POLARIS: Is Multi-Agentic Reasoning the Next Wave in Engineering Self-Adaptive Systems?

Anonymous Author(s)

Abstract

The growing scale, complexity, interconnectivity, and autonomy of modern software ecosystems introduce unprecedented levels of uncertainty, challenging the foundations of traditional self-adaptation. Existing self-adaptive techniques, often based on rule-driven controllers or isolated learning components, struggle to generalize across novel contexts and coordinate responses across distributed subsystems, leaving them ill-equipped to handle emergent *"unknown unknowns"* in complex, dynamic environments. Recent discussions on *Self Adaptation 2.0* established an equal partnership between AI and adaptive systems, merging learning-driven intelligence with adaptive control to enable predictive, data-driven, and proactive adaptation. Building on this foundation, we introduce a new wave of self-adaptation through *POLARIS* a three-layer multi-agentic self-adaptation framework that moves beyond reactive adaptation by combining (1) a low-latency Adapter layer for monitoring and safe execution, (2) a transparent Reasoning layer that generates and verifies plans using tool-aware, explainable agents, and (3) a Meta layer that records experiences and meta-learns improved adaptation policies over time. By using shared knowledge and predictive models, POLARIS can handle uncertainty, learn from its past actions, and evolve its strategies, enabling the engineering of autonomous systems that can anticipate change to ensure resilient and goal-directed behavior under uncertainty. Preliminary evaluation across two distinct self-adaptive exemplars, SWIM and SWITCH, demonstrates that POLARIS consistently outperforms existing state-of-the-art baselines. With this, we motivate a shift towards *Self-Adaptation 3.0* akin to *Software 3.0*, a new paradigm where systems move beyond merely learning from their environment to reasoning about and evolving their own adaptation. In this vision, adaptation contributes to a self-learning process that enables systems to continuously improve and respond to novel challenges.

1 Introduction

Modern software systems are increasingly required to operate autonomously in dynamic and uncertain environments. From large-scale distributed infrastructures to cyber-physical systems, the ability to self-adapt has become essential for maintaining resilience and long-term performance. Traditional self-adaptive systems, typically governed by feedback loops and control-theoretic models, have been successful in managing predictable or modeled uncertainties [2, 7, 12, 31]. However, the rapid integration of artificial intelligence (AI) into software ecosystems has fundamentally altered this landscape. AI-enabled systems introduce new forms of non-determinism, stemming not only from environmental variability but also from the adaptive and opaque nature of AI components themselves. While existing frameworks have started to incorporate AI-based prediction or decision modules, they largely treat AI as an auxiliary tool within classical control loops. This limits their ability to manage *emergent and evolving uncertainties* such as data drift, stochastic models, or co-evolving agents: scenarios increasingly common in AI-native systems. Moreover, adaptation in such systems must go beyond reactive adjustments; it must *learn from adaptation itself.* Systems should continuously refine their reasoning and decision strategies over time, improving with experience rather than merely responding to change [20]. This need for continual, self-improving adaptation marks a shift toward *AI-native self-adaptation*, where learning and reasoning are integral to the adaptation process.

Building on the vision of *Self-Adaptation 2.0* by Bureš [4], we argue that the next wave of self-adaptive systems [45] must treat intelligence as a first-class design principle. In this work, we present *POLARIS-Proactive Orchestrated Learning for Agentic and Reasoning-driven Intelligent Systems*, which redefines adaptation as an emergent property of interacting agents. POLARIS adopts and builds into a three-layer architecture proposed by Kramer et al. [22], comprising: (1) an Adapter layer for monitoring and safe execution; (2) a transparent Reasoning layer with tool-aware, explainable agents for planning and verification; and (3) a Meta layer to record experiences and meta-learn improved adaptation policies over time. These three layers combine to give the framework its orchestrated and continuous learning nature. We evaluate *POLARIS* on two distinct exemplars, *SWIM* [36] and *SWITCH* [29], highlighting its performance and broad applicability across both traditional legacy software systems and modern AI-based ones. On *SWIM*, *POLARIS* achieves a Total Utility value of 5445.48 succeeding the existing baselines. On *SWITCH*, *POLARIS* shows significant efficiency gains over the existing baseline, achieving a 27.3% lower median response time, 14.9% lower CPU usage, and an 87.1% reduction in disruptive switches. These results collectively demonstrate that *POLARIS* not only adapts effectively to diverse system architectures but also delivers measurable improvements in performance and resource efficiency. The complete replication package is made available .[1]

2 Related Work

The foundations of self-adaptation are built upon structured control loops for autonomic behavior, notably the *MAPE-K* model, which relied on a largely centralized and sequential organization [2, 18]. To improve modularity and scalability, hierarchical patterns such as the three-layer model [21, 46] distinguished component control, change management, and goal management, while decentralized and multi-loop approaches explored distributed coordination through multi-agent systems (MAS) [47]. These frameworks significantly advanced autonomy and resilience, but largely left open how systems can reason about and change their own adaptation mechanisms. Addressing runtime uncertainty has been a central challenge. Techniques like *POISED* [12] and uncertainty-reduction tactics [31] improve decision quality under incomplete information, while quantitative approaches that use probabilistic models and

[1]https://anonymous.4open.science/r/POLARIS-F1F3/

model checking [6, 7] reason about adaptation strategies with confidence. Control-theoretic hybrids [5], predictive frameworks such as *RAINBOW* [13], and model-based approaches like *CobRA* [1], with comparative studies validating their effectiveness [34], extend foresight and dependability. Collectively, these contributions form a rigorous foundation for dependable adaptation. However, their primary focus remains on system-level behavior rather than on enabling reasoning or self-evolving strategies. The integration of machine learning (*ML*) marked a transition toward data-driven decision-making. Systematic reviews [15] highlight the use of supervised and reinforcement learning for environment modeling and policy optimization. Online reinforcement learning (*RL*) methods have been applied to learn policies at runtime; for example, feature-model-guided exploration speeds up online *RL* in systems with large action spaces or evolving configurations [19, 30]. Other studies combine *ML* with formal analysis [10], and recent work on lifelong adaptation enables strategies to improve with experience [14]. *Self-Adaptation 2.0* [4] further advocates an AI-native paradigm. Li et al. [26] examine this intersection through a survey of Generative AI, with recent works demonstrating its use for tasks like goal-model generation within *MAPE-K* [37]. Subsequent works like *AWARE* [39] and *MSE-K*[11] build on this view. Although these works contribute compelling techniques, they employ AI to augment classical control functions without establishing how reasoning becomes an analytical activity that accounts for AI-induced uncertainty. The notion of adaptation as an emergent property of interacting reasoning agents: where cognitive processes themselves become the substrate of adaptation, remains underexplored. In parallel, the rise of *agentic AI* is redefining software architecture. LLMs, seen as autonomous reasoning components [40, 48], are prompting new architectural patterns [28, 42] and agentic orchestration frameworks [16, 43]. These lines of research point toward systems where prediction, verification, learning, and negotiation are performed by collaborating agents. Building on these trends, POLARIS operationalizes *AI-native self-adaptation* through reasoning-enabled agents that unify predictive foresight and runtime assurance with continual learning within a multi-agentic framework.

