Beyond Monolithic Models: Symbolic Seams for Composable Neuro-Symbolic Architectures

Nicolas Schuler, Vincenzo Scotti, Raffaela Mirandola
KASTEL - Institute of Information Security and Dependability
Karlsruhe Institute of Technology (KIT). Germany
{nicolas.schuler, vincenzo.scotti, raffaela.mirandola}@kit.edu

***Abstract*—Current Artificial Intelligence (AI) systems are frequently built around monolithic models that entangle perception, reasoning, and decision-making, a design that often conflicts with established software architecture principles. Large Language Models (LLMs) amplify this tendency, offering scale but limited transparency and adaptability. To address this, we argue for composability as a guiding principle that treats AI as a living architecture rather than a fixed artifact. We introduce *symbolic seams*: explicit architectural breakpoints where a system commits to inspectable, typed boundary objects, versioned constraint bundles, and decision traces. We describe how seams enable a composable neuro-symbolic design that combines the data-driven adaptability of learned components with the verifiability of explicit symbolic constraints – combining strengths neither paradigm achieves alone. By treating AI systems as assemblies of interchangeable parts rather than indivisible wholes, we outline a direction for intelligent systems that are extensible, transparent, and amenable to principled evolution.**

***Index Terms*—Neuro-Symbolic Architectures, Software Architecture, Machine Learning, AI**

I. Introduction

Artificial Intelligence (AI)[1] has advanced through models of ever-increasing scale, yet this progress has come at a cost to architectural discipline. Software engineering has long emphasized modularity (principled decomposition into cohesive components) [1], [2] and composability (recombination through explicit interfaces) [3]. Contemporary AI systems, by contrast, are frequently constructed as monolithic entities that bind perception, reasoning, and decision-making into opaque structures [4]. In architectural terms, this design trajectory constitutes an anti-pattern [5]: a recurring solution that undermines the principles of maintainability, transparency, and adaptability.

Large Language Models (LLMs) exemplify this anti-pattern. As trained statistical artifacts, they demonstrate extraordinary fluency and predictive capacity, but their design reinforces a paradigm in which scale can substitute for structure and statistical power can eclipse architectural clarity. The *systems* built around such artifacts, which integrate them with data pipelines, orchestration, and control logic, inherit these structural limitations. This creates a critical tension: the achievements of systems based on monolithic AI components conflict with design principles that have guided software architecture for decades [6], [7].

The challenge, then, is not simply technical but architectural. If AI is to evolve beyond its current trajectory, it must be reimagined in terms of design commitments rather than sheer scale. Symbolic AI provides formal structure and – to some extent – verifiability, but struggles to learn from raw data at scale [8]. Data-driven approaches, e.g., Machine Learning (ML), deep learning, and LLMs, achieve remarkable learning capacity, but sacrifice transparency and principled decomposition [8], [9]. Neuro-symbolic integration seeks to combine both strengths [10], yet the architectural foundations for composing learned and symbolic components remain underspecified. This paper offers a first step to tackle that challenge. Using composability as a lens, we connect established software architecture principles to neuro-symbolic designs that combine learned components with explicit symbolic artifacts (rules, constraints, logic-based reasoning) through defined interfaces. With this first step, we outline a reference direction for systems that must evolve, repositioning AI as a living architecture [11].

This paper contributes three elements:

(1) the concept of *symbolic seams* as explicit architectural breakpoints for constraining and recomposing AI systems,
(2) four *design commitments* – typed boundary objects, evolvable constraint configuration, externalized reasoning traces, and bounded change propagation – that operationalize composability in neuro-symbolic settings, and
(3) a research agenda to make these commitments practical through contracts, change-impact reasoning, and constraint governance.

To make this concrete, consider an enterprise assistant that triggers workflows under evolving constraints (access control, approvals). In monolithic and prompt-orchestrated designs, policy updates are costly to validate: retraining may be required, or constraints remain scattered across prompts. A symbolic seam makes boundaries explicit, enabling localized validation when policies change.

