

Large Language Models for Self-Adaptive Systems: Feedback-Loop Integration and Agentic Adaptation

Kerem Kara[†], Balint Mate^{*}, Vincenzo Scotti[†], Diego Perez-Palacin^{†,‡}, Raffaella Mirandola[†]

[†]*KASTEL, Karlsruhe Institute of Technology, Karlsruhe, Germany*

kerem.kara@student.kit.edu, vincenzo.scotti@kit.edu, raffaella.mirandola@kit.edu

^{*}*FZI Research Center for Information Technology, Karlsruhe, Germany*

mate@fzi.de

[‡]*Linnaeus University, Växjö, Sweden*

diego.perez@lnu.se

Abstract—Self-Adaptive Systems (SAS) address uncertainty and change through feedback loops that monitor the system and its environment, analyze possible adaptation options, plan suitable adaptations, and execute them to achieve system goals. The pervasive uprising of Large Language Model (LLM)-based technologies introduces new opportunities to enhance decision-making in SAS. However, so far, the integration of LLMs raises fundamental architectural questions that have not been sufficiently explored. In this paper, we investigate two architectural paradigms for LLM-driven self-adaptation to gain a practical understanding about whether LLMs should be embedded into existing feedback-loop structures or employed as autonomous, end-to-end adaptive agents. The first paradigm integrates an LLM into the classical MAPE-K feedback loop by replacing the analysis and planning components while preserving explicit monitoring, execution, and knowledge models. The second paradigm adopts an agentic approach in which an LLM operates as an autonomous decision-maker, implicitly realizing the adaptation loop in an end-to-end manner. We instantiate both paradigms on two SAS exemplars to empirically compare them with respect to adaptation quality, stability, responsiveness, and transparency. Our results highlight clear trade-offs between the two approaches. While the LLM-enhanced MAPE-K architecture offers greater controllability and explainability, the agentic approach provides increased flexibility at the cost of reduced architectural clarity and predictability.

Index Terms—Large Language Model, Self-Adaptive Systems, MAPE-K, Agentic AI

I. INTRODUCTION

Recent advances in Large Language Models (LLMs) have renewed interest in increasing the level of autonomy achievable by Self-Adaptive Systems (SASs) [1]. Traditional approaches to engineering self-adaptation typically rely on expert-designed feedback loops, encoded through analytical models, rule sets, or search-based planners that must be carefully crafted and maintained by domain specialists [2]. While these approaches offer strong guarantees and predictability, they often incur high development and maintenance costs and may be difficult to produce or revise in settings characterized by open-ended uncertainty, evolving environments, or limited expert availability [3]. As systems grow in complexity and operate in increasingly dynamic contexts, the dependence on

human experts to continuously specify and refine adaptation logic becomes a limiting factor. This is the exact context where LLMs become a helpful tool. In fact, LLMs exhibit strong capabilities in semantic understanding, pattern matching, and reasoning over unstructured and partially specified information [4]–[6], making them a promising lead to the shift of adaptive decision-making from expert-authored logic to more autonomous, data- and context-driven mechanisms.

In this paper, we investigate how LLMs can be employed as *managing decision-making components* within SAS feedback loops, focusing on their behavior, effectiveness, and operational cost when used in place of, or alongside, expert-designed decision logic. We study two complementary architectural paradigms. The former embeds an LLM within an existing MAPE-K loop, where the model supports or replaces the *Analyze* and *Plan* functions while preserving explicit Monitoring, Execution, and Knowledge components. The latter adopts an *agentic* approach, in which an LLM acts as an autonomous adaptive agent that implicitly realizes the feedback loop end-to-end by iteratively observing the system state, reasoning about adaptation options, and issuing adaptation decisions. By implementing and evaluating these paradigms, we aim to understand how LLMs behave as decision-support components within structured control architectures or as self-contained adaptive agents that internalize the adaptation logic. Ultimately, we investigate what trade-offs these choices entail in terms of control, cost, and reliability.

To address these questions, we implement and empirically evaluate both paradigms on representative SASs exemplars. Our evaluation analyses the effectiveness, responsiveness, and runtime cost of LLM-driven adaptation, as well as the typical failure modes that emerge when LLMs are used in feedback loops. Through this comparative study, we highlight the trade-offs between architectural clarity and controllability on the one hand, and flexibility and autonomy on the other, providing practical insights into the design space of LLM-enabled SASs.

The main contributions of this paper are as follows:

- A conceptual and architectural comparison of two paradigms for integrating LLMs into SASs: (i) embedded integration within the MAPE-K feedback loop, and (ii) agentic, end-to-end LLM-driven adaptation.

This work was supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF) and KASTEL Security Research Labs, Karlsruhe.

- An empirical evaluation of both paradigms on established SASs exemplars, characterizing adaptation quality, runtime and token costs, and the conditions under which LLM-based management approximates or diverges from expert-designed behavior.
- An analysis of common error patterns and limitations of LLMs in adaptive decision-making, providing guidance on when agentic adaptation is viable and when structured feedback-loop integration is preferable.

We organize the remainder of this paper into the following sections. In Section II, we report about current research lines combining SAS with LLMs. In Section III, we detail two alternative implementations of feedback loops augmented with LLMs. In Section IV, we report about the experiments to evaluate the proposed implementations. In Section V we present and discuss the results of our evaluation. In Section VI we discuss the threats to validity. Finally, in Section VII we summarize our contribution and suggest future extensions.

II. RELATED WORK

Despite growing interest in employing LLMs as reasoning components for SASs [1], [7], existing work commits to a single integration architecture per study and does not compare across architectural alternatives. No prior work characterises whether to embed an LLM within an existing MAPE-K loop or run it as an end-to-end agentic adaptation, nor investigates the failure modes that distinguish each architectural option. We address this gap in this paper.

Li et al. [1] survey Generative Artificial Intelligence (GenAI) for SASs and organise the potential benefits around the MAPE-K functionalities and human-on-the-loop interactions, while flagging the inadequacy of current SAS exemplars and benchmarks for evaluating LLM-based methods as a key open challenge. A SEAMS companion paper [7] corroborates this gap, noting that the potential of LLMs in SAS remains largely unexplored due to the absence of relevant literature in flagship SAS venues. The capability of LLMs on SAS-adjacent reasoning and synthesis tasks is well documented in the broader software engineering literature [8], [9], including the production of structured analytical artefacts such as goal models from natural-language descriptions [10]. How well these capabilities transfer to the closed-loop, online setting of self-adaptation is still an open question, and the transfer is non-trivial: stochastic generation can produce unintended or factually incorrect outputs [11] that propagate through the feedback loop.

