Adaptation of Agentic AI: A Survey of Post-Training, Memory, and Skills

Pengcheng Jiang[1*], Jiacheng Lin[1*], Zhiyi Shi[1,4*], Zifeng Wang[13], Luxi He[3], Yichen Wu[4], Ming Zhong[1], Peiyang Song[5,6], Qizheng Zhang[2], Heng Wang[1], Xueqiang Xu[1], Hanwen Xu[7], Pengrui Han[1], Dylan Zhang[1], Jiashuo Sun[1], Chaoqi Yang[1], Kun Qian[14], Tian Wang[14], Changran Hu[5], Manling Li[10], Quanzheng Li[4,12], Hao Peng[1], Sheng Wang[7], Jingbo Shang[8], Chao Zhang[9], Jiaxuan You[1], Liyuan Liu[1], Pan Lu[2], Yu Zhang[11], Heng Ji[1], Yejin Choi[2], Dawn Song[5], Jimeng Sun[1,13], Jiawei Han[1†]

[1] UIUC [2] Stanford [3] Princeton [4] Harvard [5] UC Berkeley [6] Caltech [7] UW
[8] UCSD [9] Georgia Tech [10] Northwestern [11] TAMU [12] MGH [13] Keiji AI [14] Unity

Large language model (LLM) agents are moving beyond prompting alone. ChatGPT marked the rise of general-purpose LLM assistants, DeepSeek showed that on-policy reinforcement learning with verifiable rewards can improve reasoning and tool use, and OpenClaw highlights a newer direction in which agents accumulate persistent memory and reusable skills. Yet the research landscape remains fragmented across post-training, retrieval, memory, and skill systems. This survey studies these developments under a single notion of *adaptation*: improving an agent, its tools, or their interaction after pretraining. We organize the field with a four-paradigm framework spanning agent adaptation and tool adaptation. On the agent side, A1 (tool-execution-signaled) and A2 (agent-output-signaled) improve the agent itself through supervised fine-tuning, preference optimization, and reinforcement learning with verifiable rewards. On the tool side, T1 (agent-agnostic) provides reusable pre-trained modules any agent can call, while T2 (agent-supervised) uses the agent's outputs to train memory systems, skill libraries, or lightweight subagents. Using this framework, we review post-training methods, adaptive memory architectures, and agent skills; compare their trade-offs in cost, flexibility, and generalization; and summarize evaluation practices across deep research, software development, computer use, and drug discovery. We conclude by outlining open problems in agent-tool co-adaptation, continual learning, safety, and efficient deployment.

Github Repository: https://github.com/pat-jj/Awesome-Adaptation-of-Agentic-AI

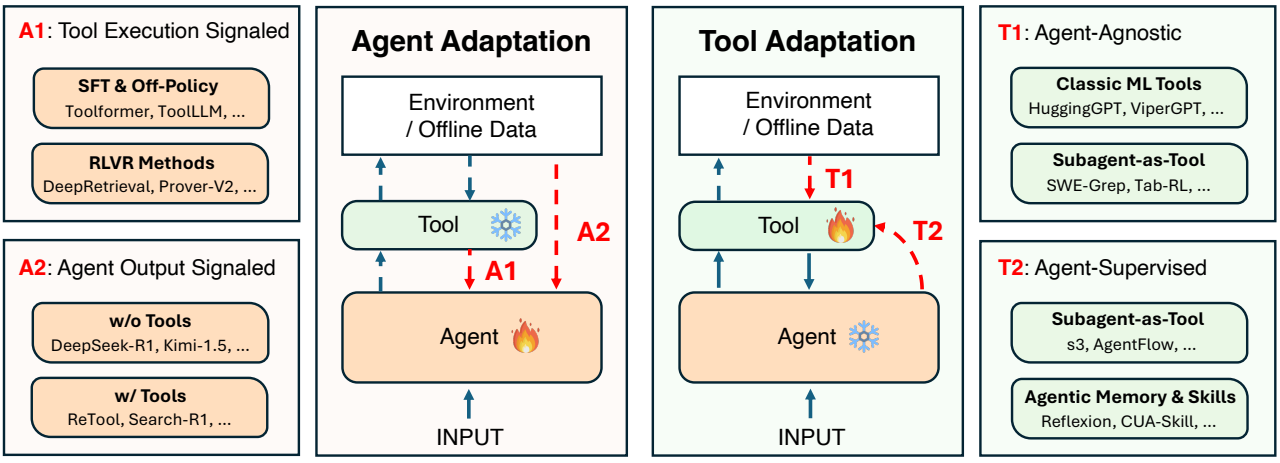

Figure 1 Overview of adaptations in agentic AI. *Agent*: the foundation models serving as orchestration and reasoning modules; *Tool*: callable components other than the agent model that operate independently, e.g., APIs, ML models, subagents, or memory. We categorize these adaptations into two: agent adaptation (**A1** & **A2**): adapting agent models, and tool adaptation (**T1** & **T2**): adapting tools for agents. See more details in §3.

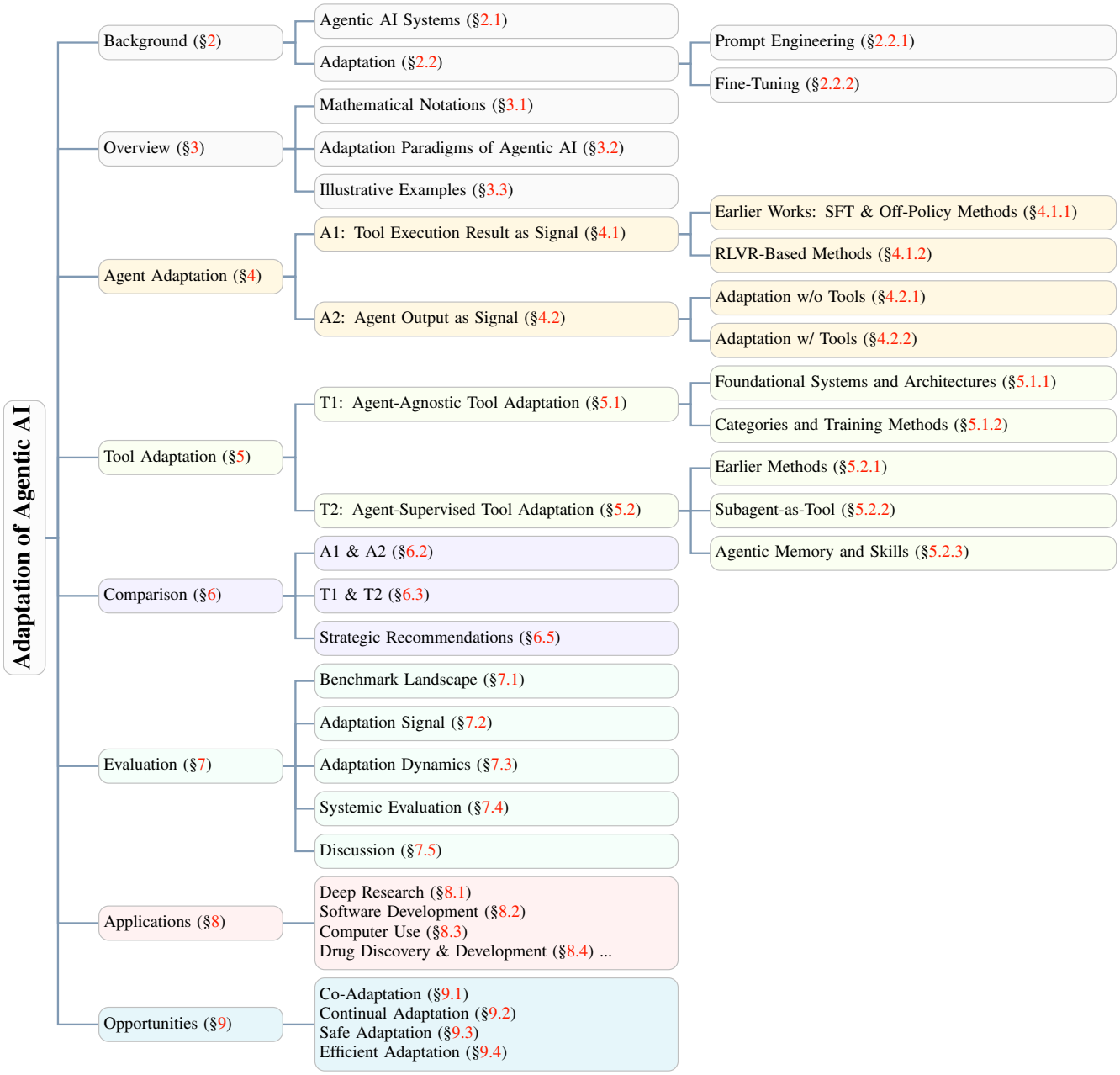

Figure 2 The structure of this paper.

1 Introduction

The rapid progress of foundation models, large language models (LLMs) in particular, has enabled a new class of agentic AI systems that perceive their environment, invoke external tools, manage memory, and execute multi-step plans to complete complex tasks [1–4]. These systems show strong potential across scientific discovery [5, 6], software development, and clinical research [7–9]. Yet current agentic systems remain limited by unreliable tool use, shallow long-horizon planning, domain-specific reasoning gaps, and poor generalization to environments where the agent lacks prior interaction experience [10–13]. Even the most capable foundation models require additional *adaptation* to specialize for particular tasks or deployment scenarios.

Three mechanisms drive this adaptation process. First, **post-training** (supervised fine-tuning, reinforcement learning with verifiable rewards, preference optimization) modifies the agent's parameters to improve reasoning and tool use. Second, **memory** systems (episodic buffers, reflective databases, structured knowledge graphs) allow agents to

retain and build upon past experience without retraining the core model. Third, **skills**—reusable units of procedural knowledge encoding *how* to perform tasks [14]—accumulate through diverse mechanisms: agents internalize tool-use procedures via post-training, while external skill libraries support discovery, invocation, and refinement across sessions. These three mechanisms interact within a shared design space that this survey maps.

Unlike existing surveys on AI agents [1, 15–19], we focus specifically on adaptation. We introduce a unified framework that organizes agentic adaptation into four paradigms spanning both agent and tool adaptation (Figure 1), exposing trade-offs that guide paradigm selection based on supervision signals, task requirements, and system-level constraints.

Our framework organizes adaptation along two dimensions according to which component is optimized (§3). **Agent Adaptation** modifies the agent's internal parameters, representations, or behavioral policies to meet task requirements, through fine-tuning [20] or reinforcement learning with environment feedback [21, 22]. **Tool Adaptation** shifts the optimization target to the agent's external tools (retrievers, planners, memory modules, specialized models), enabling frozen agents to benefit from an adaptive operational environment [23, 11, 24]. Within these two dimensions, we identify four distinct adaptation strategies:

- **A1: Tool Execution Signaled Agent Adaptation** (§3.2.1, §4.1): The agent is optimized using verifiable outcomes produced by external tools it invokes. This paradigm captures settings where correctness signals arise directly from tool execution, such as code sandbox results, retrieval relevance scores, or API call outcomes.
- **A2: Agent Output Signaled Agent Adaptation** (§3.2.2, §4.2): The agent is optimized using evaluations of its own outputs, e.g., final answers, plans, or reasoning traces, possibly after incorporating tool results. This paradigm includes both tool-free outcome-based learning and tool-augmented adaptation driven by answer correctness or preference scores.
- **T1: Agent-Agnostic Tool Adaptation** (§3.2.3, §5.1): Tools are trained independently of the frozen agent. These tools include retrievers, domain-specific models, and other pre-trained components that can be used as plug-and-play modules orchestrated by the frozen agent.
- **T2: Agent-Supervised Tool Adaptation** (§3.2.4, §5.2): The agent remains fixed while its tools are adapted using signals derived from the agent's outputs. This paradigm includes reward-driven retriever tuning, adaptive rerankers, search subagents, and memory-update modules trained to better support the frozen agent.