Our contribution is architectural. In fact, neuro-symbolic systems already employ explicit neural–symbolic interfaces, and prompt-orchestration frameworks expose intermediate steps and states [12]–[15]. Conversely, we specify seams as explicit connectors, shifting composability from "modular execution"

This work was supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF) and supported by the German Research Foundation (DFG) - SFB 1608 - 501798263 and KASTEL Security Research Labs, Karlsruhe.

[1]We use *AI* as the umbrella term for intelligent systems. Our architectural arguments apply across the AI $\supset$ ML $\supset$ LLM hierarchy.

to "governed recombination under stable contracts", regardless of whether components are neural, symbolic, or hybrid.

II. Background and Related Work

Our argument draws on four research areas: (1) foundational architectural principles that make systems evolvable, (2) empirical evidence that ML systems violate those principles, (3) prompt-orchestration frameworks that modularize execution without governing interaction semantics, and (4) neuro-symbolic approaches that restore composable structure but leave a connector-level gap.

Architectural foundations. Software engineering rests on foundational principles established through decades of empirical research. Examples include Parnas's information hiding [1], Dijkstra's separation of concerns [3], and Baldwin and Clark's theory of modularity [2], collectively demonstrating that principled decomposition improves flexibility and comprehensibility. Maintainability, transparency, and adaptability emerge not coincidentally, but systematically from such decomposition.

Technical debt in ML systems. ML systems incur unique forms of technical debt [5], including what the authors term *abstraction debt*: the difficulty of enforcing strict abstraction boundaries, which degrades modularity. Empirical studies confirm these risks: ML repositories exhibit higher levels of technical debt than non-ML projects [16], often characterized by weak component boundaries and fragile orchestration ("glue code") [17]. While recent research proposes methods to detect ML-specific code smells [18], these approaches often treat symptoms of the underlying architectural tension rather than resolving it.

Prompt-orchestration frameworks. Frameworks such as LangChain [14] and DSPy [15] respond pragmatically to this tension: they decompose AI application logic into pipelines of discrete steps (prompting, retrieval, tool invocation), thereby modularizing flow and exposing intermediate states for inspection. Yet, the constraints governing interaction between steps (input and output schemas, validation rules, consistency requirements) typically remain embedded in prompt templates and ad-hoc checks rather than in governed interfaces.

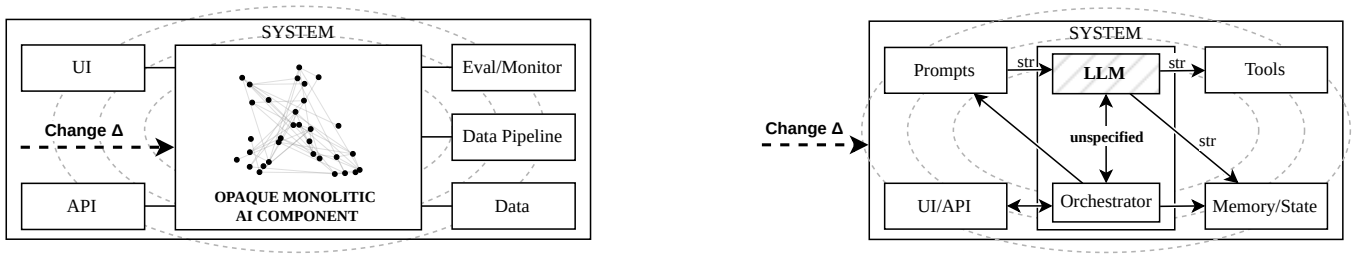
Neuro-symbolic approaches and composability. Recent work on neuro-symbolic AI emphasizes explicit interfaces between neural and symbolic components, motivated in part by concerns that end-to-end integration conflicts with architectural principles that have long proven effective in software engineering and symbolic AI [12]. Neuro-symbolic systems typically partition functionality into distinct layers with defined interfaces: neural components extract patterns from unstructured data, while symbolic components apply logical inference and rule-based constraints to those abstractions [10], [13]. Researchers emphasize that this separation exposes human-readable reasoning traces and supports component-level replacement and evolution [12]. By contrast, purely neural modular designs can also be composed, but their interfaces are often statistical and may require additional alignment when modules are replaced. Existing frameworks such as Logic Tensor Networks [13] and DeepProbLog [19] demonstrate that composable neuro-symbolic designs are implementable and can achieve strong performance where interpretability or domain constraints matter.

