *AIRO@ AI*IA*, pp. 13–18 (cit. on p. 78).

Cao, Z., G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh (2018). “OpenPose: realtime multi-person 2D pose estimation using part affinity fields”. In: *arXiv preprint* (cit. on p. 105).

Chalvatzaki, G., A. Younes, D. Nandha, A. T. Le, L. F. R. Ribeiro, and I. Gurevych (2023). “Learning to reason over scene graphs: a case study of finetuning GPT-2 into a robot language model for grounded task planning”. In: *Frontiers in Robotics and AI* 10 (cit. on p. 102).

Chen, X., J. Xu, and B. Xu (2019). “A working memory model for task-oriented dialog response generation”. In: *Annual Meeting of the Assoc. for Comput. Linguistics.* Annual Meeting of the Assoc. for Comput. Linguistics, pp. 2687–2693 (cit. on p. 101).

Cheng, H. K. and A. G. Schwing (2022). “XMem: long-term video object segmentation with an atkinson-shiffrin memory model”. In: *arXiv preprint* (cit. on p. 15).

Chodorow, K. (2013). *MongoDB: the definitive guide.* O’Reilly Media, Inc. (cit. on p. 65).

Chu, H.-P., K. Devendra, and D. K.-L. Chan (2007). “Severe intrauterine hemolysis due to anti-cw”. In: *American Journal of Perinatology* (cit. on p. 13).

Cleveland, R. B. (1990). “STL: a seasonal-trend decomposition procedure based on loess”. In: (cit. on p. 127).

Cohen, M. X. (2011). “Hippocampal-prefrontal connectivity predicts midfrontal oscillations and long-term memory performance”. In: *Current Biology* 21, pp. 1900–1905 (cit. on p. 21).

Coon, D. and J. O. Mitterer (2018). *Introduction to Psychology: Gateways to Mind and Behavior.* Cengage Learning (cit. on p. 17).

Cowell, R. A., M. D. Barense, and P. Sadil (2019). “A roadmap for understanding memory: decomposing cognitive processes into operations and representations”. In: *Eneuro* 6 (cit. on p. 25).

Craik, F. I. M., R. Govoni, M. Naveh-Benjamin, and N. D. Anderson (1996). “The effects of divided attention on encoding and retrieval processes in human memory.” In: *Journal of Experimental Psychology* 125, pp. 159–180 (cit. on p. 11).

Cushing, W., S. Kambhampati, Mausam, and D. S. Weld (2007). “When is temporal planning really temporal?” In: *International Joint Conference on Artifical Intelligence*, pp. 1852–1859 (cit. on pp. 3, 120).

Deal, J. A., J. Betz, K. Yaffe, T. B. Harris, E. Purchase-Helzner, S. Satterfield, S. Pratt, N. Govil, E. M. Simonsick, and F. R. Lin (2016). “Hearing impairment and incident dementia and cognitive decline in older adults: the health abc study”. In: *The Journals of Gerontology Series A: Biological Sciences and Medical Sciences*, glw069 (cit. on p. 14).

Derbinsky, N. and J. E. Laird (2009). “Efficiently implementing episodic memory”. In: *International Conference on Case-Based Reasoning (ICCBR).* Springer, pp. 403–417 (cit. on pp. 78, 79, 153).

Derbinsky, N. and J. E. Laird (2013). “Effective and efficient forgetting of learned knowledge in Soar’s working and procedural memories”. In: *Cognitive Systems Research* 24, pp. 104–113 (cit. on p. 78).

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019). “BERT: pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint* (cit. on p. 5).

Draschkow, D., M. Kallmayer, and A. C. Nobre (2021). “When natural behavior engages working memory”. In: *Current Biology* 31, 869–874.e5 (cit. on p. 19).

Duff, S. (2000). “Whats working in working memory: a role for the central executive”. In: *Scandinavian Journal of Psychology* 41, pp. 9–16 (cit. on p. 16).

Duro, R. J., J. A. Becerra, J. Monroy, and F. Bellas (2019). “Perceptual generalization and context in a network memory inspired long-term memory for artificial cognition”. In: *International Journal of Neural Systems* 29, p. 1850053 (cit. on p. 46).

Ede, F. and A. Nobre (2022). “A neural decision signal during internal sampling from working memory”. In: (cit. on p. 19).

Elshaw, M., R. K. Moore, and M. Klein (2010). “An attention-gating recurrent working memory architecture for emergent speech representation”. In: *Connection Science* 22, pp. 157–175 (cit. on p. 25).

Endo, Y. (2008). “Anticipatory robot control for a partially observable environment using episodic memories”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2852–2859 (cit. on p. 79).

Engström, J., S. Liu, A. Dinparastdjadid, and C. Simoiu (2022). “Modeling road user response timing in naturalistic settings: a surprise-based framework”. In: *arXiv preprint* (cit. on p. 79).

Ergorul, C. and H. Eichenbaum (2004). “The hippocampus and memory for “what,” “where,” and “when””. In: *Learning & Memory* 11, pp. 397–405 (cit. on p. 16).

Eric, M., L. Krishnan, F. Charette, and C. D. Manning (2017). “Key-value retrieval networks for task-oriented dialogue”. In: *Annual SIGdial Meeting on Discourse and Dialogue*, pp. 37–49 (cit. on p. 101).

Erlhagen, W. and E. Bicho (2006). “The dynamic neural field approach to cognitive robotics”. In: *Journal of Neural Engineering* 3, R36–R54 (cit. on p. 44).

Eyerich, P., R. Mattmüller, and G. Röger (2012). “Using the context-enhanced additive heuristic for temporal and numeric planning”. In: *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*. Springer, pp. 49–64 (cit. on p. 121).