These four strategies overlap in practice; we treat them as a working classification. When a method simultaneously modifies both the agent and a tool, we assign it to the paradigm corresponding to its *dominant locus of optimization* and note the secondary component explicitly. Current leading systems combine multiple paradigms. A deep research system may employ T1-style retrieval tools (pre-trained dense retrievers), T2-style adaptive search agents (trained via frozen LLM feedback), and A1-style reasoning agents (fine-tuned with execution feedback) in a cascaded architecture [25–27, 6].

The choice among these paradigms involves trade-offs along several dimensions (§6). (1) **Cost and flexibility**: Agent adaptation (A1/A2) requires substantial computational resources for training billion-parameter models but offers broad flexibility, while tool adaptation (T1/T2) optimizes external components at lower cost but is constrained by the frozen agent's capabilities [28, 23]. (2) **Generalization**: T1 tools trained on broad data distributions generalize well across agents and tasks [24, 29], whereas A1 methods may overfit to specific environments unless carefully regularized [21]. (3) **Modularity**: T2 approaches enable independent tool upgrades without agent retraining [30, 23], while A1/A2 methods risk catastrophic forgetting when adapted to new tasks.

Scope and contributions. By grounding the four-paradigm taxonomy in three concrete mechanisms—post-training, memory, and skills—we connect the abstract framework to the techniques that practitioners deploy in practice. Our key contributions are:

- A **unified** 2×2 **framework** (Figures 1, 3, 7) that decomposes agentic adaptation along two axes—*what is adapted* (agent vs. tool) and *how the signal is obtained* (execution-grounded vs. output-evaluated)—yielding four paradigms (A1/A2/T1/T2) with distinct cost, flexibility, and generalization profiles, together

with **cross-paradigm empirical synthesis** (§6) showing, e.g., that T2 tool adaptation can match A2 agent training accuracy with far fewer examples in retrieval settings, and that A1-trained agents can be frozen and redeployed as T1 tools.

- A **systematic treatment of agentic memory and skills** as adaptation mechanisms. We show how adaptive memory systems (episodic buffers, reflective databases, knowledge graphs) instantiate T2 adaptation, and how skill libraries bridge A1/A2 post-training with T1/T2 tool ecosystems (§5).
- A **multi-dimensional evaluation framework** (§7) that maps benchmarks to paradigms, distinguishes verifiable execution metrics from holistic utility metrics, and identifies gaps in current evaluation practice, particularly for T2 and cross-paradigm comparison.
- **Domain-specific paradigm mapping** across deep research, software development, computer use, and drug discovery (§8), showing how the dominant adaptation paradigm varies with the availability of verifiable feedback and the cost of agent retraining.
- Identification of **open challenges** including agent-tool co-adaptation, continual adaptation under non-stationary task distributions, safe exploration during on-policy RL, and efficient adaptation for resource-constrained deployment (§9).

Survey methodology and scope. We surveyed the literature on agentic adaptation published through early 2026, drawing primarily from top-tier venues (NeurIPS, ICML, ICLR, ACL, EMNLP, NAACL) and high-impact arXiv preprints. Methods are assigned to paradigms based on their *dominant* locus of optimization and signal source; boundary cases are flagged explicitly. Quantitative cross-paradigm comparisons (e.g., data efficiency of T2 vs. A2) are drawn from case studies that may differ in confounding factors; we note these limitations where they arise.

Organization. Section 2 introduces foundational concepts. Section 3 formalizes the four paradigms with illustrative examples. Sections 4 and 5 review agent adaptation (A1, A2) and tool adaptation (T1, T2) methods, respectively. Section 6 compares paradigms along key dimensions. Section 7 presents a multi-dimensional evaluation framework. Section 8 examines real-world applications with explicit paradigm mapping. Section 9 discusses open challenges, and Section 10 concludes the paper.

2 Background

This section introduces the background necessary for the remainder of the survey. We first describe the core components of *Agentic AI Systems* (§2.1), then discuss the two primary forms of *Adaptation* (§2.2) through which these systems specialize for particular tasks or deployment scenarios.

2.1 Agentic AI Systems

An agentic AI system combines a foundation model with planning, tool use, and memory modules to autonomously execute multi-step tasks, using environmental feedback to refine its behavior over time. This survey focuses on *single-agent systems*, which provide a controlled yet expressive setting to study how an individual agent perceives, plans, and acts. Understanding single-agent adaptation is a prerequisite for studying *multi-agent systems*, in which multiple agents coordinate, cooperate, or compete; see [1, 2, 31] for overviews.

At the core of an agentic AI system lies a **foundation model**, typically a large language model (LLM) or multimodal model that serves as the agent's reasoning and control center. Several additional components extend the agent's autonomy:

- **Planning Module:** Decomposes complex goals into actionable steps and organizes their sequential or hierarchical execution. Depending on the degree of feedback integration, planning takes two forms. *Static planning* methods, such as Chain-of-Thought [32] and Tree-of-Thought [33], enable structured reasoning through single-path or multi-path task decomposition. In contrast, *dynamic planning* approaches, such as ReAct [34] and Reflexion [35], incorporate feedback from the environment or past actions, allowing the agent to iteratively refine its plans and improve performance in long-horizon or partially observable scenarios.

- **Tool Use:** Enables the agent to interact with external resources and computational systems, extending its capabilities beyond the limitations of its internal knowledge. Typical tools include web search engines, APIs, code execution environments, Model Context Protocols (MCPs), and browser automation frameworks [36, 3]. The tool-selection policy itself is an adaptation surface: agents must learn which tool to invoke for a given subtask and how to parse the tool's output for downstream reasoning.
- **Memory Module:** Allows the agent to retain, retrieve, and utilize past information for context-aware reasoning and long-term consistency. Memory is typically divided into *short-term memory*, which stores contextual information generated during the current task, and *long-term memory*, which persists across sessions to accumulate reusable knowledge and experience [1, 37]. To access relevant information from long-term memory, many systems employ retrieval-augmented generation (RAG) mechanisms that retrieve and integrate stored knowledge into the agent's reasoning process. Designing an effective memory module involves challenges such as how to structure stored information, when and what to retain, how to retrieve relevant knowledge efficiently, and how to integrate it into ongoing reasoning and decision-making.[1]

2.2 Adaptation

Adaptation modifies the agent or its tools over time to improve performance on specific domains, tasks, or deployment conditions. Without adaptation, even capable foundation models struggle to generalize beyond their pre-training distribution. We distinguish two broad categories: prompt-based adaptation (§2.2.1) and fine-tuning-based adaptation (§2.2.2).

2.2.1 Prompt Engineering

Prompt engineering is a lightweight form of adaptation that guides the behavior of an agentic system without modifying model parameters. The agent's behavior is shaped by carefully crafted input prompts that define goals, constraints, and contextual instructions.

A *prompt* refers to the input context provided to the agent's core model, typically consisting of instructions, examples, or task descriptions that specify the desired behavior. By modifying or composing prompts, an agent can be adapted to new goals or environments without any additional model training, making this approach efficient and transferable across tasks. Such prompt-based adaptation has been widely adopted in recent agentic systems, such as CAMEL [38], AutoGen [39], MetaGPT [40] and ChatDev [41]. For a comprehensive overview of prompt engineering techniques and their design principles, we refer readers to the survey by Sahoo et al. [42].

2.2.2 Fine-Tuning

In contrast, *fine-tuning* adapts the agent by updating the internal parameters of the core model. Training on task-specific data shifts the model's distribution toward domain-appropriate behavior.

Fine-tuning can be performed at different granularities depending on data availability, computational cost, and the desired degree of adaptation. *Full fine-tuning* updates all model parameters using labeled data, providing maximal flexibility but often requiring substantial resources. Alternatively, *parameter-efficient fine-tuning* (PEFT) methods, such as low-rank adaptation (LoRA) [20], update only a small subset of parameters. LoRA, for example, typically matches full fine-tuning quality while updating less than 1% of parameters, making it practical for large agentic systems. For a comprehensive overview of PEFT methods, we refer readers to the survey by Han et al. [43].

Fine-tuning for adapting agents encompasses several major training paradigms. Supervised Fine-Tuning (SFT) [44] performs imitation learning on curated demonstrations. Preference-based methods, such as Direct Preference Optimization (DPO) [45] and its extensions [46], align the model with human or automated preference signals. Reinforcement learning methods such as Proximal Policy Optimization (PPO) [47] and Group Relative Policy

[1] While memory is a fundamental component of an agent, this survey treats most *adaptive memory systems* under the Tool Adaptation paradigm (discussed in §5.2.3). External, non-parametric memory stores (e.g., episodic buffers, reflective databases, skill libraries) that are updated using the frozen agent's outputs are classified as T2; pre-trained memory modules that operate independently of a specific agent fall under T1; and parametric or hybrid memory that modifies model weights occupies the boundary between tool and agent adaptation. A detailed discussion of these distinctions appears in §3.