3 Motivation

Ever since Kephart and Chess proposed the vision of autonomic computing [18], the broader goal has been to build systems that can manage themselves with minimal human intervention. The original MAPE-K model laid the foundation for this vision by introducing feedback loops for monitoring, analysis, planning, and execution. But true autonomy requires more than automated control. It demands systems that can reason about their actions, learn from experience, and continually improve their adaptation strategies.

In practice, when faults occur, human experts collaborate, reason through causes, and decide how to respond. Achieving this kind of autonomous reasoning and teamwork in software has been a long-standing problem. Traditional feedback-based approaches remain reactive in nature as they adapt but rarely learn. Bureš et al. [4] describe this limitation as the threshold between early self-adaptive systems and what they call Self-Adaptation 2.0, where data-driven and AI techniques make adaptation predictive and proactive. Yet, even Self-Adaptation 2.0 stops short of reasoning and self-improvement. To reach that level, systems must evolve toward Self-Adaptation 3.0, where reasoning, learning, and coordination work together, thereby allowing adaptation itself to adapt. The rise of Agentic AI now makes this vision realistic: a collection of goal-driven LLM-based agents can function like a team of expert engineers, analyzing context, proposing actions, and improving over time. This shift parallels the move from Software 2.0 to Software 3.0 as articulated by Andrej Karpathy [2]. It marks the next leap: systems powered by large language models (LLMs) and reasoning engines that flexibly access, interpret, and orchestrate both symbolic (Software 1.0) and learned (Software 2.0) components. In this era, natural language becomes the new programming interface, and large models act as intelligent engines capable of reasoning, collaboration, and adaptive behavior. Yet, relying solely on a single LLM for reasoning introduces challenges, such as hallucination, lack of context retention, and inconsistent decision quality, necessitating a design that actively guides these models through feedback, improvement, and tool-mediated interaction [17]. As a result, software moves from passive execution toward *reasoning-driven adaptation*, where systems understand, plan, and respond to goals in real time. Incorporating this paradigm into self-adaptive systems calls for a new generation of adaptive intelligence. Such systems must integrate reasoning, continual learning, and distributed coordination to handle the unknowns of open, dynamic environments. These requirements motivate *POLARIS*, which unites proactive reasoning, reflective learning, and coordinated multi-agent intelligence to achieve robust and autonomous adaptation. Taken together, this direction points toward what could be seen as the next stage in the evolution of self-adaptation. POLARIS aims to address the following key challenges:
CH1: Generalizable, Reasoned Adaptation: Integrate language-based reasoning, reflective learning, and predictive modeling to anticipate and plan adaptive responses under uncertainty.
CH2: Learning-Driven Evolution: Capture and reuse experiential knowledge through meta-learning to continuously refine adaptation strategies and decision quality.
CH3: Coordinated Multi-Agent Intelligence: Enable coherent, goal-aligned collaboration among autonomous agents to achieve distributed yet globally consistent adaptations.

4 Agentic Preliminaries

This section establishes the foundational concepts underlying the agentic organization of POLARIS, namely the notions of *agents*, *AI agents*, and *tools*. These definitions provide the conceptual basis for distinguishing between autonomous, supportive, and facilitating entities within the framework.

Agent - An *agent* is a computational entity that operates autonomously within an environment to pursue defined objectives. According to Wooldridge [47], an agent is "a computer system capable of autonomous action in an environment to meet its design objectives." Agents possess local goals, perceive their operational context, and act upon it to maintain or improve system-level performance.
AI Agent - An *AI agent* extends a basic agent by incorporating reasoning, learning, and adaptive decision-making capabilities. Unlike

[2]https://www.ycombinator.com/library/MW-andrej-karpathy-software-is-changing-again

reactive agents that follow fixed rules, AI agents can interpret context, predict outcomes, and refine their strategies over time using AI models as their "cognitive core" [28].
Tool - A *tool* refers to a non-autonomous software artifact that supports agents in perception, reasoning, or learning but lacks independent goals or agency. It extends an agent's capabilities and efficiency[38].

Beyond these, the framework also includes other infrastructural and structural elements that facilitate communication, coordination or execution across the system. These components serve as the bridge enabling agent collaboration and interaction with the managed environment.

5 POLARIS: Overview and Methodology

POLARIS orchestrates self-adaptation through a structured, multi-agent architecture designed for proactive reasoning and continuous improvement. Its design, visualized in Figure 1, consists of three conceptual layers: *Adaptive Control*, *Reasoning & Knowledge*, and *Meta-Learning*, that operationalize adaptation across different timescales. This layered approach, inspired by the three-layer reference model [21], provides a clear separation of concerns, distinguishing immediate system interaction from deliberative reasoning and long-term strategic evolution.

Table 1: Overview of POLARIS Core elements and Responsibilities

Component	Layer	Primary Responsibility
Kernel (K)	*Adaptive Control*	Central orchestrator; triages system states and dispatches control to the appropriate loop.
Metric Collector (MC)	*Adaptive Control*	Gathers raw and derived performance telemetry from the managed system.
Execution Adapter (EA)	*Adaptive Control*	Translates abstract adaptation commands into domain-specific, executable actions.
Reasoner (R)	*Reasoning*[*]	AI-driven agentic system that uses tools for deliberative, evidence-based strategic planning.
Verifier (V)	*Reasoning*	Acts as a safety backstop, validating proposed plans against operational invariants.
Fast Controller (F)	*Reasoning*	Executes a predefined, low-latency policy to rapidly stabilize the system during crises.
Knowledge Base (KB)	*Reasoning (Tool)*	Serves as the system's episodic memory, storing historical experiences and learned patterns.
World Model (WM)	*Reasoning (Tool)*	Provides a predictive, simulatable model for "what-if", counterfactual reasoning.
Meta-Learner (M)	*Meta-Learning*[*]	Analyzes long-term adaptation history to identify sub-optimal patterns and evolve system policies.

[*]Dedicated AI components that require reasoning cores; can instantiate multiple specialized AI agents with local knowledge.

5.1 Operational Flow

Formally, the POLARIS framework ($\mathcal{F}$) comprises three primary constituent sets that define its structure and capabilities:

$$\mathcal{A} = \{R, M, V, F\}, \quad \mathcal{T} = \{KB, WM\}, \quad C = \{K, MC, EA\}$$

Here, $\mathcal{A}$ represents the set of *active agents* responsible for decision-making: the **Reasoner** (R), **Meta-Learner** (M), **Verifier** (V), and **Fast Controller** (F). $\mathcal{T}$ is the set of *foundational tools* that support deliberation: the **Knowledge Base** (KB) and **World Model** (WM). Finally, C is the set of *operational components* forming the interface to the managed system: the **Kernel** (K), **Metric Collector** (MC), and **Execution Adapter** (EA).