Related work and gap. Across these four lines of work, the literature identifies symptoms and provides building blocks, but it does not close the architectural loop: it lacks a connector-level specification that turns inevitable change into bounded, checkable revalidation. Work on ML technical debt, entanglement diagnoses, and glue-code fragility exists [5], [16], [17], yet offers no notion of *seam-level contracts* that scope the impact of updates. Neuro-symbolic frameworks introduce explicit interfaces for injecting constraints [10], [12], [13], but typically treat constraints and traces as task-bound mechanisms rather than *versioned, governed artifacts* that persist across system evolution. Prompt-orchestration frameworks, despite modularizing execution, leave interaction semantics informal and therefore difficult to validate or evolve systematically. Self-adaptive systems (SAS) frameworks such as MAPE-K [20] formalize adaptation loops with architectural models, and ADLs such as Wright [21] specify typed connector protocols. These provide general-purpose adaptation mechanisms but target managed components with predictable behavior; they do not address the stochastic outputs, distributional contracts, and retraining-driven evolution specific to learned components. Symbolic seams close this gap by elevating boundary objects, constraint bundles, and decision traces into *durable connector specifications*: changes are mediated through seam versions, triggering localized validation rather than system-wide revalidation.

III. Why Monolithic AI Resists Architectural Evolution

Before introducing symbolic seams, we examine what makes monolithic architectures brittle. The core pattern is clear; *Symptom*: small changes require broad revalidation; *Cause*: latent coupling from concerns fused in a single parametric space; *Consequence*: expensive evolution and brittle governance. Sculley et al. first articulated the *Changing Anything Changes Everything* (CACE) phenomenon in their study of ML technical debt [5]. While their core argument was framed as the accumulation of technical debt, CACE also points to a deeper architectural problem: *entanglement* in end-to-end trained ML models. Entanglement means that multiple concerns are encoded jointly, so changing one aspect perturbs others in hard-to-predict ways, undermining *architectural scalability* (i.e., the ability to absorb change with bounded impact and diagnosable consequences).

Consider an end-to-end trained ML component such as an LLM: representations, intermediate computations, and output selection are optimized together against a single global loss function. Changes to learned representations (e.g., modifying how entities are encoded) can cascade through downstream computations, which in turn alter decision-making behavior. This coupling is structural: upstream changes shift downstream input distributions, potentially invalidating learned mappings throughout the system [5]. Figure 1a exemplifies this posture: perception, reasoning, and decision-making are collapsed into



(a) Monolitic AI system

(b) Prompt-Orchestrated System

Fig. 1: Current entanglement postures: (a) system is substantially overlapped with its neural logic; (b) entanglement is hidden through informal mechanisms. Δs represent the changes that affect the system.

a single statistical artifact and its surrounding "glue" – an extreme case of entanglement.

This structural entanglement conflicts with at least three foundational software architecture principles. *Separation of Concerns* becomes difficult to enforce when concerns are fused in a single parametric space: responsibilities cannot be cleanly isolated because they share optimization objectives and parameter updates. *Information Hiding* is weakened because the decision pathway remains opaque – there is often no clean boundary where intermediate reasoning is exposed for inspection or validation [22], [23]. *Composability* becomes difficult to achieve: upgrading or swapping one component (e.g., perception) can cascade through reasoning and decision logic, locking the system in mutual dependence. These problems are not merely implementation details to be fixed through better engineering, but constraints imposed by how monolithic AI systems are built and optimized end-to-end [24].