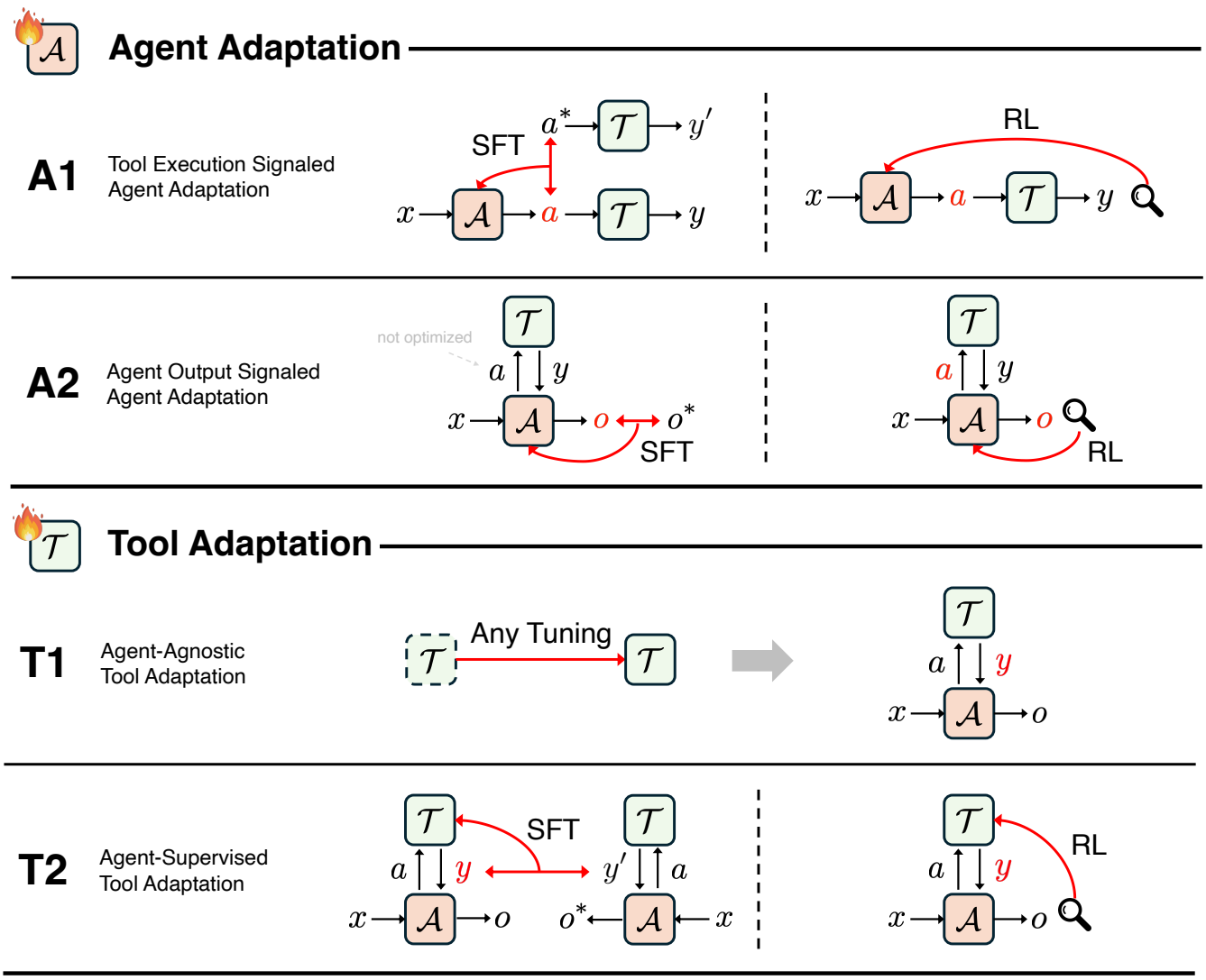

Figure 3 Illustration of Four Adaptation Paradigms (A1, A2, T1, and T2). In all the panels, letters highlighted in Red denote the components directly being optimized during adaptation. The red arrows show the sources of adaptation signals. The dotted black lines separate the cases of supervised fine-tuning (SFT) and reinforcement learning (RL).

Optimization (GRPO) [48] optimize behavior through environment interaction; they require only a reward signal but are harder to stabilize than SFT. For a more comprehensive review of these approaches, see the survey by Zhang et al. [49].

3 Overview of Adaptation Paradigms of Agentic AI

We present the adaptation paradigms that form the analytical basis of this paper. We categorize existing studies on agentic AI systems according to **what is adapted** (the agent or the tool) and **how the adaptation signal is obtained**, yielding four canonical paradigms that capture the major directions of adaptation in recent literature.

3.1 Mathematical Notations

We introduce the key mathematical notations used throughout this paper, organized into three categories: *adaptation targets*, *adaptation data sources*, and *adaptation objectives*.

Adaptation Targets. The entities that undergo adaptation within an agentic AI system are as follows.

- **Agent** ($\mathcal{A}$): The foundation model that serves as the core reasoning and decision-making component of the system, parameterized by θ. Adaptation of the agent can occur through *parameter updates*, *prompt refinement*, or other modifications to its internal policy.
- **Tool** ($\mathcal{T}$): The set of external callable components that extend the agent's capabilities beyond its internal parameters. Tools can include retrievers, planners, executors, simulators, or other computational modules. In this paper, we **treat external memory modules as components of** $\mathcal{T}$ when they function as dynamic, updatable stores that interact with and learn from the agent's outputs. The retrieval process for accessing stored information is typically performed through a dedicated retriever or search tool, which allows the agent to query and integrate relevant past knowledge into its reasoning process. We note that parametric and hybrid memory forms may blur the boundary between $\mathcal{T}$ and $\mathcal{A}$; see the discussion below (§3.2).

Adaptation Data Sources. The sources from which adaptation signals are obtained include the following.

- **Offline Data** ($\mathcal{D}$): Offline data that serve as alignment references or supervision sources for improving either the agent or the tool. These data may include human-labeled demonstrations, synthetic trajectories, or logs of prior interactions.
- **Environment** ($\mathcal{E}$): The external environment in which the agent or tool interacts and receives feedback. It provides online experience signals that reflect task performance or execution quality.

Adaptation Objectives. The objective that guides the adaptation process quantifies performance or alignment quality.

- **Objective Function** $\mathcal{O}(\cdot)$: The objective function optimized during adaptation, which evaluates how effectively the agent-tool system performs according to the designated evaluation protocol. For example, the objective for offline data $\mathcal{D}$ may correspond to supervised or imitation learning losses such as supervised fine-tuning (SFT) or behavior cloning. When adaptation relies on interactions with the environment $\mathcal{E}$, the objective is typically defined by outcome-based metrics such as task success rate.

3.2 Four Adaptation Paradigms of Agentic AI

Building on these notations, we present the four adaptation paradigms proposed in this paper. Adaptation is first categorized by the optimization target: the *agent* or the *tool*. For *agent adaptation*, paradigms are further differentiated by the optimization signal: tool-execution feedback (A1) or evaluations of the agent's final output (A2). For *tool adaptation*, the distinction concerns whether tools are optimized independently of any agent (T1) or adapted under the supervision of a fixed agent (T2).

3.2.1 A1: Tool Execution Signaled Agent Adaptation

In the A1 paradigm, the agent $\mathcal{A}$ is improved through feedback signals derived from the execution results of external tools $\mathcal{T}$, capturing scenarios where tool outcomes serve as a measurable basis for optimization.

Agent-Tool Interaction Process. The agent receives an input x (e.g., a user query or task description) and generates a structured tool call or action $a = \mathcal{A}(x)$, which may include the tool name, arguments, and calling context. The tool set $\mathcal{T}$ then executes this call to produce a result $y = \mathcal{T}(a)$. The pair (a, y) represents a single agent-tool interaction, and the overall process can be summarized as

$$x \xrightarrow{\mathcal{A}} a \xrightarrow{\mathcal{T}} y.$$

The pipeline captures how the agent uses tools to complete tasks. For simplicity and without loss of generality, we describe the interaction using a single tool invocation; multi-turn tool use follows as a direct extension of the formulation.

Optimization Objective. Given this interaction process, the general optimization goal is to adjust the agent $\mathcal{A}$ to generate high-quality tool call action a such that the tool-executed outcomes achieve better performance. Formally,

$$\textbf{(A1)} \quad \mathcal{A}^* = \arg\max_{\mathcal{A}} \mathcal{O}_{\text{tool}}(\mathcal{A}, \mathcal{T}), \tag{1}$$

where $\mathcal{O}_{\text{tool}}$ measures the quality or correctness of the outputs obtained from invoking $\mathcal{T}$, such as tool execution success rate or retrieval scores. This optimization can be instantiated in two primary forms: (1) by imitating collected successful tool-call trajectories, or (2) by generating actions interactively and using the resulting tool feedback to optimize $\mathcal{A}$ via Reinforcement Learning.

- **Supervised Fine-Tuning (SFT).** When explicit target actions are available, the agent learns to imitate successful tool-using behaviors from recorded trajectories without performing online interaction. Let $\mathcal{D}_{\text{succ}} = \{(x, a^*)\}$ denote a dataset of input x and reference action a^* that is known to lead to a correct or desirable tool outcome (y'). The supervised objective is formulated as:

$$\mathcal{A}^* = \arg\min_{\mathcal{A}} \mathbb{E}_{(x,a^*)\sim\mathcal{D}_{\text{succ}}}\big[\ell(\mathcal{A}(x), a^*)\big] \equiv \arg\max_{\mathcal{A}} \mathbb{E}_{(x,a^*)}\big[\log p_{\mathcal{A}}(a^*|x)\big], \tag{2}$$

  where ℓ denotes the cross-entropy loss used for next-token prediction in language models.

- **Reinforcement Learning (RL).** Alternatively, the agent can acquire adaptation signals through interactions with the environment, where it executes tool calls and receives evaluative feedback from the resulting outcomes. The process follows:

$$x \xrightarrow{\mathcal{A}} a \xrightarrow{\mathcal{T}} y, \quad \text{with reward} \quad R = \mathcal{O}_{\text{tool}}(y).$$

  Here, the agent $\mathcal{A}$ generates an action or tool call a based on input x, the tool $\mathcal{T}$ executes a to produce a result y, and the evaluation function $\mathcal{O}_{\text{tool}}$ assigns a scalar feedback R indicating task success or quality. The optimization objective can be expressed as:

$$J(\mathcal{A}) = \mathbb{E}_{x\sim\mathcal{D}_0,\, a\sim\mathcal{A}(\cdot|x),\, y=\mathcal{T}(a)}[\mathcal{O}_{\text{tool}}(y)], \tag{3}$$

  where $\mathcal{D}_0$ denotes the input distribution.

3.2.2 A2: Agent Output Signaled Agent Adaptation

Unlike the A1 paradigm, where the adaptation signal is derived from tool-execution outcomes, the A2 paradigm obtains its optimization signal from the agent's own final output. For simplicity, the following description focuses on a single-turn interaction; the multi-turn case extends naturally.

Agent-Tool Interaction Process. In the A2 paradigm, the agent first generates a tool call a from the input x, the tool $\mathcal{T}$ executes this call and returns an executed result y, and the agent then integrates x and y to produce the final output o:

$$x \xrightarrow{\mathcal{A}} a \xrightarrow{\mathcal{T}} y \xrightarrow{\mathcal{A}} o,$$

where $o = \mathcal{A}(x, a, y)$. This formulation naturally includes the special case where the agent produces o directly without calling any tools.

Optimization Objective. The goal of A2 adaptation is to optimize the agent such that its final output aligns with correctness, quality, or alignment criteria. Formally:

$$\textbf{(A2)} \quad \mathcal{A}^* = \arg\max_{\mathcal{A}} \mathcal{O}_{\text{agent}}(\mathcal{A}, \mathcal{T}), \tag{4}$$

where $\mathcal{O}_{\text{agent}}$ evaluates the final output o generated by the agent. Similarly, A2 paradigm optimization also includes two main forms:

- **Supervised Fine-Tuning (SFT).** Let $\mathcal{D}_{\text{ans}} = \{(x, y, a^*, o^*)\}$ denote a dataset of inputs, optional tool outputs, reference tool calls, and target final outputs. Supervising only the final output o^* is insufficient for learning

tool-use behavior, because the agent could improve final-answer likelihood without ever invoking tools. Effective A2-style SFT therefore combines final-output supervision with A1-style tool-call imitation:

$$\mathcal{A}^* = \arg\max_{\mathcal{A}} \mathbb{E}_{(x,y,a^*,o^*)} \Big[\underbrace{\log p_{\mathcal{A}}(a^*|x)}_{\text{tool-call (A1-style)}} + \underbrace{\log p_{\mathcal{A}}(o^*|x,a^*,y)}_{\text{final answer (A2-style)}} \Big]. \tag{5}$$

When no tools are invoked, a^* and y are empty and the objective reduces to standard answer-level SFT.

- **Reinforcement Learning (RL).** When explicit target outputs are unavailable, the agent learns from feedback assigned to its final response. The interaction follows:

$$x \xrightarrow{\mathcal{A}} a \xrightarrow{\mathcal{T}} y \xrightarrow{\mathcal{A}} o, \quad \text{with reward} \quad R = \mathcal{O}_{\text{agent}}(o).$$

The optimization objective becomes:

$$J(\mathcal{A}) = \mathbb{E}_{x\sim\mathcal{D}_0,\, a\sim\mathcal{A}(\cdot|x),\, y=\mathcal{T}(a),\, o=\mathcal{A}(x,a,y)} \big[\mathcal{O}_{\text{agent}}(o)\big],$$

where $\mathcal{D}_0$ is the distribution of task inputs. Here, the agent receives rewards based solely on the quality of its final output, irrespective of how many intermediate tool calls were invoked.