As depicted in Figure 1, these components collaborate across three nested feedback loops:

- an immediate *ReactiveStabilization* loop
- a short-term *ProactiveAdaptation* loop
- and a long-term *Meta-Learning* loop

To ground the discussion, we illustrate the architectural flow using the SWIM exemplar [36] as a representative case study, where adaptation involves dynamically scaling the servers and adjusting request processing fidelity using *dimmer* control.

5.2 Reactive Stabilization

The innermost loop guarantees immediate, bounded-latency responses to critical system failures, prioritizing stability.

Trigger and Dispatch. The process begins with the **Metric Collector (MC)**, which provides a continuous stream of telemetry. This data flows to the **Kernel (K)**, the framework's central orchestrator. The Kernel analyzes each metric observation m against a critical threshold $\theta_{critical}$. If a violation is detected (e.g., $m > \theta_{critical}$), the Kernel activates the *Stabilization Path* (illustrated in Fig 1), immediately delegating control. For instance, if the average response time m_{rt}=900 ms exceeds a $\theta_{critical\_rt}$ of 750 ms, this loop is triggered.

Reactive Action. Control is passed to the **Fast Controller (F)**, a lightweight agent executing a predefined policy. This policy consists of simple, reliable rules that map an observed system state s to a corrective action a_{react}:

$$a_{react} = \Phi(s)$$

For example, Φ might map the high response time state to a 'set dimmer to 70%' action. As illustrated in Figure 1, this action is routed through a verification stage before execution. It is then passed to the **Execution Adapter (EA)**, which translates it into a domain-specific command, ensuring the system is rapidly returned to a known-safe state.

5.3 Proactive Adaptation

When the system is not in immediate crisis, POLARIS engages its deliberative reasoning capabilities.

Trigger and Context. This loop operates as the system's default, continuous adaptive reasoning cycle. The **Kernel** invokes the **Reasoner Agent (R)** (using the *Strategic Adaptation* path as shown in Fig 1) with a specific reasoning context, derived from observed system metrics and operational state:

$$P_R = \langle G, \mathcal{I}, \mathcal{T} \rangle$$

This context defines the task: G specifies optimization goals (e.g., minimize cost), $\mathcal{I}$ represents hard invariants (e.g., `max_servers` $\leq$ 3), and $\mathcal{T}$ provides tool interfaces. The Reasoner operates autonomously, using telemetry and internal heuristics to detect emerging trends or inefficiencies warranting action.

Prompt Configuration. The Reasoner (which has an LLM agent(s)) operates using a structured *prompt configuration* that encodes system constraints, adaptation thresholds, and reasoning workflow templates. This configuration defines how contextual data, goals,

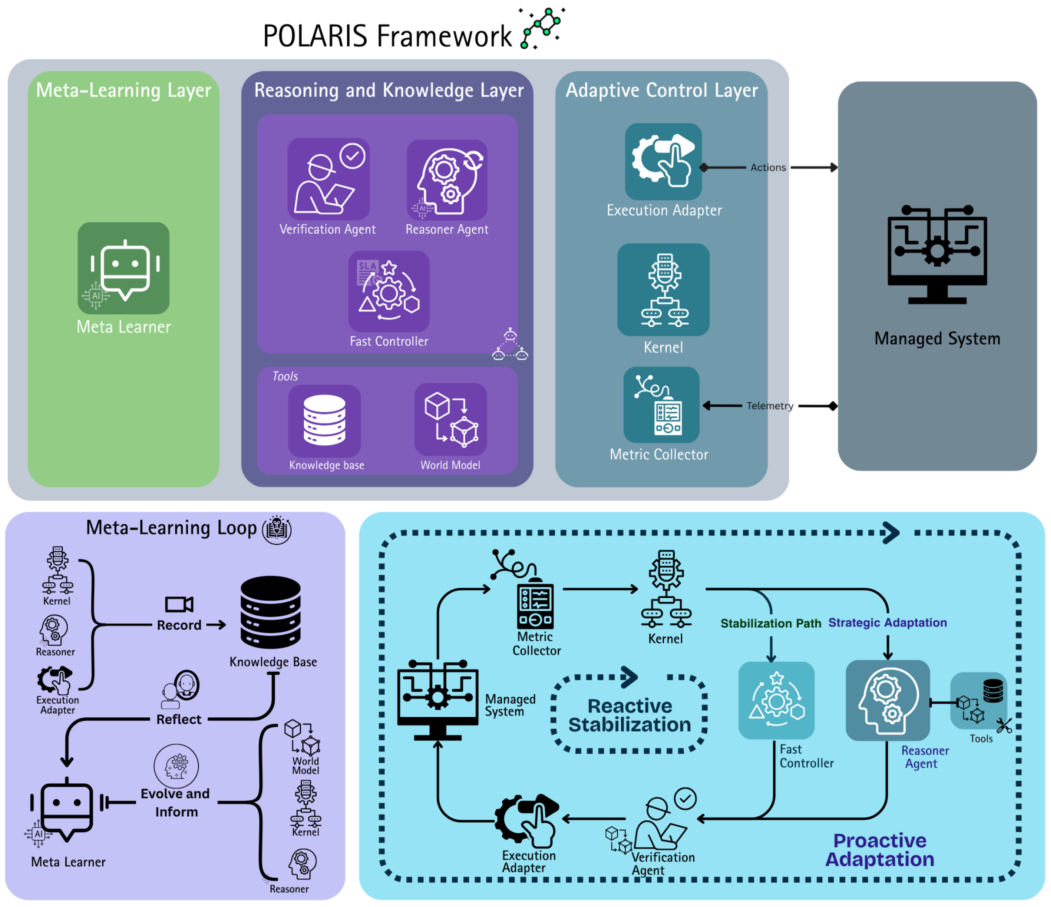

Figure 1: POLARIS Framework: The three layers along with adaptation loops visualized

and invariants are mapped into a consistent reasoning process, ensuring interpretable and reproducible decisions. Formally, $\mathcal{P}_{conf} = \langle \mathcal{I}, \mathcal{D}, \mathcal{R}, \mathcal{O} \rangle$, where $\mathcal{I}$ represents fixed invariants, $\mathcal{D}$ defines key operational directives and performance targets, $\mathcal{R}$ outlines reasoning procedures (e.g., Chain of Thought Prompting, Tool call definitions), and $\mathcal{O}$ specifies structured output formats. For instance, $\mathcal{D}$ might define a dangerous utilization level (`server_util_danger` $= 0.90$) or a target response time (`response_time_goal` $= 0.5s$), which the LLM uses to inform its planning. Through $\mathcal{P}_{conf}$, prompts are dynamically instantiated from context P_R, maintaining consistent logic across adaptation cycles.

Example Configuration. A skeleton of the prompt for Reasoner

> ***System Role:*** *Proactive controller maintaining SLA (response time $< 1s$) via adaptive dimmer and server scaling.*
>
> ***Goals:*** *Keep dimmer near 1.0, and minimize servers (1–3).*
>
> ***Logic:*** *Analyze trends (use KB tool) $\rightarrow$ simulate(use WM tool) $\rightarrow$ act (adjust dimmer first, scale servers only if persistent overload).*
>
> ***Constraints:*** *Stable actions, valid JSON output, respect cooldowns, and never violate SLA.*

Deliberation via Tools. The Reasoner leverages its tools for evidence-based decision-making. The KB provides a unified access layer to structured and unstructured knowledge such as operational logs, adaptation traces, and historical actions. Any data management or retrieval mechanism may be used, provided it enables queries that return contextually relevant system episodes. The **World Model (WM)** is used for "what-if" simulations through learned performance models, simulators, or domain specific models etc.