Earlier learning-based approaches to MAPE-K predate LLMs and span supervised, reinforcement, and interactive learning, primarily applied to updating adaptation rules and policies and to resource management [12]; the closest line of work to ours, by Gheibi and Weyns [3], [13], introduces a lifelong-learning layer that updates classical Machine Learning (ML) models under concept drift, evaluated on the DeltaIoT exemplar [14] that we also adopt. That layer targets long-term model evolution rather than per-iteration reasoning; it does not address whether a more flexible reasoning component can replace hand-crafted rules within a single feedback iteration. ReAct [15] introduced

the agentic-LLM pattern of interleaving chain-of-thought reasoning with tool invocation in a single decision loop. Generative-agents architectures [16] extend the pattern to longer trajectories using language-model-backed memory and planning. The closest precedents to our agentic paradigm in self-adaptive systems explore LLMs as autonomous adaptation agents: Nascimento et al. [17] embed LLMs into a multi-agent realisation of MAPE-K in which each agent’s loop is driven by an LLM that also handles inter-agent natural-language communication, demonstrated on an online-marketplace application; a follow-up [18] applies the same paradigm to a smart-streetlight Internet of Things (IoT) scenario and baselines against neuroevolutionary and engineer-crafted solutions. Donakanti et al. [19] propose the MSE-K (Monitor, Synthesize, Execute) framework in which LLMs synthesise context-sensitive adaptation strategies, and report a single-trace comparison against the SWIM exemplar’s [20] reactive adaptation manager. MSE-K is a single-shot LLM-driven replacement of the Analyse and Plan phases, with conventional Monitor and Execute surrounding it; it corresponds to one point within the embedded LLM-in-MAPE-K design space we study, and does not cover the iterative, tool-using agentic paradigm we also evaluate. A complementary line of SAS work treats the LLM as the managed component rather than the manager: Vázquez et al. [21] formalise prompt engineering for an LLM-based explanation generator as a Partially Observable Markov Decision Process (POMDP) and synthesise a self-adaptive policy that refines prompts in response to user cognitive state. Each of these works establishes the feasibility of one specific configuration: a single agent-level MAPE-K loop with an LLM per agent [17], [18], an MSE-K replacement of Analyse and Plan [19], or an inverted setup with adaptation around an LLM-driven generator [21]. However, none compares architectural alternatives within the same study, and none catalogues the characteristic failures that distinguish them.

Our work compares two architectural alternatives for integrating an LLM into a MAPE-K loop: an embedded replacement of the Analyse and Plan phases (the design point matching Donakanti et al.’s MSE-K) and an end-to-end agentic configuration. Evaluating both against an expert-authored baseline on two SAS exemplars, we characterise when each integration paradigm is preferable and the failure modes specific to each.

III. APPROACH

Hereafter, we provide an overview about feedback loop augmentation via LLMs (Section III-A), we then detail architectural paradigms to embed LLMs in a MAPE-K feedback loop (Section III-B) and to use LLM agents to run the feedback loop autonomously (Section III-C).

A. Overview

Our approach explores different ways of employing LLMs within the decision-making processes of SASs. We consider LLMs not as monolithic replacements of existing control logic, but as adaptable, reasoning, components that can assume different roles within the feedback loop, as shown in the

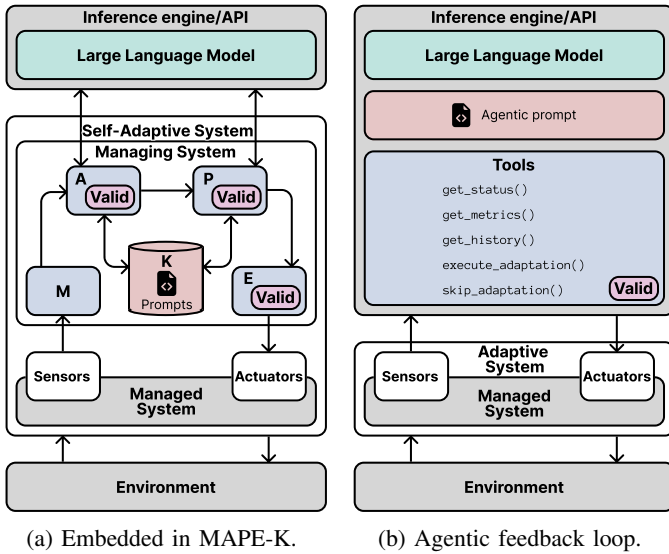


Fig. 1: LLM integration approaches.

architectures proposed in Fig. 1. Between both configurations, the managed system, sensors, and actuators remain unchanged. The distinguishing factor lies in how adaptation decisions are produced and how responsibilities are distributed between traditional feedback-loop components and the LLM. Besides that, both LLM-based strategies rely on structured prompts and expect structured completions (i.e., output generated following a requested structure) to enable automated processing. Prompts include the current system state, relevant monitored metrics, adaptation goals, and execution constraints.

Given the stochastic nature of LLMs outputs, we incorporate a validation mechanism applicable to all the proposed implementation strategies. Typically, validation mechanisms can be of two kinds 1) *constrained generation*, like grammar constrained decoding [22], and 2) *posterior validation*, parsing the generated output to ensure it follows the desired structure like Pydantic¹. For the scope of this work, we limit ourselves to the latter approach using handmade patterns since it represents a simpler and more efficient alternative. With this configuration, whenever the generated output is syntactically invalid, incomplete, or cannot be parsed into executable adaptation actions, the adaptation step (or a sub-step) for the current iteration is skipped. Moreover, if the actions would violate some integrity constraints of the managed system they are ignored. This mechanism prevents malformed decisions from being propagated to the execution phase and allows the feedback loop to continue safely without moving the exemplar into an invalid status.

In the remainder of this section, we outline the two proposed integration approaches. For the sake of simplicity and clarity, we rely on a single running example throughout this discussion; additional examples related to the evaluation phase are provided in the accompanying replication package. Through the description we provide sketches of the prompts

(Listings 1 to 3) and expected completions. These sketches are grounded in an autonomous navigation robot example where the status is described by speed, current direction and sensors information about possible directions (e.g., north direction is viable). Progress toward the goal is quantified by a metric representing the distance from the target point on the grid being navigated. The available actions allow to increase or decrease the speed and change the direction.

B. Embedded Integration

The first architectural paradigm embeds the LLM into a classical MAPE-K feedback loop by replacing selected decision-making components while preserving the overall control structure. Fig. 1a illustrates this embedded integration. In this configuration, the *Monitor*, *Execute*, and *Knowledge* components remain explicit and are implemented using conventional techniques. The use of the LLM is embedded into the *Analyze* and *Plan* components.