3.2.3 T1: Agent-Agnostic Tool Adaptation

In the T1 paradigm, the agent $\mathcal{A}$ is kept fixed, and adaptation is applied to only the external tool set $\mathcal{T}$. The setting arises when the agent is a closed-source API (such as GPT, Claude, or Gemini) that cannot be fine-tuned, or when the goal is to augment a fixed agent by training specialized tools such as retrievers, rerankers, planners, simulators, or additional foundation models. In this sense, a "tool" in T1 primarily refers to a trainable model, regardless of whether it is a traditional machine-learning model or a large-scale foundation model.

Optimization Objective. The goal of T1 is to optimize the tool in an agent-agnostic manner:

$$\textbf{(T1)} \quad \mathcal{T}^* = \arg\max_{\mathcal{T}} \mathcal{O}_{\text{tool}}(\mathcal{T}),$$

where $\mathcal{O}_{\text{tool}}(\mathcal{T})$ evaluates the quality of tool-produced results, often through metrics such as retrieval accuracy, ranking quality, simulation fidelity, or downstream task success. Since the agent is fixed and only $\mathcal{T}$ is trainable, T1 reduces to standard model training under various learning paradigms, such as supervised learning, contrastive learning, or reinforcement learning.

3.2.4 T2: Agent-Supervised Tool Adaptation

In the T2 paradigm, tool adaptation is guided by the frozen agent $\mathcal{A}$. Unlike T1, where tools are trained independently, T2 adapts or constructs tools that complement the fixed agent and improve its overall capability. When the main agent is a closed-source foundation model, training auxiliary tools around it is often preferable to modifying the agent itself.

Agent-Tool Interaction Process. As before, we describe a single-turn interaction; the multi-turn case extends naturally. The agent receives input x and produces a tool call $a = \mathcal{A}(x)$. The tool $\mathcal{T}$ executes this call to return a result $y = \mathcal{T}(a)$, and the agent integrates (x, a, y) or (x, y) to produce the final output o:

$$x \xrightarrow{\mathcal{A}} a \xrightarrow{\mathcal{T}} y \xrightarrow{\mathcal{A}} o.$$

Optimization Objective. The tool is optimized to improve the performance of the fixed agent-tool system:

$$\textbf{(T2)} \quad \mathcal{T}^* = \arg\max_{\mathcal{T}} \mathcal{O}_{\text{agent}}(\mathcal{A}, \mathcal{T}),$$

where $\mathcal{O}_{\text{agent}}$ evaluates how effectively the agent performs when equipped with tool $\mathcal{T}$. The objective emphasizes that T2 adapts the tool specifically to the needs of the given agent. Tool adaptation in T2 generally takes two forms:

- **Supervised Learning.** In the supervised setting, the frozen agent provides signals that indicate how the tool should improve. The core idea is to adjust the tool so that its future outputs $\mathcal{T}(a)$ become more helpful for the agent's downstream reasoning. This can be instantiated in several ways. For example:
  - **Quality-Weighted Training.** The agent's final output o induces a quality score $w = \omega(o)$ that reflects the desirability or correctness of the agent's behavior. The tool is trained by weighting each trajectory according to this score:
    $$\mathcal{T}^* = \arg\min_{\mathcal{T}} \mathbb{E}_{(a,y,o)}\Big[w(o)\,\ell\big(\mathcal{T}(a),\, y\big)\Big],$$
    where ℓ is a task-specific loss encouraging the tool's output to improve. If $w(o)$ takes binary values $\{0,1\}$, this reduces to a data-selection scheme where only trajectories associated with desirable agent outputs o are used to train the tool.
  - **Output-Consistency Training.** The agent's final output o induces an implicit supervision target $\tau = \phi(a,y,o)$, which prescribes how the tool output should change to better support the agent. The tool is updated by:
    $$\mathcal{T}^* = \arg\min_{\mathcal{T}} \mathbb{E}_{(a,y,o)}\Big[\ell\big(\mathcal{T}(a),\, \tau\big)\Big],$$
    where the mapping $\phi(a,y,o)$ extracts a learning target from the relationship between y and o. This encourages the tool to produce outputs that more effectively align with the agent's downstream reasoning.
- **Reinforcement Learning (RL).** The tool is updated using a scalar reward based on the final quality of the agent's output. Let $R = \mathcal{O}_{\text{agent}}(o)$ denote the reward assigned to the final output $o = \mathcal{A}(x,a,y)$. The RL objective becomes:
  $$J(\mathcal{T}) = \mathbb{E}_{x\sim\mathcal{D}_0,\, a=\mathcal{A}(x),\, y=\mathcal{T}(a),\, o=\mathcal{A}(x,a,y)}\big[\mathcal{O}_{\text{agent}}(o)\big],$$
  where $\mathcal{D}_0$ is the distribution of task inputs.

Memory and the Adaptation Paradigms. In this paper, the memory module is treated as a tool within $\mathcal{T}$. However, not all memory systems map cleanly onto a single paradigm; the appropriate classification depends on the memory's *form* and *update mechanism*:

- **External adaptive memory (predominantly T2).** When a frozen agent produces outputs that are used to update an external memory store (e.g., episodic buffers, reflective databases, skill libraries) through a fixed or learnable write function $\mathcal{M} \leftarrow \text{Update}(\mathcal{M}, o)$, the process aligns with T2: the agent remains fixed, the adaptation signal originates from the agent's own output, and the memory module evolves to better support future reasoning.
- **Pre-trained or plug-in memory modules (predominantly T1).** Memory components that are trained independently of any specific agent, such as pre-trained dense retrievers, static knowledge bases, or off-the-shelf embedding indices, function as agent-agnostic tools under T1.
- **Parametric and hybrid memory (boundary cases).** Memory encoded within model parameters (e.g., LoRA adapters for knowledge injection, differentiable memory modules like Titans) or hybrid architectures that combine parametric and external storage (e.g., Memory^3) blur the boundary between tool adaptation and agent adaptation. When the memory update requires gradient-based changes to the agent's own parameters, the method is better classified under A1 or A2; when only an auxiliary parameter set is updated while the core agent remains frozen, the method occupies the T1/T2 boundary.

We adopt the convention that the *dominant* form of memory adaptation determines the paradigm label. For most external, non-parametric memory systems discussed in this survey, the T2 label applies: the frozen agent's outputs supervise the evolution of the memory module. We flag boundary cases explicitly in §5 to avoid flattening the important architectural differences among memory forms.

3.3 Illustrative Examples

We illustrate the adaptation paradigms through two representative settings: retrieval-augmented generation (RAG) and code-execution-based tasks. These settings highlight the central role of tool use in agentic AI while exhibiting distinct interaction patterns and evaluation protocols.

For each application, we present paired A1 and A2 examples sharing the same tool-call action (document retrieval or code execution), allowing a direct contrast between tool-feedback-based adaptation (A1) and agent-output-based adaptation (A2). We also provide a T2 example in the RAG setting.

3.3.1 Agent Adaptation Examples Across Two Applications

We illustrate the A1 and A2 paradigms through two tool-use settings: retrieval-augmented generation (RAG) and code-execution-based question answering. For each, we describe the problem setup and then present paired examples that instantiate A1 and A2 under the same tool-call action.

Retrieval-Augmented Generation (RAG) Setting. In the RAG setting, the agent receives a query and performs a retrieval action to obtain relevant documents from a database. Formally, the agent produces a retrieval query a, the retriever returns a set of documents y, and the agent synthesizes these documents together with the original query to generate a final answer o.

- **A1 example**. DeepRetrieval [22] optimizes the agent using feedback signals computed directly from retrieval quality. After generating a retrieval query a, the retriever returns documents y, and metrics such as recall or nDCG are computed from y and used as the reward for updating the agent. Since the adaptation signal depends solely on the tool-execution outcome, this represents the A1 paradigm.
- **A2 example**. Search-R1 [50] follows the full RAG pipeline, where the agent first retrieves documents and then integrates them into its context to produce a final answer o. The adaptation signal is computed from the correctness or quality of this final answer by calculating exact matching accuracy. Because the optimization is guided by the agent's final output rather than the retrieval result alone, this falls under the A2 paradigm.

Contrast. DeepRetrieval (A1) directly optimizes retrieval quality but provides no gradient signal for answer synthesis; Search-R1 (A2) optimizes end-to-end answer correctness but must discover good retrieval strategies as a side effect, making credit assignment harder.

Code-Execution-Based Task Setting. In code-execution-based tasks, the agent receives a problem description and produces executable code as the tool-call action. The sandbox executes the code and returns an execution result y, which the agent may optionally use to generate a final answer o.

- **A1 example:** DeepSeek-R1 (code) [25]. During reinforcement learning, DeepSeek-R1 generates code that is executed inside a sandbox (when the reward is derived solely from test-case pass rates, this is A1). The execution output, such as test-case pass rate or numerical correctness, is used directly as the reward for policy optimization. Since adaptation is based entirely on the tool's execution result, this example fits the A1 paradigm.
- **A2 example:** ReTool [51] also generates executable code, but the sandbox result is fed back into the agent as additional context. The agent then produces a final answer o, whose correctness determines the reward. Because the adaptation signal depends only on the final output of the agent after integrating tool feedback, this corresponds to the A2 paradigm.

Contrast. DeepSeek-R1's A1 formulation ties the reward to test-case pass rates, giving the agent a precise signal per code invocation. ReTool's A2 formulation, by contrast, rewards only the final answer, so the model must learn *when* to invoke the code sandbox—a richer but noisier learning problem.

3.3.2 Tool Adaptation Examples in the RAG Setting

In many practical systems, the central agent is a closed-source API model (e.g., GPT, Claude, or Gemini) that cannot be fine-tuned. Training an open-source model to match such performance is challenging due to data curation

requirements, scaling laws, and infrastructure demands. A more feasible strategy is to treat the closed-source model as a fixed agent and adapt auxiliary tools around it. In the RAG setting, this motivates tool adaptation for components such as retrievers.

T1 examples. (1) Classic Dense Retrievers. A standard dense retriever (bi-encoder trained with contrastive learning) is the canonical T1 tool: given a query a, it returns documents $y = \mathcal{T}(a)$ optimized for recall, and any frozen agent can consume y for downstream reasoning without having participated in the retriever's training. **(2) Learned Subagents as Agent-Agnostic Tools.** Beyond classic dense retrievers, once agent adaptation has produced strong retrieval-oriented models, these learned models can themselves be reused as tools under the T1 paradigm. For example, a model trained in the DeepRetrieval [22] style can be deployed purely as a high-quality subagent that rewrites queries for improved retrieval over specific document databases. Given a tool call action a (a retrieval query), the subagent returns a reformulated query or a curated document set $y = \mathcal{T}(a)$ with higher recall or relevance, which is then consumed by a fixed closed-source agent that performs the final reasoning and answer synthesis.

T2 examples. Under the T2 paradigm, the tool is adapted using supervision signals derived directly from a fixed agent's final outputs. In the RAG setting, both S3 [28] and AgentFlow [52] provide representative examples. The tool (a learnable search subagent) is updated based on the fixed agent's output signal so that its behavior becomes increasingly aligned with what the fixed agent needs for successful downstream reasoning.