The Reasoner's deliberation is guided by two query formalisms that interface with its tools:

$$q_{KB} = (d_{req}, \lambda) \longrightarrow D_{hist} \quad \text{and} \quad q_{WM} = (\phi, \psi) \longrightarrow f_{sys}$$

A query to the Knowledge Base, q_{KB}, is composed of a semantic filter d_{req} and a set of structured key-value constraints λ, returning D_{hist} as a collection of relevant historical event tuples. In our prototype, this query is executed against an in-memory collection of Python objects. Similarly, a query to the World Model, q_{WM}, simulates the effect of a proposed action vector ϕ on the current system state ψ. Implemented in example as an inference call to a Bayesian model, it returns f_{sys}, a probability distribution over the predicted next state.

The Reasoner integrates the historical data (D_{hist}) and the predictive model (f_{sys}) to deliberate over potential actions.

In our example, once condition is stabilized, the Kernel puts the Reasoner back in control whose goal is to restore high fidelity (dimmer=1.0). Reasoner then queries the KB to analyze the recent crisis (the 900ms spike) and confirms it was due to high server load and goes to formulates the hypothesis that adding a server might create enough capacity to restore the dimmer. To test this, it queries the World Model with the candidate action $a^* = \texttt{add\_server}$. The WM simulates this action against the current system state, and its prediction, f_{sys}, indicates that response time would remain safely below the 750ms threshold even with the dimmer restored to 1.0.

Decision and Verification. The Reasoner formulates a candidate adaptation directive a^*. This is sent to the **Verification Agent (V)**, which acts as a safety backstop, validating the plan against all invariants/constraints $\mathcal{I}$:

$$V(a^*, \mathcal{I}) \longrightarrow \{\text{accept}, \text{reject}\}$$

For instance, V can be realized as: (a) a lightweight checker for static invariants (e.g., $\texttt{max\_servers} \leq 3$), (b) a runtime verification monitor evaluating plans against temporal logic specifications (e.g., LTL rules to prevent undesirable oscillations), or (c) an interface to a formal model checker validating the plan against a formal system model[24, 25]. In our current evaluation, we employ approach (a).

As shown in Figure 1, reasoning and stabilization paths converge here, and only upon acceptance does the directive proceed to the *EA* for execution.

5.4 Meta-Learning

The outermost loop in Figure 1 enables the system to learn from its own adaptive behavior to become more effective over time.

Record (Experience Capture). After every adaptation cycle, an experience tuple ϵ is systematically recorded and stored in the *KB*:

$$\epsilon = (\epsilon_{ctx}, \epsilon_{dec}, \epsilon_{out})$$

In our running example, this process captures the full sequence of events. First, the reactive cycle is recorded (ϵ_1): capturing the critical state (ϵ_{ctx}), Fast Controller's `reduce_dimmer` action (ϵ_{dec}), and the immediate stabilizing outcome (ϵ_{out}). Subsequently, the proactive cycle is recorded (ϵ_2), capturing the follow-up system context, the Reasoner's `add_server` decision, and the final optimized outcome.

Reflect and Evolve. Periodically, the **Meta-Learner Agent (M)** analyzes the accumulated experiences $\{\epsilon_i, \ldots\}$ from the *KB* to identify statistically significant patterns (P) and then derives an improved strategy (S'):

$$f_{reflect} : KB \longrightarrow P \quad \text{and} \quad f_{evolve} : P \longrightarrow S'$$

This new strategy S' is disseminated by updating policies in the Reasoner (modifying it's prompt template P_{conf}), tuning thresholds in the Kernel, or enhancing parameters in the World Model. This continuous self-refinement embodies the shift to *Self-Adaptation 3.0*. For instance, after the experiences (high utilization $\longrightarrow F$ acts $\longrightarrow R$ acts), $f_{reflect}$ identifies a statistically significant pattern P: "when server utilization approaches 80%, the Fast Controller is consistently triggered, followed by an action to add a server." Through $f_{evolve}(P) \longrightarrow S'$, the Meta-Learner derives an improved strategy S': it lowers the `server_util_danger` parameter in $O \subset \mathcal{P}_{conf}$ from 0.90 to 0.80. This update allows the Reasoner to anticipate this condition and proactively `add_server`, preventing the SLA breach and the need for reactive intervention.

Prompt Configuration. Since the Meta-Learner is implemented using an LLM agent, its reflective and evolutionary behavior is governed through a structured *prompt configuration* that formalizes how experiences are interpreted and transformed into improved strategies. This configuration defines the reflection workflow, evolution logic, and update dissemination, ensuring consistent meta-level reasoning across learning cycles. Formally,

$$\mathcal{M}_{conf} = \langle \mathcal{R}_m, \mathcal{E}_m, \mathcal{U}_m \rangle$$

where $\mathcal{R}_m$ represents reflection procedures (e.g., extract trends $\longrightarrow$ correlate patterns $\longrightarrow$ summarize insights), $\mathcal{E}_m$ defines evolution mappings from patterns to refined strategies, and $\mathcal{U}_m$ governs how updates propagate across the system. Through $\mathcal{M}_{conf}$, the Meta-Learner achieves structured and interpretable continual improvement.

Example Prompt. A skeleton of the prompt for Meta-Learner

You are an expert systems engineer specializing in adaptive optimization.

*Decide whether to **store new observations**, **tune thresholds**, or **refine templates** based on system evidence.*

Modify only when patterns or parameters clearly justify it; otherwise, return an empty update.

All outputs must be valid JSON with concise justifications.

The meta-learner has access to tools to enforce the required changes and query the current reasoner prompt, knowledge base etc. The Meta-Learner accesses its tools via bidirectional queries $q_{ML} = (\eta, \sigma) \leftrightarrow \Delta_S$, enabling both retrieval and update of knowledge, configurations, prompt template and thresholds across the system.

6 Experimental details

Exemplars We evaluate our framework using two diverse self-adaptive exemplar systems: *SWIM*[36] and *SWITCH*[29]. *SWIM* is a web application simulator for resource management under traffic uncertainty, emphasizing elasticity and dimmer control. It monitors response time and server utilization to adapt by adding/removing servers (with a 60s provisioning delay) or adjusting the *dimmer value*, the probability of returning optional content (0–1 range). *SWITCH* is a self-adaptive exemplar for ML-enabled system (MLS). It enables self-adaptation by dynamically switching between machine learning models to balance latency and accuracy. In the evaluated object detection scenario, it selects among vision models based on confidence, size, CPU usage, and response time. This diversity provides a strong test of our framework's generalizability.