The LLM is invoked through an inference engine or some remote API that mediates communication between the feedback loop and the model. During each feedback-loop iteration, monitored data is printed into a structured representation and injected into a role-specific prompt following an LLM-specific chat template. We use separate prompts for analysis and planning (exemplified in Listings 1 and 2) to maintain a clear separation of responsibilities and to improve interpretability of the generated outputs. Moreover, to improve robustness and reduce hallucinations, we employ few-shot prompting using examples sampled from expert executions that demonstrate the correct response format.

Listing 1: Example analyzer prompt and completion.

```
[System] You are an
assistant expert in management of self adaptive systems.
Your role is
to act as the analyzer of an autonomous navigation robot.
The user will provide you with a description
of the current status and metrics and you will decide
whether to adapt (change speed and direction) or not.
[User] Status:
- Direction: north
- Speed: 10/10
- Perception:
  - North: obstacle close
  - East: viable
  - South: closed
  - West: closed
Metric(s):
- Distance from goal: 5 (initial distance 10)
Adaptation needed? [True/False]
[Assistant] True
```

Listing 2: Example planner prompt and completion.

```
[System] You are an
assistant expert in management of self adaptive systems.
Your role is
to act as the planner of an autonomous navigation robot.
The user will provide you with a description
of the current status and metrics and you will decide
the adaptation plan specifying the action(s) to take.
The possible actions are the following:
- 'speed [increase|decrease]'
- 'direction [north|east|south|west]'
[User] Status:
- Direction: north
- Speed: 10/10
```

¹<https://github.com/pydantic/pydantic>

```

- Perception:
  - North: obstacle close
  - East: viable
  - South: closed
  - West: closed
Metric(s):
  - Distance from goal: 5 (initial distance 10)
[Assistant] Adaptation plan:
  - speed decrease
  - direction north

```

The output generated by the LLM is expected to follow a predefined schema encoding either analysis results or concrete adaptation actions. For the analysis step we ask for a True/False indicating whether to run an adaptation or not (see Listing 1). For the planning step we add a prefix string ("Adaptation plan:\n-") to the LLM response (after all the context information) enforcing the generation of a list of actions that should follow the structure described in the system prompt (see Listing 2). These outputs are validated syntactically before being passed to the subsequent components inside the responsible component. During the execute step, we further validate to avoid violating the constraints of the managed system, possibly filtering out more adaptation (sub-)steps.

C. Agentic Integration

The second paradigm adopts an agentic perspective, where the LLM operates as an autonomous adaptive agent that implicitly realizes the feedback loop end-to-end. Fig. 1b depicts this agentic integration. This agentic paradigm emphasizes flexibility and minimal upfront modeling effort, but it reduces transparency and predictability, as adaptation logic is now longer decomposed into explicit feedback-loop components.

Listing 3: Example agent prompt and completion.

```

[System] You are an
assistant expert in management of self adaptive systems.
Your role is to act as
the managing system of an autonomous navigation robot.
You are provided
with the following list of tools to manage the system.
Tools:
- get_status()
  - Description: read latest system status
- get_metrics
  - Description: compute metrics from latest status
- get_history()
  - Description: read last n metrics and actions.
  - Parameter(s):
    - int n
- execute_adaptation()
  - Description: execute action described by parameters
  - Parameter(s):
    - str speed ("increase", "decrease")
    - str direction ("north", "east", "south", "west")
- skip_adaptation()
  - Description: no adaptation needed
Call the tools specifying the parameters in JSON format.
[User] Proceed managing the robot, please.
[Assistant]
Let's start by reading the status and the metrics.
<tool>
  get_status()
  get_metrics()
</tool>
[Output] Status:
- Direction: north
- Speed: 10/10
- Perception:
  - North: obstacle close
  - East: viable

```

TABLE I: Tools for agentic implementation.

Tool	Description
get_status	Read latest system status
get_metrics	Compute metrics from latest status
get_history	Read last n metrics and actions
execute_adaptation	Execute action described by parameters in a JSON string
skip_adaptation	No adaptation needed

```

- South: closed
- West: closed
Metric(s):
  - Distance from goal: 5 (initial distance 10)
[Assistant] I need to understand if the direction is
correct and if the robot is getting closer to the goal.
<tool>
  get_history({"n": 2})
</tool>
[Output] Step -2:
Metric(s):
  - Distance from goal: 7 (initial distance 10)
Adaptation plan:
  - {"speed": "increase", "direction": "north"}
Step -1:
Metric(s):
  - Distance from goal: 6 (initial distance 10)
Adaptation plan:
  - {"direction": "north"}
[Assistant]
The robot is getting closer to the objective, but from
the status I can see we are going towards some obstacle.
I will decrease the speed and stay on the same direction
<tool>
  execute_adaptation({"speed":
"decrease", "direction": "north"})
</tool>
[Output] Adaptation executed successfully.

```

In this configuration, the traditional MAPE-K structure is no longer explicit. Instead, the LLM is prompted with high-level adaptation goals and is granted access to a limited set of tools that expose system observations and execution primitives, as in the example in Listing 3. These tools, described in Table I, allow the agent to retrieve the current system status, query historical information, and trigger adaptation actions.

The LLM iteratively and autonomously alternates between perception, reasoning, and action by invoking tools and updating its internal context (see the [Assistant] part of Listing 3). Rather than producing a single plan per iteration, the agent may condition its decisions on previous outcomes, effectively internalizing the feedback loop within its reasoning process. Whenever an adaptation is issued or skipped the system moves forward one step.

To ensure safe execution, the same output validation and fallback mechanisms used in the embedded approach are applied to agentic decisions at tool level. However, they are limited to managed system integrity and not to the syntactic correctness of the validated input, which is delegated to the wrapper for the agent deployment. Moreover, unlike the embedded MAPE-K architecture, architectural boundaries between analysis, planning, and execution are implicit and not externally observable.

IV. EVALUATION

In this section, we present the Research Questions (RQs) guiding our evaluation (Section IV-A), the experimental setup

TABLE II: Selected LLMs for the integration.

Model	Embedded	Agentic
Qwen3.5-9B	✓	✓
gemma-3-12b-it	✓	✗
Ministral-3-8B-Instruct-2512	✓	✓

(Section IV-B), and the evaluation approach (Section IV-C).

A. Objectives

We design our evaluation to answer the following RQs:

RQ1: *What is the effectiveness of LLM in managing a SAS?*

This question aims at quantifying whether LLMs are capable of managing successfully a SAS compared with other alternative approaches.

RQ2: *What is the cost in time and tokens LLMs require?*

This question aims at quantifying the latency introduced by querying a LLM to do a feedback loop iteration and the energy consumption of the feedback loops.