Concretely, in S3 [28], given a question x, the learnable subagent $\mathcal{T}$ takes x as input and internally generates a retrieval query $a' = \mathcal{T}(x)$. This query is executed on a static search engine to retrieve a document set y, which is then inserted into the frozen agent's context. The fixed agent consumes (x, y) and produces a final answer $o = \mathcal{A}(x, y)$. An evaluation function $\mathcal{O}_{\text{agent}}(o)$ (e.g., answer correctness) assigns a scalar reward, which is then used to update the tool so that its future retrieval behaviors yield document sets that more effectively support the agent's downstream reasoning. Thus, S3 directly realizes the T2 objective with the agent-output signaled tool adaptation specifically for the fixed agent's downstream performance. AgentFlow [52] further extends this idea by training a more expressive planning-oriented subagent capable of multi-tool decision-making, applying T2 supervision signal to align a richer planning policy with the fixed agent's final-output preferences.

4 Agent Adaptation

Agent adaptation refers to the mechanisms through which agents refine their behavior based on feedback from tools, environments, or their own outputs. A central design choice is *where* the feedback signal originates, which determines both what the agent can learn and what failure modes remain invisible. When the signal comes from tool execution (**A1**), the agent receives dense, causally grounded feedback tied to specific actions—but this signal is blind to whether the agent's overall reasoning strategy is sound. When the signal comes from evaluating the agent's own outputs (**A2**), the agent can optimize holistic task success—but the sparser, episode-level reward makes credit assignment harder and training less data-efficient. This A1/A2 tension between *mechanistic precision* and *strategic flexibility* structures the design space reviewed in this section.

Formally, let $\mathcal{A}$ denote an agent parameterized by its internal configuration or policy (prompt templates or model weights), and let $\mathcal{T}$ represent the set of accessible tools. The two paradigms correspond to distinct optimization objectives:

$$\textbf{(A1)} \quad \mathcal{A}^* = \arg\max_{\mathcal{A}} \mathcal{O}_{\text{tool}}(\mathcal{A}, \mathcal{T}), \qquad \textbf{(A2)} \quad \mathcal{A}^* = \arg\max_{\mathcal{A}} \mathcal{O}_{\text{agent}}(\mathcal{A}, \mathcal{T}),$$

where $\mathcal{O}_{\text{tool}}$ scores the correctness or utility of tool execution outcomes (e.g., code compilation success, retrieval precision), and $\mathcal{O}_{\text{agent}}$ scores the quality of the agent's generated outputs (reasoning validity, factual accuracy, or preference alignment). The distinction has practical consequences: $\mathcal{O}_{\text{tool}}$ yields per-action credit but is silent about reasoning quality, whereas $\mathcal{O}_{\text{agent}}$ captures end-to-end performance but conflates tool-use skill with reasoning skill, complicating diagnosis when either degrades.

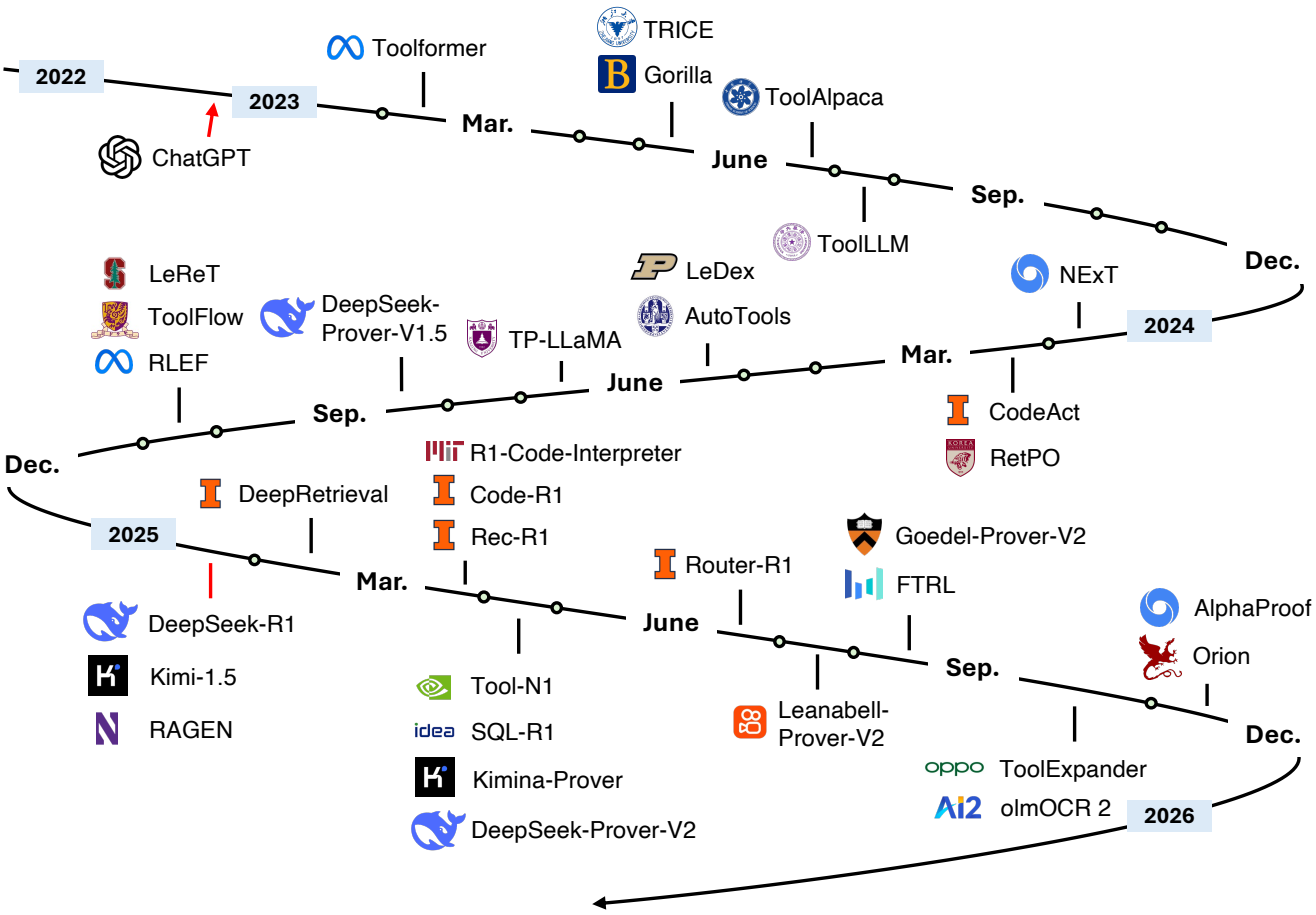

Figure 4 Development timeline of A1 methods (agent adaptation with tool-execution result as signal).

4.1 A1: Tool Execution Result as Signal

A1 adaptation treats tool and environment outputs as ground-truth supervision: because the signal is determined by external execution rather than model-internal beliefs, it is verifiable, reproducible, and dense enough to support both supervised and reinforcement-based learning. The key challenge is that such signals optimize *individual actions* (a single API call, a single code execution) and can miss system-level failures that only surface when multiple actions compose.

Table 1 A1 Methods (Tool Execution Signaled): Earlier Methods (SFT & DPO) and Recent RLVR-based Methods

Time	Method	Venue	Task(s)	Tool(s)	Agent Backbone	Tuning	Links
SFT & Off-Policy Methods							
2023.02	Toolformer	NeurIPS'23	QA, Math	Calculator, QA system, Search Engine, Translation System, Calendar	GPT-J	SFT	
2023.05	TRICE	NAACL'24	Math Reasoning, QA	Calculator, WikiSearch, Atlas QA Model, NLLB Translator	ChatGLM, Alpaca, Vicuna	SFT, Contrastive Learning	
2023.05	Gorilla	NeurIPS'24	Tool-Calling, API Retrieval	APIs	LLaMA	SFT	

Continued on next page

Table 1 – Continued from previous page

Time	Method	Venue	Task(s)	Tool(s)	Agent Backbone	Tuning	Links
2023.06	ToolAlpaca	arXiv	Multi-Turn Tool-Use	Simulated APIs	Vicuna	SFT	
2023.07	ToolLLM	ICLR'24	Tool-Calling, API Planning, Multi-Tool Reasoning	Real-World APIs	LLaMA, Vicuna	SFT	
2024.01	NExT	ICML'24	Program Repair	Code Executor	PaLM2	SFT	
2024.02	CodeAct	ICML'24	Coding	Code Executor	LLaMA2, Mistral	SFT	
2024.02	RetPO	NAACL'25	IR	Retriever	LLaMA2-7B	SFT, DPO	
2024.03	CYCLE	OOPSLA'24	Coding	Code Executor	CodeGen, StarCoder	SFT	
2024.05	AutoTools	WWW'25	Tool-Calling	APIs	GPT4, LLaMA3, Mistral	SFT	
2024.06	TP-LLaMA	NeurIPS'24	Tool-Calling	APIs	LLaMA2	SFT, DPO	
2024.10	ToolFlow	NAACL'25	Tool-Calling	APIs	LLaMA3.1	SFT	
2024.10	LeReT	ICLR'25	IR	Dense Retriever	LLaMA3, Gemma2	DPO-like (IPO)	
			RLVR Methods				
2024.05	LeDex	NeurIPS'24	Coding	Code Executor	StarCoder & CodeLlaMA	SFT, PPO	
2024.08	DeepSeek-Prover-V1.5	ICLR'25	Formal Theorem Proving	Lean 4 Prover	DeepSeek-Prover-V1.5-RL	SFT, GRPO	
2024.10	RLEF	ICML'25	Coding	Code Executor	LLaMA3.1	PPO	
2025.01	DeepSeek-R1-Zero (Code)	Nature	Coding	Code Executor	DeepSeek-V3-Base	GRPO	
2025.02	DeepRetrieval	COLM'25	Web Search, IR, Text2SQL	Search Engine, Retrievers, SQL exec.	Qwen2.5, LLaMA3.2	PPO, GRPO	
2025.03	Code-R1	—	Coding	Code Executor	Qwen2.5	GRPO	
2025.03	ReZero	arXiv	Web Search, IR	Web Search Engine	LLaMA3.2	GRPO	
2025.03	Rec-R1	TMLR'25	Recommendation Optimization	Recommendation System	Qwen2.5, LLaMA3.2	GRPO	
2025.04	SQL-R1	NeurIPS'25	Text2SQL Search	SQL Engine	Qwen2.5, OmniSQL	SFT, GRPO	
2025.04	Kimina-Prover	arXiv	Formal Theorem Proving	Lean 4 Compiler, Numina Lean Server	Qwen2.5	SFT, GHPO	
2025.04	DeepSeek-Prover-V2	arXiv	Formal Theorem Proving	Lean 4 Compiler	DeepSeek-V3	SFT, GRPO	
2025.05	Tool-N1	arXiv	Tool-Calling	Tool APIs	Qwen2.5	GRPO	
2025.05	R1-Code-Interpreter	arXiv	Coding	Code Execution Sandbox	Qwen2.5	GRPO	
2025.06	Router-R1	NeurIPS'25	Multi-Round Routing	LLM Routing Pool	Qwen2.5, LLaMA3.2	PPO	
2025.07	Leanabell-Prover-V2	arXiv	Formal Theorem Proving	Lean 4 Verifier	Kimina, DeepSeek-V2	SFT, DAPO	
2025.08	Goedel-Prover-V2	arXiv	Formal Theorem Proving	Lean Compiler	Qwen3	SFT, GRPO	
2025.08	FTRL	arXiv	Multi-Step Tool-Use	Simulated APIs	Qwen3	GRPO	