Fox, M. and D. Long (2003). “PDDL2.1: an extension to PDDL for expressing temporal planning domains”. In: *Journal of Artificial Intelligence Research (JAIR)* 20, pp. 61–124 (cit. on p. 119).

Franklin, S., T. Madl, S. D’mello, and J. Snaider (2013). “LIDA: a systems-level architecture for cognition, emotion, and learning”. In: *IEEE Transactions on Autonomous Mental Development* 6, pp. 19–41 (cit. on pp. 27, 31, 33, 155).

Freedman, S. T. and J. A. Adams (2011). “Filtering data based on human-inspired forgetting”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 41.6, pp. 1544–1555 (cit. on pp. 30, 46, 78, 79).

Funahashi, S. (2013). “Thalamic mediodorsal nucleus and its participation in spatial working memory processes: comparison with the prefrontal cortex”. In: *Frontiers in Systems Neuroscience* 7 (cit. on p. 20).

Gais, S., G. Albouy, M. Boly, T. T. Dang-Vu, A. Darsaud, M. Desseilles, G. Rauchs, M. Schabus, V. Sterpenich, G. Vandewalle, P. Maquet, and P. Peigneux (2007). “Sleep transforms the cerebral trace of declarative memories”. In: *National Academy of Sciences (NAS)* 104, pp. 18778–18783 (cit. on p. 21).

Gardner, E. S. and E. McKenzie (1985). “Forecasting trends in time series”. In: *Management science* 31.10, pp. 1237–1246 (cit. on p. 127).

Georgiou, A., M. Katkov, and M. Tsodyks (2023). “Forgetting dynamics for items of different categories”. In: *Learning & Memory* 30, pp. 43–47 (cit. on p. 28).

Ghallab, M., A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins (1998). *PDDL - the planning domain definition language* (cit. on p. 118).

Ghetti, S. and S. A. Bunge (2012). “Neural changes underlying the development of episodic memory during middle childhood”. In: *Developmental Cognitive Neuroscience* 2, pp. 381–395 (cit. on p. 21).

Gonzalez-Aguirre, J. A., R. Osorio-Oliveros, K. L. Rodríguez-Hernández, J. Lizárraga-Iturralde, R. M. Menendez, R. A. Ramirez-Mendoza, M. A. Ramírez-Moreno, and J. d. J. Lozoya-Santos (2021). “Service robots: trends and technology”. In: *Applied Sciences* 11, p. 10702 (cit. on p. 35).

Grabbe, J. W. (2010). “Olfaction and emotion content effects for memory of vignettes”. In: *Current Research in Psychology* 1, pp. 53–60 (cit. on p. 17).

Grande, X., D. Berron, A. Maaß, W. Bainbridge, and E. Düzel (2021). “Content-specific vulnerability of recent episodic memories in alzheimer’s disease”. In: *Neuropsychologia* 160, p. 107976 (cit. on p. 16).

Grauman, K., A. Westbury, E. Byrne, Z. Chavis, A. Furnari, R. Girdhar, J. Hamburger, H. Jiang, M. Liu, X. Liu, et al. (2022). “Ego4d: around the world in 3,000 hours of egocentric video”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18995–19012 (cit. on pp. 78, 79).

Guennebaud, G., B. Jacob, et al. (2010). *Eigen v3*. http://eigen.tuxfamily.org, last accessed 31.05.2025 (cit. on p. 160).

Harasimczuk, J. (2024). “50 years of research on working memory”. In: *Kwartalnik Naukowy Fides Et Ratio* 58, pp. 77–84 (cit. on pp. 15, 16).

Hart, P., N. Nilsson, and B. Raphael (1968). “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107 (cit. on p. 118).

Hassabis, D., D. Kumaran, S. D. Vann, and E. A. Maguire (2007). “Patients with hippocampal amnesia cannot imagine new experiences”. In: *National Academy of Sciences (NAS)* 104.5, pp. 1726–1731 (cit. on p. 13).

Hassan, T. and S. Kopp (2020). “Towards an interaction-centered and dynamically constructed episodic memory for social robots”. In: *ACM/IEEE International Conference on Human Robot Interaction (HRI)* (cit. on p. 44).

Hebb, D. O. (2005). *The organization of behavior: A neuropsychological theory.* Psychology press (cit. on p. 25).

Hegarty, M., P. Shah, and A. Miyake (2000). “Constraints on using the dual-task methodology to specify the degree of central executive involvement in cognitive tasks”. In: *Memory & Cognition* 28, pp. 376–385 (cit. on pp. 15, 16).

Hegemann, P., T. Zechmeister, M. Grotz, K. Hitzler, and T. Asfour (2022). “Learning symbolic failure detection for grasping and mobile manipulation tasks”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Kyoto, Japan (cit. on pp. 119, 120).

Helmert, M. (2004). “A planning heuristic based on causal graph analysis.” In: *International Conference on Automated Planning and Scheduling (ICAPS).* Vol. 16, pp. 161–170 (cit. on p. 124).

Helmert, M. (2006). “The fast downward planning system”. In: *Journal of Artificial Intelligence Research (JAIR)* 26.1, pp. 191–246 (cit. on p. 128).

Henning, M. (2004). “A new approach to object-oriented middleware”. In: *IEEE Internet Computing* 8, pp. 66–75 (cit. on pp. 58, 81).

Herz, N., B. R. Bukala, J. E. Kragel, and M. J. Kahana (2022). “Hippocampal mechanisms of false recall”. In: (cit. on p. 29).

Hochreiter, S. and J. Schmidhuber (1997). “Long short-term memory”. In: *Neural computation* 9, pp. 1735–80 (cit. on p. 25).

Hoffmann, M. and R. Pfeifer (2018). “Robots as powerful allies for the study of embodied cognition from the bottom up”. In: *arXiv preprint* (cit. on p. 3).