LLM selection To test POLARIS, for the Reasoner, we selected a diverse set of LLMs to balance reasoning quality, latency, and cost. We used high-performance proprietary models [8] via API (GPT-5 and Gemini 2.5 Pro for reasoning, Gemini 2.0 Flash for speed). We also used a reasoning based, locally-hosted open-source model (GPT OSS 20B served via Ollama[3]). We used the Gemini 2.0 Flash model for the Meta-learner given the task's simplicity and easy availability of the model.

Implementation POLARIS was implemented in Python. For SWIM, Adapters managed pub-sub communication with telemetry subjects

[3]https://ollama.com/

via a NATS server, while the kernel ran asynchronous monitoring and control loops. Response time was derived from telemetry streams; if it exceeded 0.75 s, a reactive script either launched an extra server (if available) or reduced the dimmer level. The Knowledge Base, built with native Python structures, stored buffered telemetry, aggregated observations, and recent actions for rapid lookup. The reasoning layer used an instantiated LLM (local or API) with Chain-of-Thought prompting [44], linked to the Knowledge Base and a Bayesian World Model for reasoning under uncertainty. Given the agent's simplicity, no framework such as CrewAI or LangGraph was used. Control actions from the LLM were published back to SWIM through the execution adapter via NATS topics. A secondary Gemini Flash 2.0 LLM performed meta-learning by analyzing adaptation logs and telemetry summaries, updating prompt files , recalibrating thresholds, and refining reasoning parameters for improved adaptation (through file writing tool-calls).

Hardware Configuration SWIM and SWITCH experiments ran on Ubuntu systems with 16GB RAM (4.7 GHz Intel i7) and 24GB RAM (4.8 GHz Intel Ultra 7), respectively. The local GPT OSS 20B model was served via Ollama on an NVIDIA A6000 GPU

Experimental Candidates To ensure a comprehensive evaluation, we compared *POLARIS* against a diverse set of baselines across *SWIM* and *SWITCH*. In SWIM, we considered its built-in rule-based approaches, *Reactive1* and *Reactive2*, along with adaptive approaches such as *Thallium* [41], *PLA-SDP* [33], *Cobra* [1], and *PLA* [32]. For SWITCH, we used *AdaMLS* [23] as the primary baseline. Experiments were conducted using the built-in *Clarknet* [3, 9] trace for SWIM and the *General Object Detection* trace for SWITCH on 500 randomly sampled images from the *COCO 2017* dataset [27]. The same Clarknet trace was applied across all SWIM-based baselines for fairness, and selected experiments were repeated three times for statistical reliability (as detailed in RQ1). Based on the results, *Gemini Flash 2.0* was chosen as the Reasoner agent for subsequent ablation studies, having the lowest overall performance among the evaluated LLMs. Ablations were designed to assess the contribution of each framework component, including scenarios without Tools (KB and WM). This does not **restrict the LLM's access to information**; rather, KB and WM outputs are provided deterministically. While this eases context retrieval for the LLM, it lessens its agentic capabilities in generalizable scenarios. We aim to show that even with fully autonomous tool use, the framework can match or exceed the deterministic setting.

Notation for POLARIS Variants For clarity, each POLARIS variant is labeled according to which internal component is inactive. The LLM in the notation refers to the one used for Reasoner. For Meta-learner, we use Gemini Flash 2.0. If a component is absent, it is denoted with a – prefix (e.g., **–M**). For ex: *POLARIS (Gemini Flash, -M)* denotes a variant running on Gemini Flash based Reasoner without the Meta-Learner but with Tools and Fast Controller active. Our replication package for more details: here

7 Results

We define two research questions to evaluate the performance of POLARIS in terms of both effectiveness and efficiency:

RQ1. How does the effectiveness and generalizability of POLARIS compare to existing baseline approaches?

Table 2 presents the results of applying various baselines on the SWIM exemplar. For *Cobra* and *PLA*, we take results from Moreno et al[35] study, with their best performing configuration on Clarknet. From the results, it is evident that POLARIS outperforms the baseline approaches in terms of cumulative utility while maintaining lower response time violations. This improvement can be attributed to the learning-driven adaptation enabled by the interaction between the Reasoner and the Meta-learner. By continuously observing suboptimal decisions that lead to response time spikes, POLARIS refines its decision-making over successive adaptation cycles. As shown in Figure 3, the full configuration exhibits fewer response-time violations due to the meta-learner's ability to leverage past observations and past action feedback from the Knowledge Base, enabling faster and more informed adaptation. Unlike the baselines, which rely on complex, mathematically defined strategies requiring extensive design-time modeling, POLARIS leverages **natural language–based reasoning** to generalize effectively across dynamic and complex environments. The framework's architectural controls mitigate the randomness typical of single LLMs, ensuring consistent behavior across runs. Notably, it delivers comparable performance with different model backends (GPT-5, GPT-OSS:20B, Gemini Pro, Flash), indicating that reliability in LLM-driven systems stems more from design than choosing a specific model.

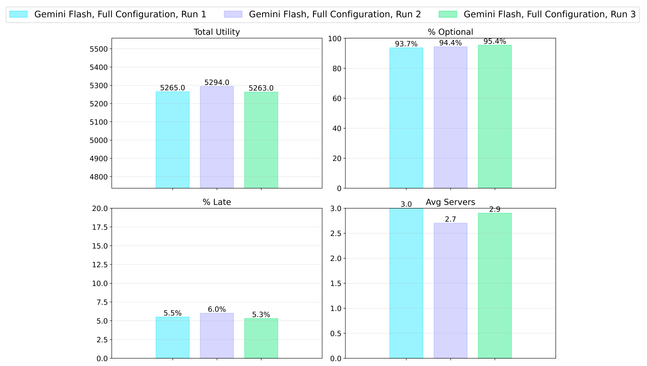

Figure 2: Same configuration of POLARIS across 3 runs show minimal variation

To assess the contribution of individual components, we conducted ablation studies. POLARIS variants without Meta-learner, Fast Controller, or tool integration showed markedly inferior performance, with frequent response time violations. Figure 3 illustrates that incorporating the Meta-learner and Fast Controller significantly improves both cumulative utility and response time stability by enabling proactive and reactive adaptation, respectively. As shown in Figure 4, the Meta-learner continuously updates thresholds and prompt configurations based on system observations, enhancing proactive reasoning over time. While the version without tools may appear to perform slightly better in deterministic settings (since tool outputs are directly provided), our objective is to enable autonomous tool use, a key aspect of POLARIS that ensures scalability and generalizability across diverse systems.

To evaluate the generalizability of POLARIS beyond a single domain, we further conducted experiments on the SWITCH exemplar.