RQ3: *What are the typical error patterns of LLMs?*

This question aims at discovering what are the limitations of LLMs throughout the feedback loop.

B. Setup

We evaluate the proposed approaches on two established self-adaptive system exemplars that capture different sources of uncertainty, adaptation goals, and decision-making complexity. The first exemplar is *DeltaIoT* [14], a self-adaptive Internet of Things system that manages a wireless sensor network. The adaptation problem consists in balancing energy consumption and communication reliability by dynamically reconfiguring network parameters; namely: transmission power and routing probability. The second exemplar models a *Malware Infection* scenario [23], [24], where adaptation aims to contain the spread of malware in a network by modifying connectivity patterns through removal and restoration of existing connections. In this case, the system must balance security and connectivity: reducing links limits infection spread but also degrades overall network connectivity. An expert-defined strategy is available for both exemplars in the form of handcrafted adaptation rules, which we adopt as a baseline.

For both exemplars we implemented the validation checks described in Section III. Besides parsing (and correcting where possible) the LLM’s generated responses to ensure well-formed output syntax, we introduced some integrity checks to prevent adaptation actions from disrupting the managed system integrity. For *DeltaIoT* we ensure that power could not be set outside the range 0 dB, 15 dB, and that forwarding distributions are equalized in cases where multiple links depart from the same node. The integrity checks applied to *Malware Infection* were designed to enforce that no link removal would disconnect the network, that only originally existing links could be restored, and that each node had a minimum of two edges.

We select a set of open-access instruction-tuned LLMs covering different model families and parameter scales, summarized

in Table II. Our selection includes models that are sufficiently expressive to support reasoning and tool calling while remaining feasible for repeated closed-loop execution. Namely, we selected *Qwen 3.5* [25] (9B parameters), *Gemma 3* [26] (12B parameters), and *Ministral 3* [27] (8B parameters). We explicitly included models in this range of sizes (i.e., number of parameters) to assess whether agentic and embedded approaches are viable without relying on very large or proprietary LLMs and to support reproducibility of our results. We used the *Transformers* library [28] for LLM inference, *Bits-and-Bytes* [29], [30] for LLM quantization and *LangChain*² for the agent deployment.

All our experiments were executed on the same workstation equipped with an AMD Ryzen Threadripper 9970X 32-Cores (64-threads) CPU, 256 GB of RAM and a NVIDIA RTX PRO 6000 Blackwell Workstation Edition GPU.

C. Approach

To enable a systematic comparison, we evaluate the following adaptation strategies:

- 1) *non-adaptive* execution, where no adaptation is performed;
- 2) *random* adaptation, serving as a baseline;
- 3) *expert* strategy, implemented using expert-defined rules;
- 4) *embedded LLM*, where we query the LLM for a decision in the analyze and plan phases of the MAPE-K loop;
- 5) *agentic LLM*, where the LLM replaces the feedback loop in an end-to-end fashion.

As reported in Table II, we implement the embedded approach with all three considered models, while we implemented the agentic approach only with two (Gemma 3 does not support tool calling, making agentic implementation impossible).

For embedded strategies, the LLM receives a structured prompt encoding monitored metrics, adaptation goals, and system constraints, and is required to output a decision conforming to a predefined schema. For agentic strategies, the LLM is provided with a high-level task description and access to tools to observe system state and execute adaptation actions. In both cases, prompts are intentionally kept minimal to reduce prior bias and allow the model to reason autonomously over the provided information.

For each exemplar and adaptation strategy, we execute the feedback loop for a fixed number of iterations and repeat each configuration ten times with different random seeds. This allows us to compute robust aggregate statistics and to mitigate the stochastic variability introduced by both the environment dynamics and the LLM outputs. Unless otherwise stated, model parameters such as temperature and maximum output length are kept constant across strategies to ensure a fair comparison.

For *Delta IoT*, we monitor (1) the total energy consumption, to be minimized, and (2) the packet loss, to be minimized. For *Malware Infection*, we monitor (1) the fraction of healthy nodes, to be maximized, and the (2) the network density (i.e., number of connections between nodes), to be maximized.

These metrics directly operationalize the adaptation goals of each exemplar and are used to assess adaptation

²<https://www.langchain.com/>

effectiveness (RQ1). To address RQ2, we additionally measure feedback-loop response time, LLM inference latency, and token usage per iteration.

To ensure safe execution, we employ a validation layer for all LLM-generated outputs. If an output cannot be parsed or violates structural constraints, the adaptation step is skipped for the current iteration and logged as an error. This mechanism allows us to systematically characterize error patterns (RQ3) without compromising the integrity of the managed system.

V. RESULTS

In this section, we present the results of our experiments (Sections V-A to V-C) and the conclusion we draw from these results (Section V-D).

A. Results for RQ1 (effectiveness of LLMs)

Both exemplars used in the experiments have two competing objectives, forcing the system-adaptation approaches to aim for a trade-off. We compare the distribution of the average objective metrics per run across ten repetitions.

We report results for DeltaIoT in Fig. 2. While Expert produces the lowest energy consumption, it performs the worst among all approaches when it comes to minimizing the packet loss ratio in the system. Oppositely, non-adaptive and Embedded Gemma minimize packet loss most efficiently, in return they also consume the most power. The remaining approaches reach a more balanced compromise between the two competing objectives. Agentic Qwen performs better across both metrics than Agentic Ministral, scoring between Expert and Random for energy consumption and better than both Expert and Random for packet loss. Embedded approaches trade energy consumption for packet loss and position between Random and Non-Adaptive for both metrics.

We report results for Malware Infection in Fig. 3. Similarly to DeltaIoT, no superior approach across both competing metrics was found. Non-adaptive trivially achieves the highest network density distribution as it keeps the network intact even if the ratio of healthy nodes drops significantly. Agentic Qwen and Agentic Ministral sit at the top with respect to the healthy nodes ratio, but Expert or Embedded Qwen manage to retain a higher network density, while still considering system’s health.

Concretely, on DeltaIoT, Agentic Qwen reduces the mean packet-loss ratio by roughly 43% relative to Expert (6.5% vs. 11.4%) at the cost of $\sim 12\%$ higher mean energy consumption (20.6 vs. 18.4). On Malware Infection, both agentic configurations retain a ~ 0.90 mean healthy-node ratio against Expert’s 0.81, while sacrificing ~ 9 percentage points of mean network density (0.80 vs. 0.89 for Agentic Qwen).

Answer to RQ1 LLMs are a viable alternative for managing SAS. Agentic approaches outperform Random in both metrics and Expert in one metric in both case studies, and embedded Ministral and Gemma also do so in Malware Infection. Moreover, none of the LLM-based approaches are outperformed by Random across both case studies.