Huang, W., A. Chella, and A. Cangelosi (2024). “A cognitive robotics implementation of global workspace theory for episodic memory interaction with consciousness”. In: *IEEE Transactions on Cognitive and Developmental Systems* 16, pp. 266–283 (cit. on p. 25).

Idrees, I., S. P. Reiss, and S. Tellex (2020). “Robomem: giving long term memory to robots”. In: *arXiv preprint* (cit. on pp. 78, 79).

Ilinykh, N. and S. Dobnik (2021). “What does a language-and-vision transformer see: the impact of semantic information on visual representations”. In: *Frontiers in Artificial Intelligence* 4 (cit. on p. 43).

Infantino, I., G. Pilato, R. Rizzo, and F. Vella (2013). “Humanoid introspection: a practical approach”. In: *International Journal of Advanced Robotic Systems* 10 (cit. on p. 45).

Ioannou, A. and X. Anastassiou-Hadjicharalambous (2018). “Non-associative learning”. In: *Encyclopedia of Evolutionary Psychological Science.* Springer International Publishing, pp. 5419–5432 (cit. on p. 17).

Jaime, K., G. Torres, F. Ramos, and G. G. García (2012). “A proposed model for visual memory identification”. In: *IEEE International Conference on Cognitive Informatics and Cognitive Computing (ICCI*CC)* (cit. on pp. 11, 79).

Jaime, K., G. Torres, F. Ramos, and G. García-Aguilar (2014). “A cognitive architecture for visual memory identification”. In: *International Journal of Software Science and Computational Intelligence* 6, pp. 63–77 (cit. on p. 19).

Jarrold, C., A. Baddeley, and A. Hewes (2000). “Verbal short-term memory deficits in down syndrome: a consequence of problems in rehearsal?” In: *Journal of Child Psychology and Psychiatry* 41, pp. 233–244 (cit. on p. 15).

Joyce, J. M. (2011). “Kullback-leibler divergence”. In: *International Encyclopedia of Statistical Science*. Ed. by M. Lovric. Springer Berlin Heidelberg, pp. 720–722 (cit. on p. 69).

Kasap, Z. and N. Magnenat-Thalmann (2010). “Towards episodic memory-based long-term affective interaction with a human-like robot”. In: *International Symposium in Robot and Human Interactive Communication*, pp. 452–457 (cit. on p. 24).

Kaski, S. (2010). “Self-organizing maps”. In: *Encyclopedia of Machine Learning*. Springer US, pp. 886–888 (cit. on p. 24).

Kawamura, K., S. M. Gordon, P. Ratanaswasd, E. Erdemir, and J. F. Hall (2008). “Implementation of cognitive control for a humanoid robot”. In: *International Journal of Humanoid Robotics* 5, pp. 547–586 (cit. on pp. 26, 31, 33, 152).

Kelley, H. J. (1960). “Gradient theory of optimal flight paths”. In: *Ars Journal* 30.10, pp. 947–954 (cit. on p. 25).

Kieras, D. E. and D. E. Meyer (1997). “An overview of the EPIC architecture for cognition and performance with application to human-computer interaction”. In: *Human–Computer Interaction* 12, pp. 391–438 (cit. on pp. 23, 31, 33, 147).

Kim, C. Y., C. P. Lee, and B. Mutlu (2024). “Understanding large-language model LLM-powered human-robot interaction”. In: *ACM/IEEE International Conference on Human Robot Interaction (HRI)*, pp. 371–380 (cit. on p. 102).

Kingdon, R. (2008). “A review of cognitive architectures”. In: *ISO Project report* (cit. on p. 147).

Kingma, D. P. and M. Welling (2013). *Auto-Encoding Variational Bayes* (cit. on pp. 25, 69).

Kollar, T., S. Tellex, D. Roy, and N. Roy (2014). “Grounding verbs of motion in natural language commands to robots”. In: *Experimental Robotics*. Vol. 79, pp. 31–47 (cit. on p. 101).

Kollar, T., S. Tellex, M. Walter, A. Huang, A. Bachrach, S. Hemachandra, E. Brunskill, A. Banerjee, D. Roy, S. Teller, and N. Roy (2017). “Generalized grounding graphs: a probabilistic framework for understanding grounded commands”. In: *arXiv preprint* (cit. on p. 101).

Kotseruba, I. and J. K. Tsotsos (2018). “40 years of cognitive architectures: core cognitive abilities and practical applications”. In: *Artificial Intelligence Review* (cit. on pp. 22, 24, 26, 30, 41, 142, 147, 149).

Krüger, N., C. Geib, J. Piater, R. Petrick, M. Steedman, F. Wörgötter, A. Ude, T. Asfour, D. Kraft, D. Omrčen, A. Agostini, and R. Dillmann (2011). “Object–Action Complexes: grounded abstractions of sensory–motor processes”. In: *Robotics and Autonomous Systems* 59, pp. 740–757 (cit. on pp. 2, 23, 74).

Laird, J. E. (2008). “Extending the Soar cognitive architecture”. In: *Frontiers in Artificial Intelligence and Applications* 171, pp. 224–235 (cit. on p. 26).

Laird, J. E. (2009). “Toward cognitive robotics”. In: *International Society for Optical Engineering (SPIE)* 7332 (cit. on p. 27).

Laird, J. E. (2019). *The Soar cognitive architecture.* The MIT press (cit. on pp. 24, 26, 153).

Laird, J. E. (2022a). *An analysis and comparison of ACT-R and Soar* (cit. on p. 154).

Laird, J. E. (2022b). *Introduction to Soar* (cit. on pp. 24, 33, 153).

Laird, J. E., C. Lebière, and P. S. Rosenbloom (2017). “A standard model of the mind: toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics”. In: *AI Magazine* 38, pp. 13–26 (cit. on p. 26).