This rigidity manifests in three critical deficiencies:

(1) *Explainability*: decisions emerge from opaque parameter spaces; post-hoc methods typically offer local approximations rather than faithful explanations and remain limited under entanglement [22], [23].
(2) *Adaptability*: model and concept drift (degradation as real-world data distributions shift) often drive expensive full retraining rather than localized component-level updates [25], [26].
(3) *Composability*: perception, reasoning, and decision components are difficult to swap independently, forcing monolithic development or ad-hoc composition without principled interfaces [27].

Prompt-orchestration frameworks (Figure 1b) attempt to address these deficiencies by modularizing execution flows. Yet, the critical semantics of intermediate states and constraints often remain informal and entangled in prompts and ad-hoc checks.

This raises an architectural question: can we reimagine AI systems as decomposable structures in which perception, reasoning, and decision-making are articulated as interoperable, independently evolvable layers?

IV. Symbolic Seams for Composable Architecture

Decomposition is the natural response to architectural brittleness introduced by entanglement, but it requires stable boundaries that preserve the ability to reason about, evolve, and govern a system under change. As Fig. 1 showed, neither the monolithic nor the prompt-orchestrated posture provides such boundaries.

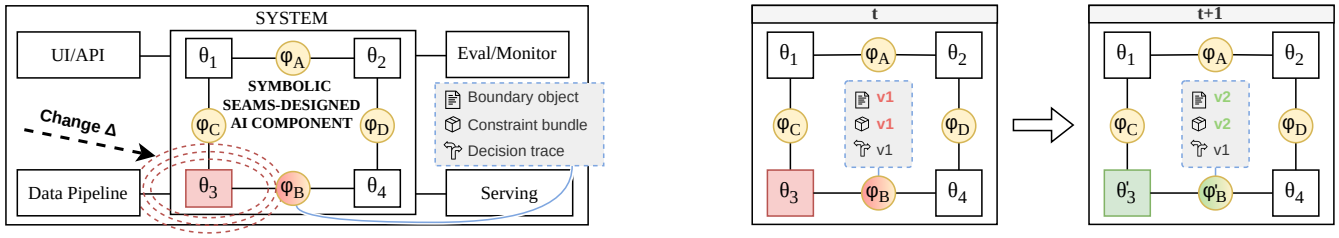
A. Overview

Symbolic seams are our envisoned solution to tackle the challenges of current AI-based systems.

> A symbolic seam is a connector-level constraint checkpoint that emits typed boundary objects, evaluates versioned constraint bundles, and records decision traces.

Seams are not components or modules themselves, but commitments that externalize constraints and trace decisions *between* components. Figure 2a makes this architecture concrete. Learned modules (θ_1–θ_4) handle data-driven tasks, perception, classification, generation, while symbolic seam connectors (ϕ_A–ϕ_D) mediate every crossing between them. Each ϕ validates boundary objects against the active constraint bundle and emits a decision trace before data passes to the next module. When a change arrives (change Δ in the figure), its impact is bounded by the surrounding seams: the dashed circle around θ_3 shows that only the affected module and its adjacent contracts require revalidation, leaving the rest of the assembly unaffected.

Figure 2b illustrates the temporal consequence. On the left, modules θ_1–θ_4 are wired through seams ϕ_A–ϕ_D. On the right, after a change occurred, θ_3 has been replaced by an updated version (marked $\theta_3^{'}$, denoting a retrained, fine-tuned, or swapped variant) with a versioned connector level update of $\phi_B^{'}$. Seams form the system's durable, versioned skeleton, providing stable contract behavior against which each new module version can be validated independently. Seams typically wrap learnable components, but constraints can also be partially internalized when they shape a component's structure or learning objective. They behave as connector contracts (Figure 3) – explicit interfaces that bundle typed boundary objects, constraint configuration, and trace schemas to govern interaction and change propagation. They guarantee checkability of the declared interface but not completeness of all downstream reasoning. Figure 3 scopes the seam's interface commitments, not a full formal semantics. For stochastic components, "typed" means that boundary objects declare not only structure (fields, types, ranges) but also distributional properties (e.g., calibration bounds, coverage guarantees) that the seam can validate at the interface.