B. Results for RQ2 (cost of LLMs)

We report *latency* values in Fig. 4. Iteration time spans three orders of magnitude. Non-LLM approaches take ~ 2 ms to calculate the next adaptation and close the feedback loop, embedded LLMs take 1–9 s (median), while agentic LLMs need 11–45 s (median). Within LLM families, Qwen is consistently faster than Ministral on the same task, while Gemma (although only available in the embedded configuration and being larger) is the fastest LLM on Malware Infection (1.1 s median) and matches Qwen on DeltaIoT.

To capture the computational overhead associated with each method, we report *token consumption* in Fig. 5. This cost follows the architectural split rather than the model family. Agentic approaches consume up to roughly twice as many tokens per iteration as their embedded counterparts, since each iteration interleaves tool calls, tool responses, and intermediate reasoning into the context. Tokens grow more rapidly on Malware Infection because the system state encodes more nodes than DeltaIoT and has pairwise links that scale quadratically in the number of nodes.

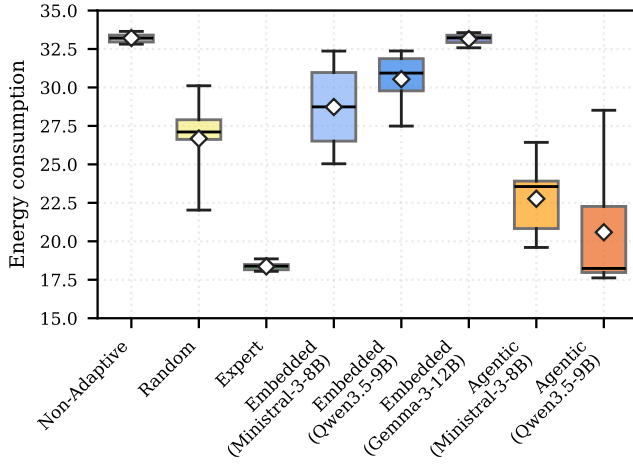
The violin-shaped distributions also expose how variability differs across implementation approaches. Embedded distributions are tight: one prompt equals one completion per iteration, so the stepwise cost is set by the prompt size and decoding length. Agentic distributions are wider and skewed. The number of tool invocations per iteration is itself a decision a model must make. Iterations that converge after one or two tool calls or choose not to adapt fall close to the embedded range, while iterations that loop through multiple observation, reasoning and correction steps stretch the upper tail far above the median.

Finally, the quantitative results reported in this section focus exclusively on *runtime* costs of the adaptation approaches. They do account for the economic and organizational costs associated with designing, validating, and maintaining expert-defined adaptation logic. As a result, the comparison against expert-based strategies should be interpreted as a comparison of *operational performance*, rather than of total cost. In scenarios where expert availability is limited or adaptation logic must evolve rapidly, LLM-based approaches may therefore offer advantages that are not captured by the runtime metrics alone.

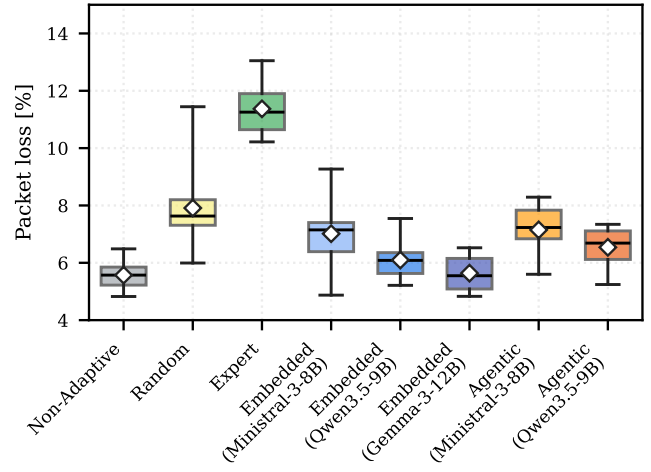
Answer to RQ2 Embedded LLM iterations cost 1–9 s and on the order of 10^3 – 10^4 tokens per step (median). Agentic iterations cost 11–45 s and up to $\sim 1.4 \times 10^4$ tokens. These latencies are three to four orders of magnitude higher than the millisecond-scale rule-based baselines (which incur no token costs), but remain within the adaptation time constraints of both exemplars.

C. Results for RQ3 (errors of LLMs)

Table III reports the mean number of actions an approach proposes per planning step (*Raw*), the subset that satisfies the validity constraints and is executed (*Kept*), and the share of proposed actions that are filtered out during validation (*Inv. %*). For each row we also report the most dominant rejection reason and its share among all rejections it accounts for.

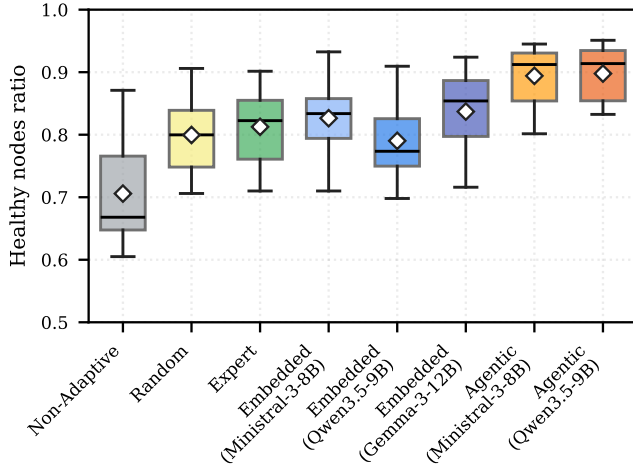


(a) Energy consumption distribution (lower is better).

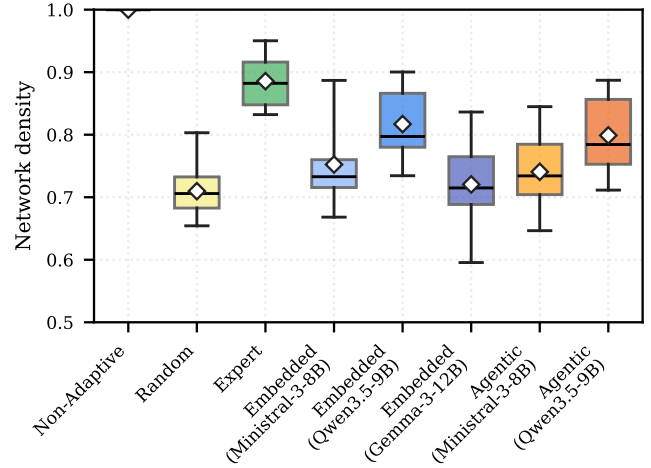


(b) Packet loss distribution (lower is better).