Laird, J. E., A. Newell, and P. S. Rosenbloom (1987). “SOAR: an architecture for general intelligence”. In: *Artificial Intelligence* 33 (cit. on pp. 23, 153).

Langley, P. and D. Choi (2006). “A unified cognitive architecture for physical agents”. In: *National Conference on Artificial Intelligence.* Vol. 21, pp. 1469–1474 (cit. on pp. 23, 31, 33, 149).

Langley, P., J. E. Laird, and S. Rogers (2009). “Cognitive architectures: research issues and challenges”. In: *Cognitive Systems Research* 10.2, pp. 141–160 (cit. on p. 147).

Lei, J., L. Yu, T. L. Berg, and M. Bansal (2019). “TVQA+: spatio-temporal grounding for video question answering”. In: *arXiv preprint* (cit. on p. 101).

Leidner, D. (2024). “Toward robotic metacognition: redefining self-awareness in an era of vision-language models”. In: *ICRA 40th Anniversary of the IEEE International Conference on Robotics and Automation (ICRA@40)* (cit. on p. 43).

Lemus, L., A. Hernandez, and R. Romo (2009). “Neural codes for perceptual discrimination of acoustic flutter in the primate auditory cortex”. In: *National Academy of Sciences (NAS)* 106, pp. 9471–9476 (cit. on p. 19).

Levine, P. A. (2015). *Trauma and memory: Brain and body in a search for the living past: A practical guide for understanding and working with traumatic memory.* North Atlantic Books (cit. on pp. 14, 29).

Li, J. and J. E. Laird (2013). “Preemptive strategies for overcoming the forgetting of goals”. In: *AAAI Conference on Artificial Intelligence* 27, pp. 1234–1240 (cit. on p. 80).

Lieto, A., M. Bhatt, A. Oltramari, and D. Vernon (2018). “The role of cognitive architectures in general artificial intelligence”. In: *Cognitive Systems Research* 48, pp. 1–3 (cit. on pp. 22, 26).

Linden, M., S. Brédart, and A. Beerten (1994). “Age-related differences in updating working memory”. In: *British Journal of Psychology* 85, pp. 145–152 (cit. on p. 15).

Liu, D., M. Cong, and Y. Du (2017). “Episodic memory-based robotic planning under uncertainty”. In: *IEEE Transactions on Industrial Electronics* 64, pp. 1762–1772 (cit. on pp. 24, 43, 46).

Logacjov, A., M. Kerzel, and S. Wermter (2021). “Learning then, learning now, and every second in between: lifelong learning with a simulated humanoid robot”. In: *Frontiers in Neurorobotics* 15, p. 669534 (cit. on p. 79).

Logie, R. H. (2014). “Visuo-spatial working memory”. In: (cit. on pp. 15, 16).

Luan, Z., Y. Lai, R. Huang, S. Bai, Y. Zhang, H. Zhang, and Q. Wang (2024). “Enhancing robot task planning and execution through multi-layer large language models”. In: *Sensors* 24, p. 1687 (cit. on p. 102).

Ma, S., B. Zhang, S. Cao, J. S. Liu, and W. Wang (2021). “Limited memory optimizes cooperation in social dilemma experiments”. In: *Royal Society Open Science* 8, p. 210653 (cit. on p. 20).

Macenski, S., T. Foote, B. Gerkey, C. Lalancette, and W. Woodall (2022). “Robot Operating System 2: design, architecture, and uses in the wild”. In: *Science Robotics* 7 (cit. on pp. 58, 160).

Madore, K. P., A. M. Khazenzon, C. W. Backes, J. Jiang, M. R. Uncapher, A. M. Norcia, and A. D. Wagner (2020). “Memory failure predicted by attention lapsing and media multitasking”. In: *Nature* 587, pp. 87–91 (cit. on p. 29).

Malcolm, K. and J. Casco-Rodriguez (2023). “A comprehensive review of spiking neural networks: interpretation, optimization, efficiency, and best practices”. In: *arXiv preprint* (cit. on p. 24).

Manohar, S., N. Zokaei, S. J. Fallon, and T. P. Vogels (2019). “Neural mechanisms of attending to items in working memory”. In: *Neuroscience & Biobehavioral Reviews* 101, pp. 1–12 (cit. on p. 20).

Masoura, E., A. Gogou, and S. E. Gathercole (2020). “Working memory profiles of children with reading difficulties who are learning to read in greek”. In: *Dyslexia* 27, pp. 312–324 (cit. on pp. 15, 16).

Maxcey, A. M., M. McCann, and S. Stallkamp (2020). “Recognition-induced forgetting is caused by episodic, not semantic, memory retrieval tasks”. In: *Attention, Perception, & Psychophysics* 82, pp. 1539–1547 (cit. on p. 29).

McComas, A. J. (2022). “Clive Wearing and Henry Molaison Reconsidered”. In: *Aranzio’s Seahorse and the Search for Memory and Consciousness*. Oxford University Press. Chap. 38, pp. 261–266 (cit. on p. 13).

McCulloch, W. S. and W. Pitts (1943). “A logical calculus of the ideas immanent in nervous activity”. In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133 (cit. on p. 24).

Micheli, A., A. Bit-Monnot, G. Röger, E. Scala, A. Valentini, L. Framba, A. Rovetta, A. Trapasso, L. Bonassi, A. E. Gerevini, L. Iocchi, F. Ingrand, U. Köckemann, F. Patrizi, A. Saetti, I. Serina, and S. Stock (2025). “Unified planning: modeling, manipulating and solving AI planning problems in Python”. In: *SoftwareX* 29, p. 102012 (cit. on p. 128).