(a) Symbolic Seams Posture

(b) Change Propagation

Fig. 2: Proposed posture: (a) learned modules (θ) are separated by symbolic seam connectors (ϕ) that checkpoint constraints; the dashed circle shows change bounded at the affected seam. (b) Seams persist across evolution while modules are replaced (θ' and ϕ' are the updated versions). Effect of change Δ is contained.

$$\mathsf{Seam}(\phi) \triangleq \langle \mathcal{T}_{\text{in}}, \mathcal{T}_{\text{out}}, \mathcal{C}^{(v)}, \mathcal{S}_{\text{trace}} \rangle$$
$$\phi : \mathcal{T}_{\text{in}} \times \mathcal{C}^{(v)} \longrightarrow \mathcal{T}_{\text{out}} \times \mathsf{Status} \times \mathsf{Trace}$$
$$\mathsf{Trace} \triangleq \langle v, \mathsf{evidence} \rangle$$

Fig. 3: A seam (ϕ) commits to typed boundary objects ($\mathcal{T}_{\text{in}}, \mathcal{T}_{\text{out}}$), a versioned constraint bundle $\mathcal{C}^{(v)}$, and a trace schema $\mathcal{S}_{\text{trace}}$. For stochastic components, the contract may specify distributional properties.

	Model-Centric	**Prompt-Orch.**	**Sym. Seams**
Contracts	Implicit	Informal (str)	Typed, versioned
Evolution	Retrain all	Edit prompts	Seam-level config
Change visibility	None	Unclear	Seam validation

TABLE I: Comparison of entanglement postures.

Symbolic seams differ from prior work in three respects, summarized in Table I: (1) they specify interaction at the *connector* level rather than as component-level mechanisms, (2) they treat constraints and traces as *versioned, governed artifacts* rather than task-specific logic, (3) they make *bounded change propagation* an explicit architectural property rather than an emergent side effect. The shift is from exchanging intermediate representations to governing interaction, evolution, and validation at architectural boundaries.

B. Design Commitments

We propose four design commitments that specify the interface properties a symbolic seam must uphold.

Typed, Inspectable Boundary Objects: Components communicate via explicit boundary objects with declared structure and semantics, rather than through implicit latent couplings. A boundary object may be symbolic, neural, or hybrid, but it must be inspectable and testable as an interface artifact [4], [7].

Evolvable Constraint Configuration: Behavioral constraints (policies, domain rules, safety boundaries) are externalized as first-class configuration artifacts that can be injected, modified, and rolled back without full end-to-end retraining. This turns many behavior changes from a re-optimization problem into a governed configuration change.

Externalized Reasoning Traces (Decision Receipts): The system emits intermediate reasoning checkpoints during normal operation, producing bounded, checkable traces for debugging, audit, and governance. Traces are structured execution artifacts (e.g., constraint checks and evidence references), not necessarily free-form natural-language rationales. This makes transparency a system property rather than a retroactive interpretation of opaque outputs.

Bounded Change Propagation: Symbolic seams function as abstraction barriers, where changes behind a seam have diagnosable and bounded impact on other components unless the interface changes. This enables localized evolution (e.g., replacing a perception component) while preserving system-level predictability.

C. Illustrative Scenario: Enterprise Policy Assistant

Considering the enterprise assistant sketched in Section I that answers questions, drafts tickets, and triggers workflows under evolving constraints, in a model-centric design, constraints are absorbed into training data, fine-tuning, or monolithic prompting patterns, making updates costly and difficult to validate. In an orchestrated design, policies are implemented as prompt fragments and scattered checks; workflows are modularized, but constraint semantics remain fragile and difficult to govern.