Fig. 2: Per-approach distribution of monitored metrics on DeltaIoT across ten seeded repetitions. Each box covers the per-run averages ($n=10$); the line marks the median across runs and the diamond marks the mean across runs.



(a) Healthy nodes distribution (higher is better).



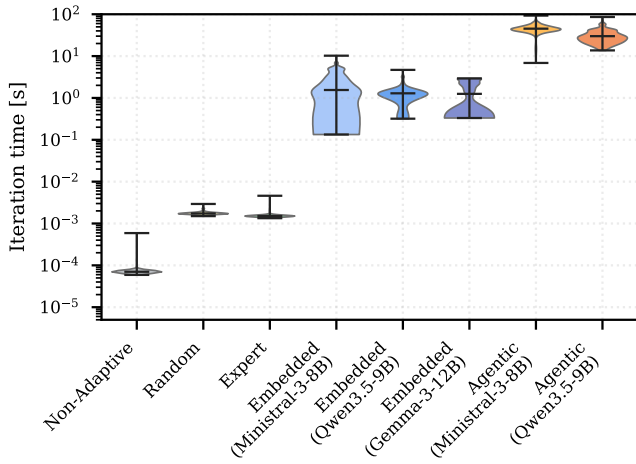
(b) Network density distribution (higher is better).

Fig. 3: Per-approach distribution of monitored metrics on Malware Infection across ten seeded repetitions. Each box covers the per-run averages ($n=10$); the line marks the median across runs and the diamond marks the mean across runs.

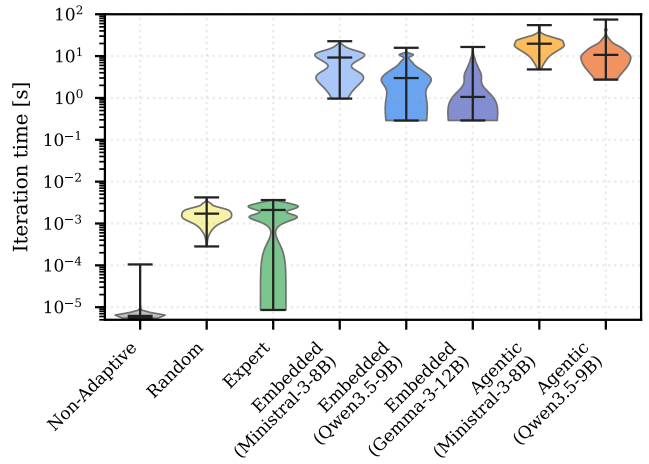
DeltaIoT Valid action generation is more influenced by the model family than by the architectural paradigm. Random’s invalid actions concentrate on `dist_sin_link` attempts, where the strategy proposes to redistribute traffic on a mote that has only one outgoing link, this is a configuration in which distribution change has no effect. Qwen produces 5.6% (embedded) and 10.6% (agentic) invalid proposals, while Ministral exceeds 30% in both configurations, mainly due to `inv_eq` cases as the planner emits an `equalize` directive when distributions values do not equal 100. Gemma scores in between for raw proposal size but the *Inv. %* reaches 60%, entirely driven by attempts to increase transmission power

past the permitted maximum (`power_at_max`). The agentic paradigm modifies the failure profile: the dominant rejection for Agentic Qwen becomes `dist_at_max`, indicating that the agent decides to push distributions past system bounds.

Malware Infection Invalid rates are higher than on *DeltaIoT*, and the failure type `min_edges` dominates, meaning that the model proposes deleting a link that would almost or completely isolate a node. Agentic strategies propose far fewer actions per step in this exemplar (1.0–1.1 raw, 0.1–0.2 kept), so the high *Inv. %* reflects the high share of rejected proposals from an already conservative agent. Embedded Qwen has the most balanced behavior overall, executing ~ 5 actions per step

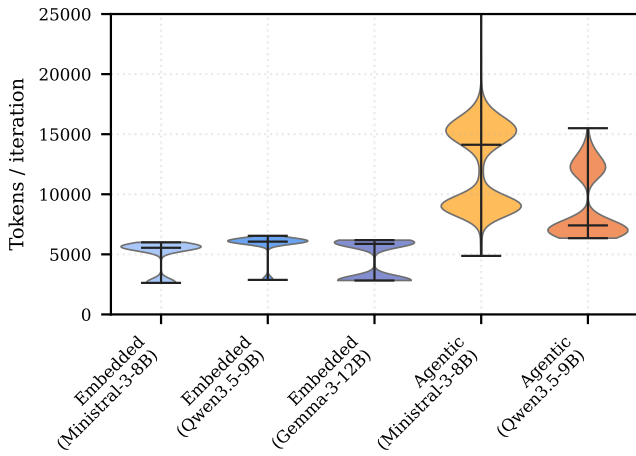


(a) DeltaIoT.

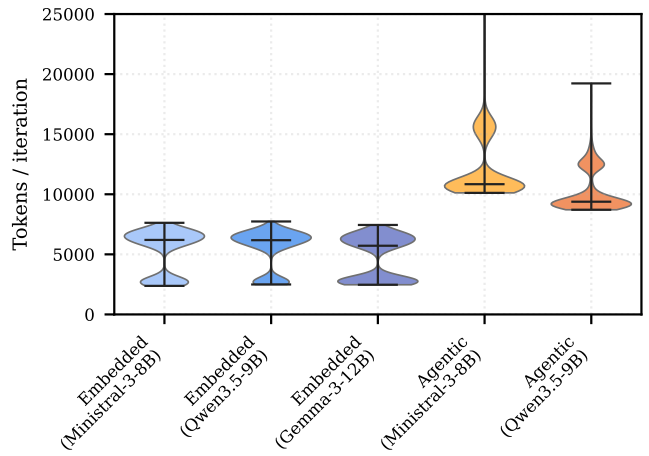


(b) Malware Infection.

Fig. 4: Per-approach distribution of feedback-loop response time across ten seeded repetitions. Each violin pools the per-step values across all reps; the central line marks the pooled median.



(a) DeltaIoT.



(b) Malware Infection.

Fig. 5: Per-approach distribution of LLM token consumption across ten seeded repetitions. Each violin pools the per-step values across all reps; the central line marks the pooled median.

at a 43.5% rejection rate (again minimum-edge violations).

Across both exemplars, the dominant error types are informative. Actions are mainly rejected due to a small set of boundary system constraints, suggesting that emphasizing these limits more explicitly could further reduce rejection rate.