Continued on next page

Table 1 – Continued from previous page

Time	Method	Venue	Task(s)	Tool(s)	Agent Backbone	Tuning	Links
2025.09	Tool-R1	arXiv	Tool-Augmented Reasoning, QA	Code Execution, Multimedia Tools	Qwen2.5	GRPO	
2025.09	WebGen-Agent	arXiv	Website Generation	VLM, GUI Agent, Code Executor	Qwen2.5-Code, Qwen3	SFT, Step-GRPO	
2025.10	ToolExpander	arXiv	Tool-Calling	Tool APIs	Qwen2.5	SFT, GRPO	
2025.10	AlphaProof	Nature	Formal Theorem Proving	Lean Solver	Transformer (3B Enc-Dec)	SFT, AlphaZero, TTRL	
2025.10	olmOCR2	arXiv	Document OCR	Synthetic Document Verifier	Qwen2.5-VL	SFT, GRPO	
2025.11	Orion	arXiv	IR	Retrievers	LFM2	GRPO	

4.1.1 Earlier Works: SFT & Off-Policy Methods

Early A1-type methods train agents from pre-collected data via SFT or DPO. Their shared limitation is an off-policy data distribution: the agent learns from trajectories it did not generate, which can cause a mismatch between training and deployment behavior. Within this family, methods diverge along a key axis—**what counts as correct?**—progressing from answer-level correctness to format-level validity to raw execution outcomes.

Toolformer [4] (NeurIPS 2023) illustrated the earliest point on this axis: tool outcomes serve as **self-supervised signals**. The model inserts candidate API calls into text, executes them, and retains a call only if it reduces perplexity beyond a threshold ($L_i^- - L_i^+ \geq \tau_f$). Because the supervision is derived from the model's own likelihood rather than an external correctness criterion, Toolformer can discover *when* tools help but cannot distinguish *how well* a tool call was constructed. Subsequent work addressed this gap through three progressively stronger notions of correctness:

- **Golden-answer alignment:** the tool-augmented output must match a known correct response or expert trajectory.
- **Golden-format alignment:** the tool call must be structurally and syntactically valid, independent of downstream output.
- **Direct-execution alignment:** supervision emerges from actually running the tool, closing the gap between training signal and deployment behavior.

Golden-answer alignment. The simplest instantiation defines correctness as matching a known answer or expert trajectory. A key question within this family is *how to exploit negative examples*: early methods discard failed trajectories, while later ones treat them as contrastive signal.

ToolAlpaca [53] pioneered the closed-loop generate-execute-evaluate-finetune cycle, where the model calls a tool, observes the runtime outcome (returned values, errors, or completion states), and retrains on the result. Iteration over this loop aligns the model's internal representation with actual tool semantics and enables generalization to unseen tools—but only positive outcomes contribute to learning. **TRICE** [54] (NAACL 2024) introduces a ranking loss that scores candidate responses by comparing execution results against ground-truth answers, teaching the model to invoke tools only when they improve accuracy. **TP-LLaMA** [55] (NeurIPS 2024) goes further by explicitly mining failure information that ToolAlpaca and earlier systems (e.g., ToolLLaMA [10]) discard: at each decision node along a successful trajectory, the expert's correct next step is preferred (y_w) and any failed branch is dispreferred (y_l), and DPO on these pairs converts failure signals into contrastive supervision. The progression from ToolAlpaca to TP-LLaMA thus illustrates a recurring theme in A1 methods: the richer the exploitation of negative feedback, the more data-efficient the adaptation.

Golden-format alignment. A complementary axis defines correctness structurally: a tool call is correct if it is

syntactically and logically valid, regardless of whether the final answer is right. This decoupling is valuable when ground-truth answers are expensive to obtain but canonical call formats exist.

Gorilla [56] (NeurIPS 2024) fine-tunes a retrieval-augmented LLaMA model on a large set of machine learning APIs, defining correctness via Abstract Syntax Tree (AST) subtree matching—more robust than string matching because it tolerates differences in parameter order or optional arguments. **ToolFlow** [57] (NAACL 2025) complements Gorilla by tackling data quality: a tool graph built from parameter and return-value similarities guides the selection of interacting tool subsets, and a planned generation step organizes multi-turn requests for logical consistency. Together, these two works show that format-level supervision can scale to large, evolving API surfaces where golden answers are impractical, but they also reveal a limitation: format correctness does not guarantee functional correctness, so models trained this way can produce well-formed but semantically wrong calls.

Direct-execution alignment. The strongest form of A1 supervision closes the loop entirely: the tool is executed, and its output—success, failure, returned values—becomes the training signal. This eliminates the need for pre-labeled answers or canonical formats but introduces a new challenge: designing execution environments that are safe, reproducible, and informative enough to support learning.

Two design patterns have emerged. The first generates executable code as the action representation. **CodeAct** [58] (ICML 2024) replaces textual or JSON-based commands with sandboxed code actions, so the environment's execution feedback directly grounds the model in tool causality. **NExT** [59] (ICML 2024) applies the same principle to program repair via an iterative Sample-Filter-Train loop: candidate fixes are validated by unit tests, and only passing solutions enter the next fine-tuning round, yielding progressively sharper execution-aware rationales. The second pattern targets autonomous tool discovery. **ToolLLM** [10] and **AutoTools** [60] (WWW 2025) have the LLM itself parse raw API documentation into callable functions (tool encapsulation) and then compose multi-function programs to solve user queries (tool programming), with execution outcomes driving iterative self-improvement.

A third pattern applies execution-grounded preference learning to retrieval. **LeReT** [61] (ICLR 2025) generates diverse search queries, scores retrieved documents via a reward function, and fine-tunes the query generator through Identity Policy Optimization (IPO):

$$\mathcal{L}_{\text{IPO}} = \mathbb{E}_{(x,y_w,y_l)\sim D_p}\left[(\tilde{r}_\phi(x,y_w) - \tilde{r}_\phi(x,y_l) - 0.5\tau^{-1})^2\right],$$

where $\tilde{r}_\phi(x,y) = \log \frac{\pi_\phi(y|x)}{\pi_{\text{ref}}(y|x)}$ and τ controls the target margin. Because the reward comes from running the retriever rather than from labeled answers, LeReT adapts to arbitrary off-the-shelf retrievers without modifying the generator. **RetPO** [62] (NAACL 2025) follows the same logic at lower cost: GPT-4 generates candidate query rewrites, an off-the-shelf retriever (e.g., BM25) scores each by retrieval quality, and a smaller open-source LM is trained via DPO to produce high-reward rewrites.

Summary of SFT & off-policy A1 methods. The three alignment stages trace a clear arc: from Toolformer's self-supervised likelihood reduction, through answer- and format-level matching, to direct execution feedback. Each step tightens the coupling between training signal and deployment behavior, reducing the "reality gap" between what the model optimizes and what determines success at inference time. Yet all off-policy methods share a fundamental ceiling: because they learn from pre-collected trajectories, they cannot explore novel tool-use strategies that were absent from the training distribution. This limitation motivates the on-policy RLVR methods described next.

4.1.2 RLVR-Based Methods

Reinforcement learning with verifiable reward (RLVR) removes the off-policy ceiling of SFT and DPO by letting the agent explore on-line: it proposes actions, executes them, observes verifiable outcomes, and updates its policy—all within the same training loop. The on-policy setting introduces two new design axes absent from the methods above: (i) *reward density*—whether feedback arrives per step (as in theorem proving) or only at episode end (as in multi-tool reasoning)—which governs credit assignment difficulty; and (ii) *reward composition*—how task-specific, format, and regularization terms are combined. We organize the domain-specific instantiations below and distill cross-domain principles at the end.

Web search and information retrieval. **DeepRetrieval** [22] (COLM 2025) established the template: query reformulation is cast as an MDP with retrieval metrics (Recall@K, NDCG, or SQL execution accuracy) as the reward, optimized via KL-regularized PPO:

$$\hat{\pi} = \arg\max_{\pi} \mathbb{E}_{q,q' \sim \pi}\left[r(q,q') - \beta \log \frac{\pi(q'|q)}{\pi_{\text{ref}}(q'|q)}\right],$$

where $r(q,q') = r_{\text{retrieval}}(q,q') + r_{\text{format}}(q')$. A single framework thereby adapts across literature search, QA retrieval, and text-to-SQL, yielding roughly a threefold recall improvement (65.1% vs. 24.7%) on real-world search engines.

Subsequent work addresses two limitations of this single-step formulation. **ReZero** [63] adds retry-aware reward shaping via GRPO, teaching the agent to persist after failed searches in partially observable web environments. **Orion** [64] extends from single-step to multi-turn adaptive search, using turn-level rewards based on normalized similarity and rank; notably, compact 350M–1.2B models learn effective multi-hop search strategies, demonstrating that on-policy RLVR can substitute for large-controller scale when reward density is sufficient.

Code-based tools. Code execution provides a near-ideal A1 environment: feedback is deterministic, sandboxable, and obtainable at every compilation or test-run step. The central design question is how to keep this feedback *reliable* as tasks grow more heterogeneous. **LeDex** [65] (NeurIPS 2024) uses a composite PPO reward combining unit-test correctness (CodeBLEU) with explanation quality (semantic similarity), illustrating the value of multi-faceted rewards. **RLEF** [21] (ICML 2025) formalizes code synthesis as a partially observable MDP where the agent generates, receives public-test feedback, and iterates—showing that multi-turn interaction is key to solving problems beyond the model's single-shot capability. **Code-R1** [66] shifts focus from the policy to the *reward pipeline*: by eliminating false positives from faulty tests, unsolvable prompts, and mismatched sandboxes, it demonstrates that reward quality dominates reward quantity. **R1-Code-Interpreter** [67] addresses the heterogeneity of code-interpreter tasks (math, retrieval, data analysis) through multi-stage curriculum learning that prioritizes samples by their improvement potential, mitigating sparse and unstable rewards. **Tool-R1** [68] tackles sample efficiency directly via a dynamic sample queue that caches and reuses high-quality trajectories, reducing the exploration cost of multi-step reasoning.

Formal theorem proving [69–77] provides a canonical domain for RLVR under the A1 paradigm, as proof assistants provide ground-truth, tool-execution-signaled feedback at every step. In this setting, the agent proposes one or more tactics (i.e., proof steps), a formal proof checker (the tool) deterministically verifies their validity, and the resulting validated proof-state transition is returned to the agent. The verification outcome (e.g., whether a tactic is accepted, whether it advances the proof state, or whether a complete proof is achieved) serves directly as a verifiable reward signal for policy optimization. Compared to code-execution RLVR, where unit tests may be sparse or incomplete, theorem proving offers step-wise semantic verification with minimal ambiguity, enabling denser rewards and substantially easing long-horizon credit assignment. Recent systems such as **AlphaProof** [78] (Nature 2025), **DeepSeek-Prover-V2** [79] (ICLR 2025), **Kimina-Prover** [73], and **Leanabell-Prover-V2** [80] use this verifier feedback to train multi-step proof search policies via reinforcement learning, while a complementary line of work augments the native proof checker feedback with auxiliary guidance signals to prioritize trajectories, shape exploration, or stabilize optimization on top of verifier-grounded rewards [81–85]. While RLVR is well suited for learning proof strategies under a fixed prover snapshot, formal theorem proving also highlights a broader adaptation challenge: formal libraries (e.g., MATHLIB [86]) and large, actively evolving formalization projects [87] built by the Lean community grow continuously, expanding the available premise space. Addressing this non-stationarity often requires complementary continual or low-resource adaptation mechanisms beyond pure RLVR, which we discuss in §9.2.