Approach	Optional %	Late %	Avg Servers	Total Utility
Baselines				
Reactive 1	91.07	28.78	2.45	2647.48
Reactive 2	88.33	35.05	2.40	1739.88
Thallium	45.56	0.30	2.00	4658.65
PLA-SDP	15.56	0.00	2.72	4089.09
Cobra	89.00	3.30	3.00	5378.00
PLA	69.10	0.60	2.80	5306.00
POLARIS - Full configurations				
POLARIS (GPT-OSS, Full Configuration)	89.14	5.42	2.77	**5405.98**
POLARIS (GPT-5, Full Configuration)	**95.08**	6.24	2.82	**5445.48**
POLARIS (Gemini Pro, Full Configuration)	85.34	4.86	2.56	5093.49
POLARIS (Gemini Flash, Full Configuration)	94.4	6.0	2.7	5294.00
POLARIS - Ablations				
POLARIS (Gemini Flash, −M)	89.31	18.18	2.40	3903.67
POLARIS (Gemini Flash, −Tools)	90.71	5.69	2.78	5272.51
POLARIS (Gemini Flash, −Tools, −F)	94.77	12.04	2.47	4580.87
POLARIS (Gemini Flash, −Tools, −M)	97.82	13.51	2.53	4846.28
POLARIS (Gemini Flash, −Tools, −M, −F)	97.84	17.95	2.45	4130.68

Table 2: Performance metrics for all evaluated on SWIM with Clarknet Trace. Variants of POLARIS differ by the exclusion (–) of the Meta-Learner (M), Tools (KB + WM), and Fast Controller (F).

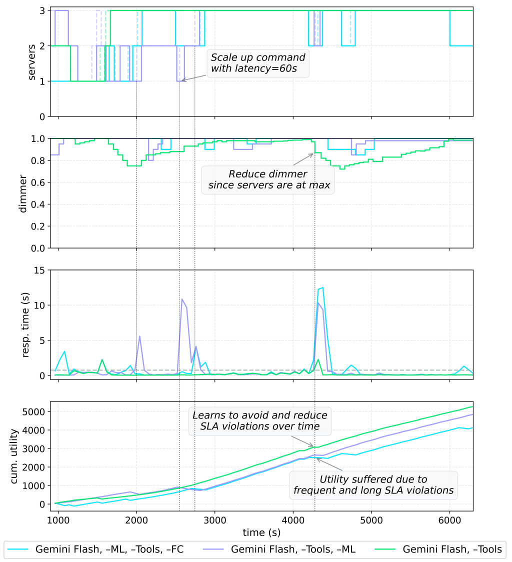

Figure 3: Effect of Fast controller and Meta-Learner in POLARIS

Approach	Confidence Mean	Response Time (s) Median	CPU Usage (%)	Inference Rate / Switches (inf/min)
ADAMLS	0.729	0.132	56.85	212.78/865
POLARIS[†]	0.688	**0.096**	**48.36**	**244.19/112**

[†]Gemini Flash with full configuration.

Table 3: Performance comparison on SWITCH

As summarized in Table 3, POLARIS consistently outperformed ADAMLS across the reported metrics, confirming its ability to generalize effectively across distinct system architectures and workloads. These results demonstrate that POLARIS not only sustains performance under varying operational conditions but also maintains robustness in the presence of uncertainty, highlighting the importance of **language based reasoning for self-adaptation**.

RQ2. How efficient is POLARIS as a framework, and what overheads does it introduce?

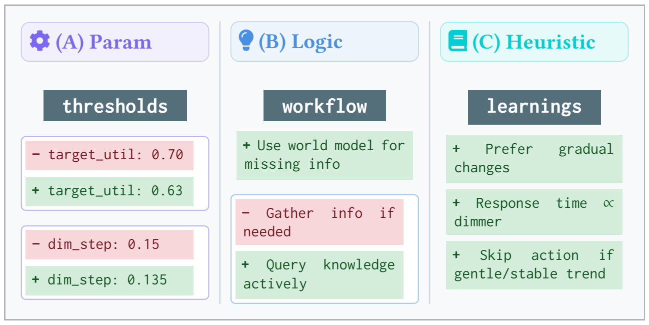

Figure 4: Meta-learner initiated Reasoner prompt evolution: (A) parameter tuning, (B) logic refinement, (C) heuristic synthesis.

Table 4: LLM-wise Latency and Token Usage per Reasoning Decision in POLARIS

LLM Variant	Input Tokens (avg)	Output Tokens (avg)
Gemini Flash 2.0	8222.97	259.66
Gemini 2.5 Pro	13346.32	490.74
GPT-5	17719.02	409.12

Table 5: Performance Efficiency of the POLARIS Framework

Overhead Type	Average Latency (ms)
End-to-End Reasoning	**7569.57**
LLM (Gemini) Call	3048.31
World Model (WM) Operation	11.84
LLM Tool Call Overhead	5.23
Knowledge Base (KB) Query	2.85

In Table 5, we define end-to-end reasoning time as the interval from the first tool call initiated after a decision is generated to the first action generated after that decision cycle. This duration is approximately 7.5 seconds, representing a nominal overhead for the proactive adaptation loop. Since the loop operates at fixed intervals, the impact on overall decision-making is minimal. We argue that the performance and generalizability benefits of POLARIS outweigh this marginal overhead, as any LLM-based reasoning approach inherently incurs inference latency. Beyond the LLM calls, the remaining system components contribute negligible time overheads on the order of milliseconds. Based on Table 4, while the average token usage for the LLMs does imply a notable operational cost, we argue that this is a manageable trade-off. This cost is clearly outweighed by the significant gains in performance and the overall generalizability that *POLARIS* achieves and is a very manageable level effort for building multi-agentic AI native frameworks.

8 Discussion

Our evaluation provides early but compelling evidence that an agentic AI, reasoning-driven approach (what we describe as Self-adaptation 3.0) is both viable and more effective than existing paradigms. This discussion synthesizes our key findings, connects them to the challenges we set out to address, and frames open questions that can guide the community's next steps.

Generalizable Adaptations with Reasoning: The evaluation of *POLARIS* demonstrates that it significantly outperforms established baseline approaches in diverse testing environments (seen in *RQ1*). Unlike traditional self-adaptive systems that require extensive design-time modeling, *POLARIS* generalizes effectively across diverse operational contexts by harnessing the power of LLMs for understanding metrics through natural language, bridging the gap between adaptation and reasoning, addressing the imminent *CH1*.

Learning-Driven Evolution: The learning capability of *POLARIS* is equally central to its success, as shown in Figures 2 and 3. The system not only reacts to conditions but also records, reflects, and avoids repeating ineffective strategies. Ablation studies (described in *RQ1*) highlight that removing the Meta-Learner or Fast Controller leads to markedly inferior performance, reinforcing the importance of continual reflection and learning. This also contributes towards advancing the vision of lifelong self-adaptation [14]. Hence, *POLARIS* takes a tangible step toward realizing *CH2*, demonstrating that adaptation can itself become a learning process.

Coordinated Multi-Agent Intelligence: Beyond individual learning, *POLARIS* provides a structured blueprint for managing complex adaptation tasks. By balancing short-term goals with long-term planning, it coordinates the Fast Controller, Reasoner, and Meta-Learner in complementary roles. For instance, as depicted in Figure 3, the Fast Controller handled rapid response-time spikes, while the Meta-Learner refined thresholds and prompt settings based on prior SLA violations. Removing either one caused a threefold rise in late responses and a drop of over 1,000 in cumulative utility. A similar observation was found for SWITCH (Table 3). This balance between short-term and long-term reasoning is a leap towards addressing the long-recognized challenge of distributed coordination in self-managed systems [21], providing early evidence that multi-agent reasoning can achieve both stability and foresight (*CH3*).