Answer to RQ3 Errors predominantly result from LLMs not obeying explicit system and action constraints (transmission-power and link-distribution maxima on DeltaIoT, minimum node connectivity on Malware Infection). Action validity correlates more strongly with the model family (Qwen consistently outperforms Ministral and Gemma) than with model size or with the integration approach.

D. Discussion

Across the three RQs, the picture is consistent. LLMs are not a drop-in replacement for well-made rule-based approaches: Expert remains Pareto-competitive on DeltaIoT’s energy consumption and Malware Infection’s network density metrics. LLMs instead become attractive where authoring rules is itself a bottleneck. The contribution of LLMs-driven adaptation is therefore less about beating an existing baseline and more about omitting the upfront expert-elicitation step, as the same prompt scaffold reaches expert-comparable trade-offs on two different exemplars without per-system tuning. “Don’t fix what isn’t broken” still applies where an expert solution exists, but when one is unavailable, or when requirements are

TABLE III: Action validity per case study and approach. *Raw/Kept* are mean actions proposed/executed per simulation step. *Inv %* is $100 \times \text{dropped}/\text{raw}$. *Top reason*: dominant validator-rejection reason and its share of all invalids.

Approach	Raw	Kept	Inv. %	Top reason
<i>DeltaIoT</i>				
Non-Adaptive	0.0	0.0	—	—
Random	22.7	13.7	39.5	dist_sin_link (81.8%)
Expert	4.5	4.5	0.0	—
Embedded (Ministral)	3.3	1.9	42.4	inv_eq (92.5%)
Embedded (Qwen)	3.3	3.1	5.6	power_at_max (91.5%)
Embedded (Gemma)	2.7	1.1	60.0	power_at_max (100%)
Agentic (Ministral)	7.1	4.6	34.5	inv_eq (75.0%)
Agentic (Qwen)	2.3	2.0	10.6	dist_at_max (61.7%)
<i>Malware Infection</i>				
Non-Adaptive	0.0	0.0	—	—
Random	27.3	8.1	70.5	no_op (70.9%)
Expert	9.4	9.4	0.0	—
Embedded (Ministral)	4.1	1.6	61.8	min_edges (66.8%)
Embedded (Qwen)	9.2	5.2	43.5	min_edges (70.8%)
Embedded (Gemma)	5.5	1.3	77.2	no_op (51.5%)
Agentic (Ministral)	1.1	0.2	86.3	min_edges (87.4%)
Agentic (Qwen)	1.0	0.1	90.0	min_edges (96.7%)

changing, both the embedded and agentic LLM paradigms are viable alternatives at a known per-step cost.

Scalability Prompt-based representations scale poorly with state size. Malware Infection encodes connectivity pairwise, so the prompt grows quadratically with the number of nodes. This is visible both in Fig. 5 and in the tightening of the agentic action volume on this exemplar (Table III), where context pressure appears to hinder exploration. Context management approaches such as state summarization, sliding-window attention over the network, or learned context selection could mitigate the scalability problem.

The dominant rejection reasons in Table III are interpretable and concentrate on not obeying system bound rules as RQ3 states. With the same model, agentic configurations tend to produce more invalid proposals than embedded ones (Qwen on Malware Infection: 90.0% vs. 43.5%), plausibly because constraints rendered in tool docstrings compete with growing observation-reasoning context across tool calls, while the embedded prompt re-primed the model at each iteration. This suggests paradigm-specific countermeasures: for agentic strategies, lifting constraints from tool docstrings into the agent’s persistent system prompt, embedding bound annotations into tool return values (e.g., `power: 12/15 (at max)`) to refresh working memory, and replacing terse rejection messages with pedagogical ones that let the agent self-correct; for embedded strategies, annotating the rendered state with feasible-action lists rather than only the static action menu may lead to more desirable results. Constrained decoding [22] closes the residual format gap deterministically, and a repair-on-reject loop (a single retry with the rejection reason fed back) would convert most bound violations into valid actions. Plausible but wrong outputs that pass the validation step but steer the system toward a worse state could be

mitigated by simulated execution on top of the current system state and short-horizon lookahead within the agent’s reasoning.

VI. THREATS TO VALIDITY

Construct validity. Construct validity concerns whether the selected metrics and experimental approach accurately capture adaptation effectiveness. We rely on standard system-level metrics commonly used in the considered exemplars (namely: energy consumption, packet loss, fraction of healthy nodes, network density), which strengthens comparability with prior work. However, these metrics may not fully reflect qualitative properties such as stability, adaptation smoothness, or decision interpretability. Moreover, while necessary for automated evaluation, constraining LLM outputs to structured, executable actions may limit the expressiveness of LLM reasoning.

Internal validity. Internal validity threats arise from the stochastic nature of both LLM inference and system dynamics, which may introduce variability in adaptation outcomes. We mitigate this by performing multiple repetitions with different random seeds and reporting aggregate statistics. Another potential contribution may come from prompt design, as differences in instruction phrasing or contextual information could independently influence LLM behavior. To reduce this risk, we standardize prompts across LLM-based strategies and keep them intentionally minimal. Finally, implementation-level mechanisms such as output validation and fallback handling may disproportionately affect some strategies, although they are applied uniformly.

Conclusion validity. Conclusion validity concerns the reliability of the conclusions drawn from experimental results. Given the limited number of exemplars and repetitions, there is a risk of insufficient statistical power or sensitivity to variance, particularly for embedded LLM strategies where the generated completions seem to be strongly affected by the selected few-shot samples. We mitigate this by avoiding overinterpretation of small differences.

External validity. External validity addresses the generalizability of our findings beyond the evaluation setup. Although the selected exemplars cover different adaptation goals and uncertainty profiles, they do not represent the full diversity of self-adaptive systems, such as those with continuous control, safety-critical constraints, or hard real-time requirements. Additionally, our evaluation focuses on open-access, small-to medium-sized LLMs deployed locally; results may differ for larger proprietary models or cloud-based deployments. Finally, our assumptions regarding bounded action spaces and observable system state may not hold in all domains, potentially affecting the applicability of both approaches.

VII. CONCLUSION

In this paper, we investigated the integration of LLMs into SAS, with the goal of understanding whether LLMs can be employed as decision-support components within structured feedback loops or as autonomous, end-to-end adaptive agents. We proposed and instantiated two architectural paradigms: an embedded integration of LLMs into the classical MAPE-K

feedback loop, and an agentic approach in which an LLM implicitly realizes the feedback loop through iterative and autonomous perception, reasoning, and action. Through an empirical evaluation on representative self-adaptive system exemplars, we assessed feasibility, adaptation effectiveness, runtime cost, and typical error patterns. The results showed that LLMs can be seen as a viable fallback or provisional managing components, rather than a replacement for expert-designed self-adaptive managers, when such expertise is unavailable. Moreover, we analyzed what are the risks and costs they introduce depending on the architectural structure.