Multi-tool reasoning systems. When multiple tools must be composed sequentially or conditionally, the agent faces a combinatorial action space and sparser episode-level rewards. Methods in this category address the challenge through three complementary strategies: (i) *routing*, (ii) *environment construction*, and (iii) *dense step-level rewards*.

Router-R1 [88] (NeurIPS 2025) exemplifies routing: a policy LLM learns to alternate between internal reasoning and external model selection, dynamically invoking different LLMs from a routing pool. **FTRL** [89] tackles environment construction by automatically synthesizing diverse tool-use training environments through a multi-stage pipeline, so

the agent can train without hand-crafted external toolsets; a verifiable reward balances invocation accuracy with task completion. **Tool-N1** [26] separates internal reasoning (`<think>` tags) from external execution (`<tool_call>` tags), giving the RL optimizer distinct credit-assignment channels for thought and action. **WebGen-Agent** [90] demonstrates the dense-reward strategy: appearance scores from a visual-language model and functionality scores from a GUI-agent are combined via Step-GRPO to supervise interactive website code generation at every turn. **ToolExpander** [91] targets the resource-constrained regime with Dynamic Multi-Round Hard Sampling (replacing hard samples with few-shot exemplars) and a Self-Exemplifying Thinking mechanism that rewards self-generated analysis, stabilizing GRPO for small LLMs.

Domain-specific extensions. A1-type RLVR has also been applied beyond the core domains above. **Rec-R1** [92] (TMLR 2025) casts recommendation as an RL policy over LLM generation, using NDCG and Recall as rewards. **SQL-R1** [93] uses a composite reward (format, execution, result, and length) for NL-to-SQL. **olmOCR 2** [94] replaces the SFT-based teacher imitation of olmOCR 1 [95] with diverse binary unit tests (text presence, reading order, table accuracy, formula rendering) as the reward signal for document OCR. These applications reinforce a common pattern: whenever a domain provides a cheap, deterministic correctness oracle, A1-style RLVR can yield large gains with modest engineering effort.

Cross-domain principles. Despite the diversity of domains (web search, code, theorem proving, multi-tool reasoning, and specialized applications), four recurring patterns cut across all A1 RLVR work:

- **Signal density determines learning efficiency.** Domains with dense, per-step feedback (theorem proving, code execution) enable faster convergence than those with sparse, episode-level rewards (multi-tool reasoning). Dense feedback explains why A1 methods in code and proof domains often require less data than A2 counterparts.
- **Reward quality dominates reward quantity.** Code-R1 [66] and DeepRetrieval [22] both show that investing in clean, verified reward signals yields larger gains than scaling training data, a finding consistent across retrieval, code, and SQL domains.
- **Format rewards are necessary but not sufficient.** Nearly all successful RLVR methods combine a task-specific reward with a format compliance reward (e.g., DeepRetrieval's r_{format}, Tool-N1's structured output tags). Format rewards alone cannot drive meaningful adaptation, but their absence leads to degenerate outputs.
- **Stabilization mechanisms are domain-agnostic.** KL regularization, dynamic sampling, and curriculum scheduling appear across all domains, suggesting that the challenge of stabilizing on-policy RL for language agents is fundamental rather than domain-specific.

These principles notwithstanding, RLVR remains expensive: it requires careful reward design, interactive compute, and explicit stabilization—costs that motivate the tool-centric adaptation paradigms discussed in later sections.

4.2 A2: Agent Output as Signal

Whereas A1 optimizes the mechanics of individual tool calls, A2 optimizes the agent's end-to-end output—reasoning chains, final answers, or generated artifacts—using evaluations of those outputs as the training signal. The shift from action-level to output-level feedback brings two consequences: (i) the agent can learn *strategic* behaviors (when to invoke a tool, how deeply to search, whether to self-correct) that A1 signals cannot reward, and (ii) credit assignment becomes harder because a single episode-level reward must be attributed across many internal decisions.

We distinguish two settings based on whether tools participate in the evaluation loop:

- **Agent Adaptation w/o Tools:** The agent improves intrinsic reasoning (mathematics, coding, logical inference) by optimizing against evaluations of its generated solutions. The design space here is defined by the *granularity* of the evaluation signal (scalar correctness vs. structured linguistic critique) and *whether weights are updated* (RL-based vs. inference-time refinement).
- **Agent Adaptation w/ Tools:** The agent's outputs are assessed in conjunction with tool interactions, so the training signal reflects both reasoning quality and tool-coordination skill. The key additional challenge is learning the *meta-policy*—when and how to invoke tools—alongside the reasoning policy.

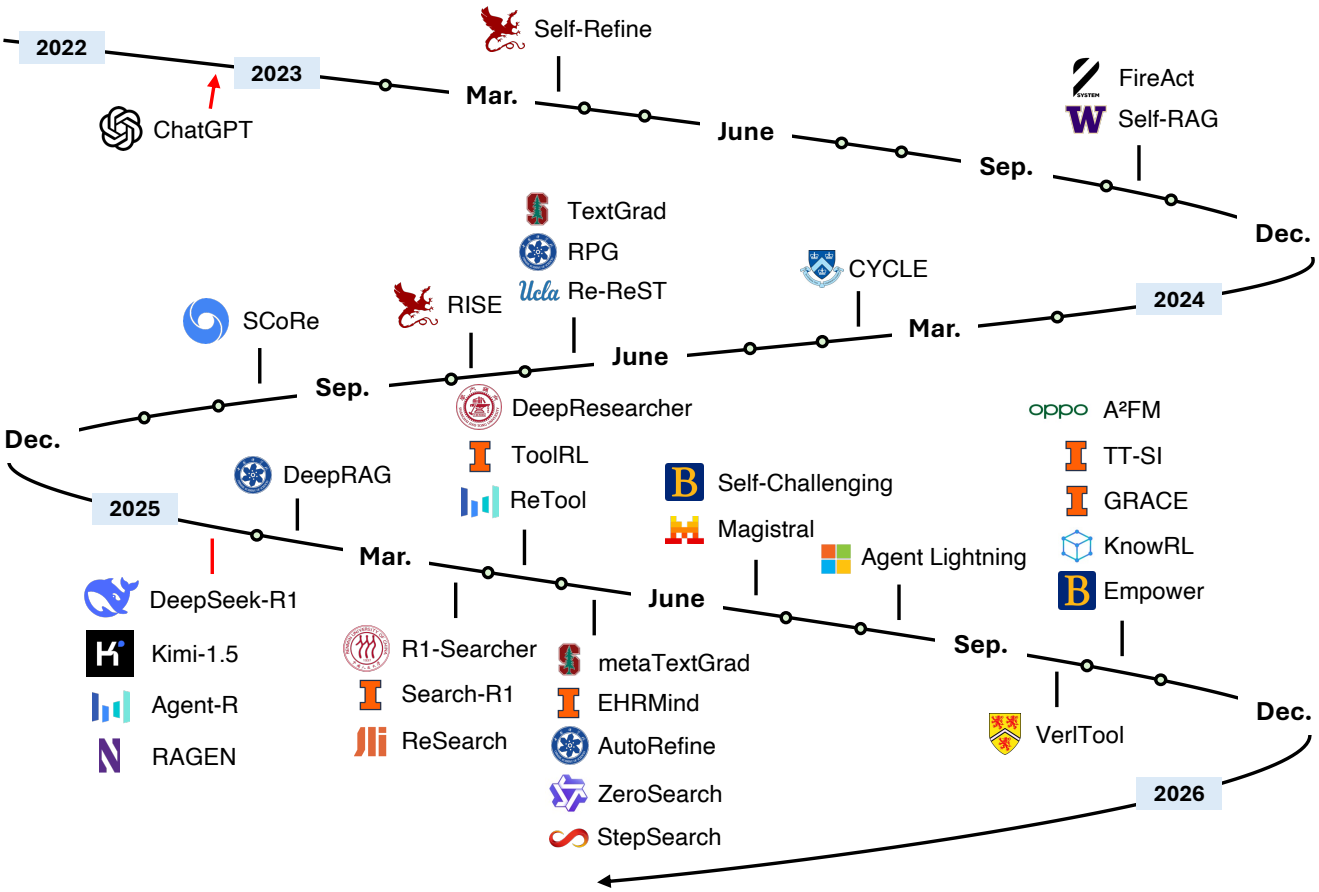

Figure 5 Development timeline of A2 methods (agent adaptation with agent output as signal).

Table 2 A2 Methods: Agent Output Signaled Agent Adaptation

Time	Method	Venue	Task(s)	Tool(s)	Agent Backbone	Tuning	Links
			w/o Tools				
2023.03	Self-Refine	NeurIPS'23	Dialogue, Math, Coding	—	GPT3.5, GPT4, CODEX	Prompt Engineering	
2024.06	TextGrad	Nature	Code Optimization, Molecule Optimization, etc.	—	GPT3.5, GPT4o	Prompt Engineering	
2024.07	RISE	NeurIPS'24	Math	—	LLaMA2, LLaMA3, Mistral	SFT	
2024.09	SCoRe	ICLR'25	Math, Coding, QA	—	Gemini1.0 Pro, Gemini1.5 Flash	REINFORCE	
2025.01	DeepSeek-R1-Zero (Math)	Nature	Math	—	DeepSeek-V3	GRPO	
2025.01	Kimi k1.5	arXiv	Math, Coding	—	Kimi k1.5	GRPO	
2025.05	EHRMind	arXiv	EHR-based Reasoning	—	LLaMA3	SFT, GRPO	