Mishra, C., R. G. Verdonschot, P. Hagoort, and G. Skantze (2023). “Real-time emotion generation in human-robot dialogue using large language models”. In: *Frontiers in Robotics and AI* 10 (cit. on p. 102).

Mizuno, K., E. Dempster, J. Mill, and K. P. Giese (2012). “Long-lasting regulation of hippocampal bdnf gene transcription after contextual fear conditioning”. In: *Genes, Brain and Behavior* 11, pp. 651–659 (cit. on p. 21).

Mokhtari, V., L. S. Lopes, and A. J. Pinho (2016). “Experience-based planning domains: an integrated learning and deliberation approach for intelligent robots”. In: *Journal of Intelligent & Robotic Systems* 83, pp. 463–483 (cit. on p. 120).

Morris, N. and D. M. Jones (1990). “Memory updating in working memory: the role of the central executive”. In: *British Journal of Psychology* 81, pp. 111–121 (cit. on p. 15).

Moscovitch, M. (1994). “Cognitive resources and dual-task interference effects at retrieval in normal people: the role of the frontal lobes and medial temporal cortex.” In: *Neuropsychology* 8, pp. 524–534 (cit. on p. 15).

Murre, J. and A. Chessa (2011). “Power laws from individual differences in learning and forgetting: mathematical analyses”. In: *Psychonomic Bulletin & Review* 18, pp. 592–7 (cit. on p. 28).

Murre, J. M. J. and J. Dros (2015). “Replication and analysis of ebbinghaus’ forgetting curve”. In: *PLoS ONE* 10 (cit. on pp. 28, 68).

Nelson, P., C. Linegar, and P. Newman (2016). “Building, curating, and querying large-scale data repositories for field robotics applications”. In: *Field and Service Robotics: Results of the 10th International Conference*. Springer, pp. 517–531 (cit. on pp. 78, 79).

Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press (cit. on pp. 5, 155).

Nolte, R., M. Pomarlan, A. Janssen, D. Beßler, K. Javanmardi, S. Jongebloed, R. Porzel, J. Bateman, M. Beetz, and R. Malaka (2025). *How metacognitive architectures remember their own thoughts: a systematic review* (cit. on p. 159).

Novianto, R., M.-A. Williams, P. Gärdenfors, and G. Wightwick (2014). “Classical conditioning in social robots”. In: *International Conference on Social Robotics (ICSR)*, pp. 279–289 (cit. on p. 22).

NWG (2025). *Dasgehirn.info: der kosmos im kopf*. https://www.dasgehirn.info, last accessed 31.05.2025 (cit. on p. 18).

OpenAI (2025). *ChatGPT: OpenAI language model*. https://openai.com/chatgpt, last accessed 31.05.2025 (cit. on p. 5).

Oyama, K., Y. Hori, Y. Nagai, N. Miyakawa, K. Mimura, T. Hirabayashi, K. Inoue, T. Suhara, M. Higuchi, and T. Minamimoto (2021). “Chemogenetic dissection of the primate prefronto-subcortical pathways for working memory and decision-making”. In: (cit. on p. 20).

Pashangpour, S. and G. Nejat (2024). “The future of intelligent healthcare: a systematic analysis and discussion on the integration and impact of robots using large language models for healthcare”. In: *Robotics* 13, p. 112 (cit. on p. 102).

Pasukonis, J., T. Lillicrap, and D. Hafner (2022). “Evaluating long-term memory in 3D mazes”. In: *arXiv preprint* (cit. on pp. 78, 79).

Patel, T. N., M. Steyvers, and A. S. Benjamin (2019). “Monitoring the ebb and flow of attention: does controlling the onset of stimuli during encoding enhance memory?” In: *Memory & Cognition* 47, pp. 706–718 (cit. on p. 14).

Paulius, D. and Y. Sun (2019). “A survey of knowledge representation in service robotics”. In: *Robotics and Autonomous Systems* 118 (cit. on pp. 3, 31, 142).

Peller, F., M. Wächter, M. Grotz, P. Kaiser, and T. Asfour (2018). “Temporal concurrent planning with stressed actions”. In: *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pp. 901–908 (cit. on pp. 117, 120–123, 128, 137, 141).

Peller-Konrad, F., R. Kartmann, C. R. Dreher, A. Meixner, F. Reister, M. Grotz, and T. Asfour (2023). “A memory system of a robot cognitive architecture and its implementation in ArmarX”. In: *Robotics and Autonomous Systems* 164, pp. 1–20 (cit. on pp. 7, 12, 27, 33, 41, 42, 44, 45, 49, 58, 60, 61, 63–65, 72, 73, 77, 80, 82, 83, 93, 139, 140).

Pereira, F. C. and T. Ebrahimi (2002). *The MPEG-4 book.* Prentice Hall Professional (cit. on p. 66).

Perera, V., S. P. Selveraj, S. Rosenthal, and M. Veloso (2016). “Dynamic generation and refinement of robot verbalization”. In: *IEEE Int. Symp. on Robot and Human Interactive Communication (RO-MAN)*, pp. 212–218 (cit. on p. 100).

Peters, R. A., K. E. Hambuchen, K. Kawamura, and D. M. Wilkes (2001). “The sensory egosphere as a short-term memory for humanoids”. In: *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pp. 451–459 (cit. on pp. 26, 142).

Pezoa, F., J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč (2016). “Foundations of JSON schema”. In: *ACM International Conference on World Wide Web (WWW).* International World Wide Web Conferences Steering Committee, pp. 263–273 (cit. on p. 49).

Plancher, G. and P. Barrouillet (2019). “On some of the main criticisms of the modal model: reappraisal from a tbrs perspective”. In: *Memory & Cognition* 48, pp. 455–468 (cit. on p. 15).