These initial findings open several promising research directions we are interested in exploring. First, from the empirical evaluation perspective, we are interested in conducting a large-scale experimental campaign to understand the sensitivity of the results to aspects like model size, number of few-shot examples, and to extend it to a more diverse set of exemplars. Second, future work could explore hybrid approaches in which LLMs generate or synthesize the adaptation logic directly (e.g., in the form of rules or code), which is subsequently executed within a conventional feedback loop, combining autonomy with explicit control and efficiency. Finally, we are interested in augmenting LLM-driven adaptation with explainability mechanisms to provide interpretable rationales for adaptation decisions, which are essential for trust, debugging, and deployment, especially in safety-critical domains.

DATA AVAILABILITY

The replication package of our experiments is available at <https://doi.org/10.5281/zenodo.19953252>.

ACKNOWLEDGMENT

Some parts of our work were assisted by Generative AI. We used Anthropic's Claude Opus 4.7 and Microsoft's Copilot (based on OpenAI ChatGPT 5 model family) to help us refine the evaluation setup and the manuscript.

REFERENCES

- [1] J. Li, M. Zhang, N. Li, D. Weyns, Z. Jin, and K. Tei, "Generative ai for self-adaptive systems: State of the art and research roadmap," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 19, no. 3, pp. 1–60, Sep. 2024.
- [2] D. Weyns, *An Introduction to Self-adaptive Systems: A Contemporary Software Engineering Perspective*. John Wiley, October 2020.
- [3] O. Gheibi and D. Weyns, "Lifelong self-adaptation: Self-adaptation meets lifelong machine learning," in *Proceedings of the 17th ACM/IEEE SEAMS*. Pittsburgh Pennsylvania: ACM, May 2022, pp. 1–12.
- [4] OpenAI, "Openai GPT-5 system card," *CoRR*, vol. abs/2601.03267, 2026.
- [5] G. Comanici *et al.*, "Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities," 2025. [Online]. Available: <https://arxiv.org/abs/2507.06261>
- [6] D. Guo *et al.*, "Deepseek-r1 incentivizes reasoning in llms through reinforcement learning," *Nature*, vol. 645, no. 8081, pp. 633–638, Sep 2025. [Online]. Available: <https://doi.org/10.1038/s41586-025-09422-z>
- [7] J. Li, M. Zhang, N. Li, D. Weyns, Z. Jin, and K. Tei, "Exploring the potential of large language models in self-adaptive systems," in *Proceedings of the 19th IEEE/ACM SEAMS*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 77–83.
- [8] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, pp. 220:1–220:79, Dec. 2024.

- [9] Q. Zhang, C. Fang, Y. Xie, Y. Zhang, S. Yu, W. Sun, Y. Yang, and Z. Chen, "A survey on large language models for software engineering," *Science China Information Sciences*, vol. 69, no. 4, p. 141102, Mar 2026. [Online]. Available: <https://doi.org/10.1007/s11432-025-4670-0>
- [10] B. Chen, K. Chen, S. Hassani, Y. Yang, D. Amyot, L. Lessard, G. Mussbacher, M. Sabetzadeh, and D. Varró, "On the use of gpt-4 for creating goal models: an exploratory study," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE, 2023, pp. 262–271.
- [11] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 248:1–248:38, Mar. 2023.
- [12] O. Gheibi, D. Weyns, and F. Quin, "Applying machine learning in self-adaptive systems: A systematic literature review," *ACM Trans. Auton. Adapt. Syst.*, vol. 15, no. 3, pp. 9:1–9:37, Aug. 2021.
- [13] O. Gheibi and D. Weyns, "Dealing with drift of adaptation spaces in learning-based self-adaptive systems using lifelong self-adaptation," *ACM Trans. Auton. Adapt. Syst.*, vol. 19, no. 1, pp. 5:1–5:57, Feb. 2024.
- [14] M. U. Iftikhar, G. S. Ramachandran, P. Bollansée, D. Weyns, and D. Hughes, "Deltaiot: A self-adaptive internet of things exemplar," in *2017 IEEE/ACM SEAMS*, May 2017, pp. 76–82.
- [15] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *The Eleventh International Conference on Learning Representations*, Sep. 2022.
- [16] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '23. New York, NY, USA: Association for Computing Machinery, Oct. 2023, pp. 1–22.
- [17] N. Nascimento, P. Alencar, and D. Cowan, "Self-adaptive large language model (llm)-based multiagent systems," in *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, Sep. 2023, pp. 104–109.
- [18] —, "Gpt-in-the-loop: Supporting adaptation in multiagent systems," in *2023 IEEE International Conference on Big Data (BigData)*, Dec. 2023, pp. 4674–4683.
- [19] R. Donakanti, P. Jain, S. Kulkarni, and K. Vaidhyanathan, "Reimagining self-adaptation in the age of large language models," in *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*, Jun. 2024, pp. 171–174.
- [20] G. A. Moreno, B. Schmerl, and D. Garlan, "Swim: An exemplar for evaluation and comparison of self-adaptation approaches for web applications," in *Proceedings of the 13th IEEE/ACM SEAMS*. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 137–143.
- [21] G. Vázquez, A. Evangelidis, S. Shahbeigi, R. Calinescu, and S. Gerasimou, "Mind the prompt: Self-adaptive generation of task plan explanations via llms," 2026.
- [22] S. Geng, M. Josifoski, M. Peyrard, and R. West, "Grammar-constrained decoding for structured NLP tasks without finetuning," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Association for Computational Linguistics, 2023, pp. 10932–10952.
- [23] V. Grassi, R. Mirandola, and D. Perez-Palacin, "A conceptual and architectural characterization of antifragile systems," *Journal of Systems and Software*, vol. 213, p. 112051, Jul. 2024.
- [24] K. J. Hole, *Anti-Fragile ICT Systems*. Springer International Publishing.
- [25] Q. Team, "Qwen3.5: Accelerating productivity with native multimodal agents," February 2026.
- [26] G. Team, "Gemma 3 technical report," *CoRR*, vol. abs/2503.19786, 2025.
- [27] M. AI, "Minstral 3," *CoRR*, vol. abs/2601.08584, 2026.
- [28] T. Wolf *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, Q. Liu and D. Schlangen, Eds. Association for Computational Linguistics, 2020, pp. 38–45.
- [29] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit optimizers via block-wise quantization," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.
- [30] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Llm.int8(): 8-bit matrix multiplication for transformers at scale," *CoRR*, vol. abs/2208.07339, 2022.