Applying *POLARIS* to manage a system requires only defining adapters for monitoring and execution, along with specifying prompt templates and the Fast Controller policy, making it flexible and easy to integrate while maintaining respectful efficiency, as observed in *RQ2*. Moreover, the LLM-based reasoning layer supports human-in-the-loop mechanisms through explainable decisions. The Reasoner can also be extended to incorporate richer feedback sources, such as document retrievals or advanced mathematical tools, broadening its capability to derive actionable insights.

A Call to the Community: We believe POLARIS is a starting point. It opens a research agenda for agentic self-adaptation, prompting key questions: How can we design cognitive feedback loops that complement MAPE-K to enable reasoning about adaptation quality? What new forms of assurance are needed for probabilistic, language-based reasoning? What architectural patterns best support explainable agentic interactions and effective human-in-the-loop collaboration? And finally, how can we evaluate these systems beyond QoS metrics to assess the quality of their learning and reasoning over time?

9 Threats to validity

Our evaluation is subject to some potential threats. *Internal validity* concerns arise from the use of simulated environments; while these are well-established self-adaptive exemplars, they may not fully capture the complexities of real-world systems. Moreover, since the framework relies on LLM-based reasoning, some degree of non-determinism is inherent. Our experimental results in 2 show remarkably consistent performance across multiple runs. This suggests that the architectural guardrails, structured prompting, and reflective learning within POLARIS effectively mitigate this risk, leading to reliable and deterministic system-level behavior and further efforts can be put in this direction. *Construct validity* threats stem from the sensitivity of decisions to prompt design, which depends on the developer constructing the managing system. Additionally, tool usage must be explicitly defined, and integrating new tools requires the model to properly interpret tool-calling conventions. Finally, *external validity* is affected by implementation constraints: the Knowledge Base is currently maintained in-memory, and the World Model outputs are implementation-specific. Persisting the Knowledge Base and refining the modeling approach could further strengthen generalization and reasoning robustness.

10 Conclusion

We presented POLARIS, a framework designed to enable self-adaptive systems to reason about and continuously improve their own adaptation strategies. POLARIS lays the foundation for the next wave of self-adaptive software systems by addressing the uncertainties and evolving challenges of modern software environments. Its layered architecture integrates reasoning, learning, and distributed intelligence to enable adaptive, coordinated decision-making. Experimental evaluations on established exemplars provide early evidence that POLARIS achieves superior performance and generalizability. As self-adaptive systems become more AI-native, we believe that integrating reasoning and learning will be central to how such systems are designed, analyzed, and trusted. POLARIS takes an initial step in this direction, encouraging the community to explore how adaptive systems can move beyond reacting to change, to understanding and improving how they adapt over time.

References

[1] Konstantinos Angelopoulos, Alessandro V. Papadopoulos, Vítor E. Silva Souza, and John Mylopoulos. 2016. Model predictive control for software systems with CobRA. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (Austin, Texas) *(SEAMS '16)*.

Association for Computing Machinery, New York, NY, USA, 35–46. doi:10.1145/2897053.2897054

[2] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. 2015. Modeling and analyzing MAPE-K feedback loops for self-adaptation. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (Florence, Italy) *(SEAMS '15)*. IEEE Press, 13–23.

[3] Martin F. Arlitt and Carey L. Williamson. 1996. Web server workload characterization: the search for invariants. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (Philadelphia, Pennsylvania, USA) *(SIGMETRICS '96)*. Association for Computing Machinery, New York, NY, USA, 126–137. doi:10.1145/233013.233034

[4] Tomáš Bureš. 2021. Self-Adaptation 2.0. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 262–263. doi:10.1109/SEAMS51251.2021.00046

[5] Ricardo Diniz Caldas, Arthur Rodrigues, Eric Bernd Gil, Genaína Nunes Rodrigues, Thomas Vogel, and Patrizio Pelliccione. 2020. A hybrid approach combining control theory and AI for engineering self-adaptive systems. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (Seoul, Republic of Korea) *(SEAMS '20)*. Association for Computing Machinery, New York, NY, USA, 9–19. doi:10.1145/3387939.3391595

[6] Radu Calinescu, Carlo Ghezzi, Marta Kwiatkowska, and Raffaela Mirandola. 2012. Self-Adaptive Software Needs Quantitative Verification at Runtime. *Commun. ACM* 55 (09 2012), 69–77. doi:10.1145/2330667.2330686

[7] Maria Casimiro, Diogo Soares, David Garlan, Luís Rodrigues, and Paolo Romano. 2024. Self-adapting Machine Learning-based Systems via a Probabilistic Model Checking Framework. *ACM Trans. Auton. Adapt. Syst.* 19, 3, Article 18 (Sept. 2024), 30 pages. doi:10.1145/3648682

[8] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. arXiv:2403.04132 [cs.AI] https://arxiv.org/abs/2403.04132

[9] ClarkNet. 1995. ClarkNet Web Server HTTP Trace, August 1995. https://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html. Accessed: 2025-10-24.

[10] Javier Cámara, Henry Muccini, and Karthik Vaidhyanathan. 2020. Quantitative Verification-Aided Machine Learning: A Tandem Approach for Architecting Self-Adaptive IoT Systems. doi:10.1109/ICSA47634.2020.00010

[11] Raghav Donakanti, Prakhar Jain, Shubham Kulkarni, and Karthik Vaidhyanathan. 2024. Reimagining Self-Adaptation in the Age of Large Language Models. 171–174. doi:10.1109/ICSA-C63560.2024.00036

[12] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. 2011. Taming uncertainty in self-adaptive software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (Szeged, Hungary) *(ESEC/FSE '11)*. Association for Computing Machinery, New York, NY, USA, 234–244. doi:10.1145/2025113.2025147

[13] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. 2004. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer* 37, 10 (2004), 46–54. doi:10.1109/MC.2004.175

[14] Omid Gheibi and Danny Weyns. 2022. Lifelong self-adaptation: self-adaptation meets lifelong machine learning. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems* (Pittsburgh, Pennsylvania) *(SEAMS '22)*. Association for Computing Machinery, New York, NY, USA, 1–12. doi:10.1145/3524844.3528052

[15] Omid Gheibi, Danny Weyns, and Federico Quin. 2021. Applying Machine Learning in Self-adaptive Systems: A Systematic Literature Review. *ACM Trans. Auton. Adapt. Syst.* 15, 3, Article 9 (Aug. 2021), 37 pages. doi:10.1145/3469440

[16] Junda He, Christoph Treude, and David Lo. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–30.