Plewnia, J., F. Peller-Konrad, and T. Asfour (2024). “Forgetting in episodic long-term memory of humanoid robots”. In: *IEEE International Conference on Robotics and Automation (ICRA).* Yokohama, Japan, pp. 6711–6717 (cit. on pp. 30, 41, 77–80, 90, 94, 97).

Plewnia, J., F. Peller-Konrad, and T. Asfour (2025). “Beyond recall: evaluating forgetting mechanisms for multi-modal episodic robotic memory”. In: *German Robotics Conference (GRC).* Nuremberg, Germany (cit. on p. 77).

Pohl, C., F. Reister, F. Peller-Konrad, and T. Asfour (2024). “MAkEable: memory-centered and affordance-based task execution framework for transferable mobile manipulation skills”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Abu Dhabi, United Arab Emirates, pp. 3674–3681 (cit. on p. 142).

Porter, S. and A. R. Birt (2001). “Is traumatic memory special? a comparison of traumatic memory characteristics with memory for other emotional life experiences”. In: *Applied Cognitive Psychology* 15 (cit. on p. 14).

Al-Qazzaz, N. K., S. H. Ali, S. A. Ahmad, S. Islam, and K. Mohamad (2014). “Cognitive impairment and memory dysfunction after a stroke diagnosis: a post-stroke memory assessment”. In: *Neuropsychiatric Disease and Treatment* 10, pp. 1677–1691 (cit. on pp. 13, 14).

Radulovic, J. (2017). “Using new approaches in neurobiology to rethink stress-induced amnesia”. In: *Current Behavioral Neuroscience Reports* 4, pp. 49–58 (cit. on p. 14).

Raffel, C., N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu (2020). “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *Journal of Machine Learning Research* 21.140, pp. 1–67 (cit. on pp. 106, 107).

Ragusa, F., A. Furnari, and G. M. Farinella (2023). “Meccano: a multimodal egocentric dataset for humans behavior understanding in the industrial-like domain”. In: *Computer Vision and Image Understanding* 235, p. 103764 (cit. on pp. 78, 79).

Rauchs, G., B. Desgranges, J. Foret, and F. Eustache (2005). “The relationships between memory systems and sleep stages”. In: *Journal of Sleep Research* 14, pp. 123–140 (cit. on p. 16).

Reitter, D. and C. Lebiere (2012). “Social cognition: memory decay and adaptive information filtering for robust information maintenance”. In: *AAAI Conference on Artificial Intelligence*. Vol. 26. 1, pp. 242–248 (cit. on pp. 78, 79).

Renoult, L. and M. Rugg (2020). “An historical perspective on endel tulving’s episodic-semantic distinction”. In: *Neuropsychologia* 139, p. 107366 (cit. on p. 16).

Riley, M. R. and C. Constantinidis (2016). “Role of prefrontal persistent activity in working memory”. In: *Frontiers in Systems Neuroscience* 9 (cit. on p. 20).

Rosenthal, S., S. Skaff, M. Veloso, D. Bohus, and E. Horvitz (2013). “Execution memory for grounding and coordination”. In: *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 213–214 (cit. on p. 44).

Rosenthal, S., S. P. Selvaraj, and M. Veloso (2016). “Verbalization: narration of autonomous robot experience”. In: *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 862–868 (cit. on pp. 75, 100).

Rothfuss, J., F. Ferreira, E. Aksoy, Y. Zhou, and T. Asfour (2018). “Deep episodic memory: encoding, recalling, and predicting episodic experiences for robot action execution”. In: *IEEE Robotics and Automation Letters (RA-L)* 3.4, pp. 4007–4014 (cit. on pp. 25, 31, 49, 66, 72, 79, 80, 106, 150).

Rusu, R. B. and S. Cousins (2011). “3D is here: point cloud library (PCL)”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE (cit. on p. 160).

Saive, A., J. Royet, N. Ravel, M. Thévenet, S. Garcia, and J. Plailly (2014). “A unique memory process modulated by emotion underpins successful odor recognition and episodic retrieval in humans”. In: *Frontiers in Behavioral Neuroscience* 8 (cit. on p. 17).

Samsonovich, A. V. (2013). “Emotional biologically inspired cognitive architecture”. In: *Biologically Inspired Cognitive Architectures* 6, pp. 109–125 (cit. on p. 44).

Sánchez, M.-L., M. Correa, L. Martínez, and J. Ruiz-del-Solar (2015). “An episodic long-term memory for robots: the bender case”. In: *RoboCup 2015: Robot World Cup XIX 19*. Springer, pp. 264–275 (cit. on p. 78).

Schacter, D. L. and D. R. Addis (2007). "The cognitive neuroscience of constructive memory: remembering the past and imagining the future". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 362, pp. 773–786 (cit. on p. 30).

Schacter, D. L., D. R. Addis, and R. L. Buckner (2008). "Episodic simulation of future events". In: *Annals of the New York Academy of Sciences* 1124, pp. 39–60 (cit. on p. 30).

Scheutz, M., T. Williams, E. Krause, B. Oosterveld, V. Sarathy, and T. Frasca (2019). "An overview of the distributed integrated cognition affect and reflection DIARC architecture". In: pp. 165–193 (cit. on pp. 30, 79).

Shanahan, M. (2006). "A cognitive architecture that combines internal simulation with a global workspace". In: *Consciousness and Cognition* 15, pp. 433–449 (cit. on p. 27).

Sigalas, M., M. Maniadakis, and P. Trahanias (2017). "Episodic memory formulation and its application in long-term HRI". In: *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN).* IEEE, pp. 599–606 (cit. on p. 78).

Slotnick, S. D. (2004). "Visual memory and visual perception recruit common neural substrates". In: *Behavioral and Cognitive Neuroscience Reviews* 3, pp. 207–221 (cit. on p. 19).