Under symbolic seam commitments, the assistant is organized around explicit contracts at connector boundaries: each stage emits a typed boundary object that can be validated and logged, rather than keeping semantics implicit in prompts. Boundary objects and traces live *at the seam*: components may produce internal intermediate representations, but the seam commits to externalized, inspectable artifacts. It governs validation, versioning, and trace emission, even when hybrid components emit boundary objects directly. When a policy changes (e.g., a new approval rule), the update becomes a constraint-bundle version change with seam-level regression checks over boundary objects and decision records. If the boundary-object contract remains stable, learned components can be swapped or retrained behind the seam without rewriting downstream policy logic. For a policy update, the postures diverge sharply: monolithic approaches require retraining with full regression; in prompt-orchestrated approaches, edits affect prompts without validation contracts; symbolic seams-based approaches version a constraint bundle with seam-scoped regression and rollback.

Symbolic seams primarily make run-time constraints explicit and governable, though the same commitments support compile-time approaches where constraints shape a component's learning objective [28], [29]. Seen through a composability lens,

seams enable three disciplined operations: (1) *evolve* behavior by editing and versioning constraints, (2) *replace* learnable components behind a seam as long as their boundary-object contract remains stable, and (3) *recompose* workflows by wiring components through typed boundary objects rather than implicit prompt orchestration. This highlights a key distinction: modularity decomposes execution, whereas composability enables safe recombination under explicit contracts. Components can be replaced, reordered, or constrained with bounded validation as long as the seam contract remains stable.

Existing neuro-symbolic approaches demonstrate that neural learning can be combined with explicit symbolic interfaces [13], [19], yet these approaches remain largely domain-specific and require substantial expertise to deploy at scale. The commitments above are architectural standards rather than recipes: closing the gap between these principles and practical tooling remains an open challenge.

V. Discussion and Research Agenda

Symbolic seams offer a complementary direction to the current emphasis on scale and end-to-end optimization, treating constraints, interfaces, and traces as durable architectural artifacts rather than addressing every change through retraining or prompt engineering. Unlike post-hoc explainability methods that infer rationales after the fact, seam-level traces are execution artifacts: intermediate decisions and constraint evaluations are emitted during normal operation and can be validated at the interface, making trace fidelity an architectural requirement. This shifts the question from "why did the model say this?" to "did the system follow the specified constraints and contracts?", turning verification and validation into the primary assurance mechanism. Rather than relying on interpretability alone, engineers can check contract satisfaction at each seam, validate trace completeness against declared decision points, and regression-test constraint enforcement when components change.

To make the above commitments actionable, we outline a research agenda focused on turning seams, constraints, and traces into engineering artifacts with the following concrete challenges:

(1) Interface formalization: How can contracts capture stochastic outputs while remaining checkable?

(2) Change-impact reasoning: How can engineers estimate the behavioral impact of changing constraints or replacing components without exhaustive testing or expensive retraining?

(3) Operational observability and trace fidelity: How can we ensure reasoning traces remain faithful to the actual decision process and operationally reliable as models, constraints, and components evolve?

(4) Tooling for constraint governance: What are the analogues of version control, diffing, and rollback for constraint specifications and their enforced implementations?

Evaluation can borrow from self-adaptive systems by treating policy updates and component swaps as adaptation scenarios and measuring change locality, assurance effort, and contract breakage rate across architectural postures.

That said, composability does not come for free: end-to-end optimization can yield higher raw performance precisely because it exploits cross-component entanglement as a learnable parameter. The architectural question is therefore not whether to decompose, but where to place seams so that the benefits of adaptability, explainability, and governance outweigh the cost of explicit interfaces and constraints – a central trade-off of treating AI as a living architecture.

References

[1] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, p. 1053–1058, Dec. 1972.

[2] C. Y. Baldwin and K. B. Clark, *Design Rules: The Power of Modularity Volume 1*. Cambridge, MA, USA: MIT Press, 1999.

[3] E. W. Dijkstra, *Selected writings on computing: a personal perspective*. Springer Science & Business Media, 2012.