[17] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2025. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *ACM Trans. Inf. Syst.* 43, 2, Article 42 (Jan. 2025), 55 pages. doi:10.1145/3703155

[18] J.O. Kephart and D.M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50. doi:10.1109/MC.2003.1160055

[19] Dongsun Kim and Sooyong Park. 2009. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. 76–85. doi:10.1109/SEAMS.2009.5069076

[20] Cody Kinneer, David Garlan, and Claire Le Goues. 2021. Information Reuse and Stochastic Search: Managing Uncertainty in Self-* Systems. *ACM Trans. Auton. Adapt. Syst.* 15, 1, Article 3 (Feb. 2021), 36 pages. doi:10.1145/3440119

[21] Jeff Kramer and Jeff Magee. 2007. Self-Managed Systems: an Architectural Challenge. In *2007 Future of Software Engineering (FOSE '07)*. IEEE Computer Society, USA, 259–268. doi:10.1109/FOSE.2007.19

[22] Jeff Kramer and Jeff Magee. 2009. A Rigorous Architectural Approach to Adaptive Software Engineering. *J. Comput. Sci. Technol.* 24 (03 2009), 183–188. doi:10.1007/s11390-009-9216-5

[23] Shubham Kulkarni, Arya Marda, and Karthik Vaidhyanathan. 2023. Towards Self-Adaptive Machine Learning-Enabled Systems Through QoS-Aware Model Switching. 1721–1725. doi:10.1109/ASE56229.2023.00172

[24] Marta Kwiatkowska, Gethin Norman, and David Parker. 2025. Probabilistic Model Checking: Applications and Trends. In *Principles of Formal Quantitative Analysis (LNCS, Vol. 15760)*. Springer, 158–173.

[25] Martin Leucker and Christian Schallhart. 2009. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming* 78, 5 (2009), 293–303. doi:10.1016/j.jlap.2008.08.004 The 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'07).

[26] Jialong Li, Mingyue Zhang, Nianyu Li, Danny Weyns, Zhi Jin, and Kenji Tei. 2024. Generative AI for Self-Adaptive Systems: State of the Art and Research Roadmap. *ACM Trans. Auton. Adapt. Syst.* 19, 3, Article 13 (Sept. 2024), 60 pages. doi:10.1145/3686803

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Zitnick. 2014. Microsoft COCO: Common Objects in Context, Vol. 8693. doi:10.1007/978-3-319-10602-1_48

[28] Yue Liu, Sin Kit Lo, Qinghua Lu, Liming Zhu, Dehai Zhao, Xiwei Xu, Stefan Harrer, and Jon Whittle. 2025. Agent design pattern catalogue: A collection of architectural patterns for foundation model based agents. *Journal of Systems and Software* 220 (2025), 112278.

[29] Arya Marda, Shubham Kulkarni, and Karthik Vaidhyanathan. 2024. SWITCH: An Exemplar for Evaluating Self-Adaptive ML-Enabled Systems. In *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (Lisbon, AA, Portugal) *(SEAMS '24)*. Association for Computing Machinery, New York, NY, USA, 143–149. doi:10.1145/3643915.3644105

[30] Andreas Metzger, Clément Quinton, Zoltán Ádám Mann, Luciano Baresi, and Klaus Pohl. 2022. Realizing self-adaptive systems via online reinforcement learning and feature-model-guided exploration. *Computing* 106, 4 (March 2022), 1251–1272. doi:10.1007/s00607-022-01052-x

[31] Gabriel A. Moreno, Javier Cámara, David Garlan, and Mark Klein. 2018. Uncertainty reduction in self-adaptive systems. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems* (Gothenburg, Sweden) *(SEAMS '18)*. Association for Computing Machinery, New York, NY, USA, 51–57. doi:10.1145/3194133.3194144

[32] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2015. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 1–12. doi:10.1145/2786805.2786853

[33] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2018. Flexible and Efficient Decision-Making for Proactive Latency-Aware Self-Adaptation. *ACM Trans. Auton. Adapt. Syst.* 13, 1, Article 3 (April 2018), 36 pages. doi:10.1145/3149180

[34] Gabriel A. Moreno, Alessandro Vittorio Papadopoulos, Konstantinos Angelopoulos, Javier Cámara, and Bradley Schmerl. 2017. Comparing Model-Based Predictive Approaches to Self-Adaptation: CobRA and PLA. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 42–53. doi:10.1109/SEAMS.2017.2

[35] Gabriel A. Moreno, Alessandro Vittorio Papadopoulos, Konstantinos Angelopoulos, Javier Cámara, and Bradley Schmerl. 2017. Comparing Model-Based Predictive Approaches to Self-Adaptation: CobRA and PLA. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 42–53. doi:10.1109/SEAMS.2017.2

[36] Gabriel A. Moreno, Bradley Schmerl, and David Garlan. 2018. SWIM: an exemplar for evaluation and comparison of self-adaptation approaches for web applications. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems* (Gothenburg, Sweden) *(SEAMS '18)*. Association for Computing Machinery, New York, NY, USA, 137–143. doi:10.1145/3194133.3194163

[37] Hiroyuki Nakagawa and Shinichi Honiden. 2023. MAPE-K Loop-based Goal Model Generation Using Generative AI. *Proc. of the IEEE 31st International Requirements Engineering Conference Workshops (REW 2023)* (9 2023), 247–251.

[38] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Guoliang Li, Zhiyuan Liu, and Maosong Sun. 2024. Tool Learning with Foundation Models. *ACM Comput. Surv.* 57, 4, Article 101 (Dec. 2024), 40 pages. doi:10.1145/3704435

[39] Brell Peclard Sanwouo Chekam, Clément Quinton, and Paul Temple. 2025. *Breaking the Loop: AWARE is the New MAPE-K.* Association for Computing Machinery,

New York, NY, USA, 626–630. https://doi.org/10.1145/3696630.3728512

[40] Ranjan Sapkota, Konstantinos I Roumeliotis, and Manoj Karkee. 2025. Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. *arXiv preprint arXiv:2505.10468* (2025).

[41] Clay Stevens and Hamid Bagheri. 2020. Reducing run-time adaptation space via analysis of possible utility bounds. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1522–1534. doi:10.1145/3377811.3380365

[42] Karthik Vaidhyanathan and Henry Muccini. 2025. Software Architecture in the Age of Agentic AI. In *European Conference on Software Architecture*. Springer, 41–49.

[43] Yanlin Wang, Wanjun Zhong, Yanxian Huang, Ensheng Shi, Min Yang, Jiachi Chen, Hui Li, Yuchi Ma, Qianxiang Wang, and Zibin Zheng. 2025. Agents in software engineering: Survey, landscape, and vision. *Automated Software Engineering* 32, 2 (2025), 1–36.

[44] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1800, 14 pages.

[45] Danny Weyns. 2018. Engineering self-adaptive software systems–an organized tour. In *2018 ieee 3rd international workshops on foundations and applications of self* systems (fas* w)*. IEEE, 1–2.

[46] Danny Weyns. 2021. *An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective.* doi:10.1002/9781119574910

[47] Michael Wooldridge. 2009. *An Introduction to MultiAgent Systems*. John Wiley & Sons.

[48] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2025. The Rise and Potential of Large Language Model-Based Agents: A Survey. *Science China Information Sciences* 68, 2 (2025), 121101.