Soikao, R., H. Chikafusa, K. Sugino, D. Kuromaru, and K. Osanai (2024). "A novel approach to autonomous language model contextualization through dynamic knowledge frames". In: *Research Square preprint* (cit. on p. 102).

Sols, I., S. DuBrow, L. Davachi, and L. Fuentemilla (2017). "Event boundaries trigger rapid memory reinstatement of the prior events to promote their representation in long-term memory". In: *Current Biology* 27, 3499–3504.e4 (cit. on p. 21).

Squire, L. R. (2017). "Memory for relations in the short term and the long term after medial temporal lobe damage". In: *Hippocampus* 27, pp. 608–612 (cit. on p. 21).

Squire, L. R. (1986). "Mechanisms of memory". In: *Science* 232.4758, pp. 1612–1619 (cit. on p. 17).

Squire, L. R. (2004). "Memory systems of the brain: a brief history and current perspective". In: *Neurobiology of Learning and Memory* 82 (cit. on p. 15).

Stachowicz, D. and G. M. Kruijff (2012). "Episodic-like memory for cognitive robots". In: *IEEE Transactions on Autonomous Mental Development* 4, pp. 1–16 (cit. on p. 24).

Sun, Q., S. Gu, and J. Yang (2018). "Context and time matter: effects of emotion and motivation on episodic memory overtime". In: *Neural Plasticity* 2018, pp. 1–13 (cit. on p. 29).

Sun, R. and S. Hélie (2013). "Psychologically realistic cognitive agents: taking human cognition seriously". In: *Journal of Experimental & Theoretical Artificial Intelligence* 25, pp. 65–92 (cit. on p. 45).

Sun, R. (2004). "Desiderata for cognitive architectures". In: *Philosophical Psychology* 17 (cit. on p. 74).

Sun, R. (2007). "The importance of cognitive architectures: an analysis based on CLARION". In: *Journal of Experimental & Theoretical Artificial Intelligence* 19, pp. 159–193 (cit. on p. 23).

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction.* MIT press (cit. on p. 25).

Takashima, A., K. M. Petersson, F. Rutters, I. Tendolkar, O. Jensen, M. Zwarts, B. L. McNaughton, and G. Fernández (2006). "Declarative memory consolidation in humans: a prospective functional magnetic resonance imaging study". In: *National Academy of Sciences (NAS)* 103, pp. 756–761 (cit. on p. 21).

Tenorth, M. and M. Beetz (2013). "KnowRob: a knowledge processing infrastructure for cognition-enabled robots". In: *The International Journal of Robotics Research* 32, pp. 566–590 (cit. on pp. 43, 159).

Tenorth, M., D. Jain, and M. Beetz (2010). "Knowledge processing for cognitive robots". In: *KI - Künstliche Intelligenz* 24, pp. 233–240 (cit. on pp. 45, 149).

The OpenCV Reference Manual (2014). 2.4.9.0. Itseez (cit. on p. 160).

Thomas-Antérion, C., K. Jacquin, and B. Laurent (2000). "Differential mechanisms of impairment of remote memory in alzheimer's and frontotemporal dementia". In: *Dementia and Geriatric Cognitive Disorders* 11, pp. 100–106 (cit. on p. 13).

Thompson, R. F. and J. J. Kim (1996). "Memory systems in the brain and localization of a memory". In: *National Academy of Sciences (NAS)* 93.24, pp. 13438–13444 (cit. on p. 11).

Tolstikhin, I., O. Bousquet, S. Gelly, and B. Schoelkopf (2017). *Wasserstein Auto-Encoders* (cit. on p. 69).

Tulving, E. (1972). "Episodic and semantic memory". In: *Organization of Memory*. Academic Press, pp. 381–403 (cit. on pp. 16, 59).

Tulving, E. (1984). "Précis of elements of episodic memory". In: *Behavioral and Brain Sciences* 7, pp. 223–238 (cit. on p. 16).

Unrug, A., A. Coenen, and G. v. Luijtelaar (1997). "Effects of the tranquillizer diazepam and the stimulant methylphenidate on alertness and memory". In: *Neuropsychobiology* 36, pp. 42–48 (cit. on p. 15).

Vahrenkamp, N., M. Kröhnert, S. Ulbrich, T. Asfour, G. Metta, R. Dillmann, and G. Sandini (2012). "Simox: a robotics toolbox for simulation, motion and grasp planning". In: *International Conference on Intelligent Autonomous Systems (IAS)*, pp. 585–594. (Cit. on p. 160).

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). "Attention is all you need". In: *Int. Conf. on Neural Information Processing Systems (NIPS)*, p. 11 (cit. on p. 106).

Vernon, D. (2016). "Reconciling constitutive and behavioural autonomy. the challenge of modelling development in enactive cognition". In: *Intellectica. Revue De l'Association Pour La Recherche Cognitive* 65, pp. 63–79 (cit. on p. 31).

Vernon, D., G. Metta, and G. Sandini (2010). "Embodiment in cognitive systems: on the mutual dependence of cognition and robotics". In: *Advances in Cognitive Systems*, pp. 1–12 (cit. on p. 24).

Vernon, D. (2014a). *Artificial cognitive systems: A primer*. MIT Press (cit. on pp. 1, 26, 44, 147).

Vernon, D. (2014b). "Cognitive system". In: *Computer Vision: A Reference Guide*, pp. 100–106 (cit. on p. 1).

Vernon, D., G. Metta, and G. Sandini (2007). “The iCub cognitive architecture: interactive development in a humanoid robot”. In: *IEEE International Conference on Development and Learning (ICDL).* IEEE, pp. 122–127 (cit. on pp. 120, 147).

Vernon, D., C. Von Hofsten, and L. Fadiga (2011). *A roadmap for cognitive development in humanoid robots.* Springer Science & Business Media (cit. on pp. 31, 42, 45).