[4] M. Mitchell *et al.*, "Model cards for model reporting," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, ser. FAT* '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 220–229.

[5] D. Sculley *et al.*, "Hidden technical debt in machine learning systems," *NeurIPS*, vol. 28, 2015.

[6] G. A. Lewis *et al.*, "Software Architecture and Machine Learning (Dagstuhl Seminar 23302)," *Dagstuhl Reports*, vol. 13, no. 7, pp. 166–188, 2024.

[7] L. J. Bass *et al.*, *Software architecture in practice*, ser. SEI series in software engineering. Addison-Wesley-Longman, 1999.

[8] H. A. Kautz, "The third ai summer: Aaai robert s. engelmore memorial lecture," *AI Magazine*, vol. 43, no. 1, pp. 105–125, 2022.

[9] A. d. Garcez and L. C. Lamb, "Neurosymbolic AI: The 3rd wave," *Artificial Intelligence Review*, vol. 56, no. 11, pp. 12 387–12 406, 2023.

[10] V. Belle, "On the relevance of logic for artificial intelligence, and the promise of neurosymbolic learning," *Neurosymbolic Artificial Intelligence*, vol. 1, 2025.

[11] I. Ozkaya, "An ai engineer versus a software engineer," *IEEE Software*, vol. 39, no. 6, pp. 4–7, Nov 2022.

[12] A. Mileo, "Towards a neuro-symbolic cycle for human-centered explainability," *Neurosymbolic Artificial Intelligence*, vol. 1, pp. NAI–240 740, 2025.

[13] S. Badreddine *et al.*, "Logic tensor networks," *Artificial Intelligence*, vol. 303, p. 103649, 2022.

[14] H. Chase, "LangChain," Oct. 2022. [Online]. Available: https://github.com/langchain-ai/langchain

[15] O. Khattab *et al.*, "Dspy: Compiling declarative language model calls into self-improving pipelines," *CoRR*, vol. abs/2310.03714, 2023.

[16] D. OBrien *et al.*, "23 shades of self-admitted technical debt: an empirical study on machine learning software," in *FSE*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 734–746.

[17] R. Nazir *et al.*, "Architecting ml-enabled systems: Challenges, best practices, and design decisions," *Journal of Systems and Software*, vol. 207, 2024.

[18] G. Recupito *et al.*, "Technical debt in ai-enabled systems: On the prevalence, severity, impact, and management strategies for code and architecture," *Journal of Systems and Software*, vol. 216, 2024.

[19] R. Manhaeve *et al.*, "Deepproblog: Neural probabilistic logic programming," in *NeurIPS*, vol. 31. Curran Associates, Inc., 2018.

[20] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[21] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 3, p. 213–249, Jul. 1997.

[22] C. Rudin *et al.*, "Interpretable machine learning: Fundamental principles and 10 grand challenges," *CoRR*, vol. abs/2103.11251, 2021.

[23] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, May 2019.

[24] A. Bhatia *et al.* An Empirical Study of Self-Admitted Technical Debt in Machine Learning Software.

[25] D. Vela, A. Sharp, R. Zhang, T. Nguyen, A. Hoang, and O. S. Pianykh, "Temporal quality degradation in ai models," *Scientific Reports*, vol. 12, no. 1, p. 11654, Jul 2022.
[26] T. Glasmachers, "Limits of end-to-end learning," in *Proceedings of the Ninth Asian Conference on Machine Learning*, vol. 77. Yonsei University, Seoul, Republic of Korea: PMLR, 15–17 Nov 2017, pp. 17–32.
[27] R. Pan and H. Rajan, "On decomposing a deep neural network into modules," in *FSE*. ACM, 2020, pp. 889–900.
[28] S. N. Tran and A. S. d'Avila Garcez, "Logical boltzmann machines," *CoRR*, vol. abs/2112.05841, 2021.
[29] Y. Du *et al.*, "Compositional visual generation with energy based models," in *NeurIPS 2020, December 6-12, 2020, virtual*, 2020.