Vinanzi, S., M. Patacchiola, A. Chella, and A. Cangelosi (2019). “Would a robot trust you? developmental robotics model of trust and theory of mind”. In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 374, p. 20180032 (cit. on pp. 44, 80).

Wächter, M., E. Ovchinnikova, V. Wittenbeck, P. Kaiser, S. Szedmak, W. Mustafa, D. Kraft, N. Krüger, J. Piater, and T. Asfour (2018). “Integrating multi-purpose natural language understanding, robot’s memory, and symbolic planning for task execution in humanoid robots”. In: *Robotics and autonomous systems* 99, pp. 148–165 (cit. on pp. 27, 119).

Waibel, M., M. Beetz, J. Civera, R. d’Andrea, J. Elfring, D. Galvez-Lopez, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo, et al. (2011). “RoboEarth”. In: *IEEE Robotics & Automation Magazine* 18, pp. 69–82 (cit. on pp. 78, 79).

Wang, H., X. Pan, J. J. Wang, M. Sun, C. Wu, Q. Yu, Z. Liu, T. P. Chen, and Y. Liu (2023). “Dual functional states of working memory realized by memristor-based neural network”. In: *Frontiers in Neuroscience* 17 (cit. on p. 20).

Wang, W., B. Subagdja, A.-H. Tan, and J. A. Starzyk (2012). “Neural modeling of episodic memory: encoding, retrieval, and forgetting”. In: *IEEE Transactions on Neural Networks and Learning Systems* 23.10, pp. 1574–1586 (cit. on pp. 78, 79).

Wang, Z., A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli (2004). “Image quality assessment: from error visibility to structural similarity”. In: *IEEE Transactions on Image Processing* 13.4, pp. 600–612 (cit. on p. 85).

Weberruß, T., L. Bärmann, F. Peller-Konrad, A. Waibel, and T. Asfour (2025). “Personalizing humanoid robot behavior through incremental learning from natural interactions”. In: *German Robotics Conference (GRC).* Nurnberg, Germany (cit. on pp. 37, 115).

Wen, T.-H., D. Vandyke, N. Mrkšić, M. Gašić, L. M. Rojas-Barahona, P.-H. Su, S. Ultes, and S. Young (2017). “A network-based end-to-end trainable task-oriented dialogue system”. In: *Conf. of the European Ch. of the Assoc. for Comput. Linguistics*, pp. 438–449 (cit. on p. 101).

Weng, J. (2002). “A theory for mentally developing robots”. In: *IEEE International Conference on Development and Learning (ICDL).* IEEE, pp. 131–140 (cit. on pp. 25, 31, 33, 157).

Weston, J., S. Chopra, and A. Bordes (2015). “Memory networks”. In: *arXiv preprint* (cit. on p. 101).

Wittmann, B. C., B. H. Schott, S. Guderian, J. U. Frey, H. J. Heinze, and E. Düzel (2005). “Reward-related fMRI activation of dopaminergic midbrain is associated with enhanced hippocampus- dependent long-term memory formation”. In: *Neuron* 45, pp. 459–467 (cit. on pp. 21, 22).

Wixted, J. T. (2004). “The psychology and neuroscience of forgetting”. In: *Annual Review of Psychology* 55, pp. 235–269 (cit. on p. 29).

Wood, R., P. Baxter, and T. Belpaeme (2012). “A review of long-term memory in natural and synthetic systems”. In: *Adaptive Behavior* 20 (cit. on pp. 3, 26, 30, 31, 43, 44).

Wu, C.-S., R. Socher, and C. Xiong (2019). “Global-to-local memory pointer networks for task-oriented dialogue”. In: *International Conference on Learning Representations (ICLR)* (cit. on p. 101).

Yang, C.-Y., E. Gamborino, L.-C. Fu, and Y.-L. Chang (2021). “A brain-inspired self-organizing episodic memory model for a memory assistance robot”. In: *IEEE Transactions on Cognitive and Developmental Systems* 14.2, pp. 617–628 (cit. on p. 79).

Ye, P., T. Wang, and F. Wang (2018). “A survey of cognitive architectures in the past 20 years”. In: *IEEE Transactions on Cybernetics* 48, pp. 3280–3290 (cit. on pp. 24, 26, 147).

Ye, Y., H. You, and J. Du (2023). “Improved trust in human-robot collaboration with ChatGPT”. In: *IEEE Access* 11, pp. 55748–55754 (cit. on p. 102).

Yu, Z., D. Xu, J. Yu, T. Yu, Z. Zhao, Y. Zhuang, and D. Tao (2019). “ActivityNet-QA: a dataset for understanding complex web videos via question answering”. In: *AAAI Conference on Artificial Intelligence* 33.1, pp. 9127–9134 (cit. on p. 101).

Zafar, A., V. B. Parthasarathy, C. L. Van, S. Shahid, A. I. Khan, and A. Shahid (2024). “Building trust in conversational AI: a review and solution architecture using large language models and knowledge graphs”. In: *Big Data and Cognitive Computing* 8, p. 70 (cit. on p. 102).

Zhao, X., M. Li, C. Weber, M. B. Hafez, and S. Wermter (2023). “Chat with the environment: interactive multimodal perception using large language models”. In: *arXiv preprint* (cit. on p. 102).

Zhao, Z., S. Xiao, Z. Song, C. Lu, J. Xiao, and Y. Zhuang (2020). “Open-ended video question answering via multi-modal conditional adversarial networks”. In: *IEEE Trans. on Image Process.* 29, pp. 3859–3870 (cit. on p. 101).

Zhu, Q., V. Perera, M. Wächter, T. Asfour, and M. M. Veloso (2017). “Autonomous narration of humanoid robot kitchen task experience”. In: *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pp. 390–397 (cit. on p. 100).