Continued on next page

Table 2 – Continued from previous page

Time	Method	Venue	Task(s)	Tool(s)	Agent Backbone	Tuning	Links
2025.05	metaTextGrad	NeurIPS'25	QA, Math, Word Sorting	—	Qwen3-235B-A22B, Claude-3.5-Sonnet	Prompt Engineering	
2025.06	Magistral	arXiv	Math, Coding	—	Magistral	PPO, GRPO	
2025.10	GRACE	arXiv	Embedding Tasks	—	Qwen2.5, Qwen3, LLaMA3.2	GRPO	
2025.10	KnowRL	arXiv	Knowledge Calibration	—	LLaMA3.1, Qwen2.5	REINFORCE++	
2025.10	Empower	arXiv	Coding	—	Gemma3	SFT	
w/ Tools							
2023.10	FireAct	arXiv	QA	Search API	GPT3.5, LLaMA2, CodeLLaMA	SFT	
2023.10	Self-RAG	ICLR'24	QA, Fact Verification	Retriever	LLaMA2	SFT	
2024.06	RPG	EMNLP'24	QA, Reasoning	Search Engine, Retriever	LLaMA2, GPT3.5	SFT	
2024.06	Re-ReST	EMNLP'24	QA, VQA, Sequential Decision, Coding	Tool APIs	Various Models	DPO	
2025.01	Agent-R	arXiv	Various Tasks	Monte Carlo Tree Search	Qwen2.5, LLaMA3.2	SFT	
2025.02	RAS	arXiv	QA	Retriever	LLaMA2, LLaMA3.2	SFT	
2025.03	R1-Searcher	arXiv	QA	Retriever	LLaMA3.1, Qwen2.5	REINFORCE++	
2025.03	Search-R1	COLM'25	QA	Search Engine, Retriever	Qwen2.5	PPO, GRPO	
2025.03	ReSearch	NeurIPS'25	QA	Search Engine, Retriever	Qwen2.5	GRPO	
2025.04	ReTool	arXiv	Math	Code Interpreter	Qwen2.5	PPO	
2025.04	DeepResearcher	arXiv	QA, Reasoning, Deep Research	Web Search API, Web Browser	Qwen2.5	GRPO	
2025.04	ToolRL	arXiv	Tool Calling	Tool APIs	Various Models	GRPO	
2025.05	AutoRefine	NeurIPS'25	QA	Retriever	Qwen2.5	GRPO	
2025.05	ZeroSearch	arXiv	QA	Search Engine, Web Search	Qwen2.5, LLaMA3.2	REINFORCE, GPRO, PPO, SFT	
2025.05	StepSearch	EMNLP'25	QA	Search Engine, Retriever	Qwen2.5	StePPO	
2025.06	Self-Challenging	arXiv	Multi-Turn Function-Calling, Calculation	Code Interpreter, Web Browser	LLaMA3.1	REINFORCE, SFT	
2025.06	MMSearch-R1	arXiv	QA, VQA	Image Search, Web Browser, Retriever	Qwen2.5	REINFORCE, SFT	
2025.07	DynaSearcher	arXiv	QA	Document Search, KG Search	Qwen2.5, LLaMA3.1	GRPO	

Continued on next page

Table 2 – Continued from previous page

Time	Method	Venue	Task(s)	Tool(s)	Agent Backbone	Tuning	Links
2025.07	CodePRM	ACL'25	Coding	Code Executor	Qwen2.5-Coder	SFT	
2025.08	Agent Lightning	arXiv	Text2SQL, Math	SQL Executor, Retriever, Calculator	LLaMA3.2	LightningRL	
2025.08	MedResearcher-R1	arXiv	Medical QA	Medical Retriever, Web Search API, Document Reader	MedResearcher-R1	SFT, GRPO	
2025.09	VerlTool	arXiv	Math, QA, SQL, Visual, Web Search, Coding	Code Interpreter, Search Engine, SQL Executor, Vision Tools	Qwen2.5, Qwen3	GRPO	
2025.10	A^2FM	arXiv	Web Navigation, Math, QA	Search Engine, Crawl, Code Executor	Qwen2.5	APO,GRPO	
2025.10	TT-SI	arXiv	Tool Calling	Tool APIs	Qwen2.5	Test-Time Fine-Tuning	

4.2.1 Agent Adaptation w/o Tools

Three design axes organize the methods below: (1) the *reward granularity* ranges from binary final-answer correctness to structured natural-language critiques; (2) the *optimization locus* ranges from weight updates (RL, SFT) to inference-time refinement (no weight changes); and (3) the *objective scope* ranges from task-specific correctness to broader goals such as calibration or human empowerment.

Scalar-reward RL (the R1 paradigm). DeepSeek-R1 [25] (Nature 2025) demonstrated that RLVR with binary final-answer correctness is sufficient to unlock strong mathematical and coding reasoning in large agents, revealing a scalable pathway beyond SFT. **Kimi-1.5** [96] extended this to multi-modal agents with simplified policy optimization, matching or surpassing DeepSeek-R1 across reasoning benchmarks. Together, these works established the **R1 paradigm**: RL on verifiable output evaluations as the primary driver of reasoning improvement.

Subsequent R1-style studies explore the boundaries of this paradigm by varying the objective: **EHRMind** [97] applies RLVR to clinical reasoning (EHR interpretation), finding that a lightweight SFT warm-up is necessary before RL to ground domain knowledge—a pattern that recurs across specialized domains. **KnowRL** [98] replaces task correctness with self-knowledge calibration: the agent is rewarded for accurately assessing its own confidence, improving reliability without targeting any single task. **Empower** [99] goes further by defining the objective as maximizing human empowerment rather than correctness, aligning assistive agents to multi-turn coding tasks using only offline text data. **GRACE** [100] transforms contrastive-learning objectives into RL-style policy signals, bridging generative reasoning with representation learning. A related study, **Rec-R1** [92], applies RL to product re-ranking using recommendation metrics (NDCG, Recall) as rewards. Rec-R1 straddles the A1/A2 boundary—A1 when the reward is computed from the tool's ranking output, A2 when it evaluates the agent's overall recommendation quality—illustrating that the distinction is not always binary but depends on where in the agent-tool pipeline the evaluation signal is computed.

Inference-time self-refinement (no weight updates). A complementary line of work achieves adaptation at inference time, avoiding the cost of retraining. **Self-Refine** [101] (NeurIPS 2023) introduced the generate-critique-revise loop: the same LLM produces an initial response, evaluates it, and revises it based on self-generated textual feedback, improving output quality across dialogue, mathematics, and code without any supervised data or auxiliary models. **SCoRe** [101] (ICLR 2025) makes self-correction trainable: multi-turn on-policy RL encourages the model to iteratively refine its own responses, operationalizing the Self-Refine loop into a stable learning signal. The

progression from Self-Refine to SCoRe mirrors the SFT-to-RLVR progression in A1: inference-time heuristics are replaced by learned policies that improve with scale.

Structured linguistic feedback. **TextGrad** [102] (Nature 2025) generalizes the scalar reward to natural-language critiques that describe *how* to improve the output. These "textual gradients" enable optimization across black-box LLM systems without parameter access, improving GPT-4o's zero-shot code accuracy on LEETCODE-HARD from 26% to 36% and MMLU-Physics from 91.2% to 95.1%. **metaTextGrad** [103] (NeurIPS 2025) applies the same principle recursively to the optimizer itself, using validation feedback to refine the optimizer's prompts. TextGrad and metaTextGrad occupy a unique position: unlike RL methods, they require no reward function design; unlike Self-Refine, they propagate structured credit through multi-component systems.

Synthesis of A2 without tools. The methods above map onto a two-dimensional design space. Along the first axis (reward granularity), scalar final-answer correctness (DeepSeek-R1) is broadly applicable but sparse, while structured linguistic feedback (TextGrad) is information-rich but requires a capable critic model. Along the second axis (optimization locus), weight-update methods (RL, SFT) amortize adaptation cost across future queries, while inference-time methods (Self-Refine, TextGrad) pay per-query cost but require no training infrastructure. A recurring finding across both axes is that SFT warm-up followed by RL yields more stable training than RL alone: SFT provides domain grounding, while RL refines the policy toward the evaluation objective.

4.2.2 Agent Adaptation w/ Tools

When tools enter the A2 loop, the agent must learn a *meta-policy*: not only how to reason but when to invoke a tool, which tool to choose, and how to integrate tool outputs back into the reasoning chain. The reward is still computed on the agent's final output, but the policy space is much larger, making credit assignment and data efficiency the central challenges.

Retrieval-based tool learning. Prior distillation-SFT approaches such as **Self-RAG** [104] (ICLR 2024) and its successors [105–107] teach retrieval tool use from expert demonstrations; RL-based methods instead let the agent discover search strategies on its own. **R1-Searcher** [108], **Search-R1** [50] (COLM 2025), **ReSearch** [109] (NeurIPS 2025), and their successors [110–116] all train LLMs to autonomously generate and refine search queries during multi-turn reasoning. The distinguishing design choice across this family is the *action representation*: R1-Searcher incentivizes search API calls via a two-stage RL framework (up to 24% over RAG baselines), Search-R1 formulates search invocation as a joint reward over retrieved evidence and final correctness, and ReSearch integrates queries and results directly into the reasoning chain via `<think>`/`<search>`/`<result>` tags, optimized with GRPO. ReSearch is notable for exhibiting emergent reflection and self-correction behaviors—strategic capabilities that were not explicitly supervised but arose from holistic task optimization.

Code- and execution-based tool learning. Code execution provides a natural bridge between A1 and A2: the execution outcome is verifiable (A1-like), but the training signal evaluates the agent's overall problem-solving trajectory (A2-like). **CodePRM** [117] (ACL 2025) scores intermediate reasoning steps via a process reward model trained on code execution results, forming a Generate-Verify-Refine pipeline that corrects errors during inference. **ReTool** [51] integrates real-time code execution directly into RL rollouts, so the agent learns when to invoke an interpreter to offload computation—illustrating how A2 training can discover tool-use strategies that A1 training cannot, because the reward depends on whether the agent's overall answer (not just the tool call) is correct.

General multi-tool and agentic learning. In the most general setting, agents interact with heterogeneous APIs and environments. Three recurring themes organize the methods below: *data generation*, *self-reflection*, and *infrastructure*.

On the data side, **Self-Challenging Agents** [118] introduce a self-generated curriculum—the model first creates tool-use tasks as a challenger, then solves them via RL as an executor—achieving over twofold improvement on multi-turn benchmarks. **Re-ReST** [119] uses environment feedback (e.g., unit-test results) to refine low-quality trajectories through reflection, yielding large gains on HotpotQA and AlfWorld. On the self-reflection side, **Agent-R** [120] formalizes iterative self-correction via model-guided critique and MCTS rollouts (+5.6% across interactive environments), and **Test-Time Self-Improvement** [121] performs on-the-fly fine-tuning at inference time on

self-identified uncertain cases. On the infrastructure side, **Agent Lightning** [122] decouples agent execution from training to support complex multi-agent workflows, **A^{2}FM** [123] unifies reasoning and acting under cost-regularized RL (dynamically choosing between thinking, tool use, and direct answering), and **VerlTool** [124] provides asynchronous rollout infrastructure that eliminates synchronization bottlenecks in agentic RL.

Synthesis of A2 with tools. A2 methods with tools share a defining characteristic: the agent must learn not only *how* to use tools (also targeted by A1) but *when* to use them and how to fold their outputs into a coherent reasoning chain. This strategic dimension explains why A2 training frequently produces emergent behaviors—self-correction, adaptive search depth, query refinement—that are never explicitly supervised. The cost is data efficiency: the sparse, episode-level reward requires either large training sets (e.g., Search-R1's ∼170k examples) or carefully designed curricula (e.g., Self-Challenging Agents' self-generated tasks). The tension between A1's per-action precision and A2's end-to-end strategic optimization is analyzed quantitatively in §6.

5 Tool Adaptation

Tool adaptation shifts the optimization target from the agent itself to its ecosystem. Instead of modifying the agent's parameters through fine-tuning or reinforcement learning, this paradigm targets the external components (pre-trained models, retrievers, planners, or executors) that the agent invokes through language or code. Methods in this category (1) employ pre-trained machine learning models as plug-and-play components, ranging from simple classifiers to complex LLM-based subagents as discussed in §4; or (2) use the agent's outputs as supervision or reinforcement signals to train, align, or refine the tool itself.

Formally, let $\mathcal{A}$ denote an agent, parameterized by its internal configuration or policy (which include prompt templates or model weights) and $\mathcal{T}$ denote a tool or a set of tools that can be trained or optimized based on task feedback. The adaptation process can be characterized by two complementary paradigms:

$$\textbf{(T1)} \quad \mathcal{T}^* = \arg\max_{\mathcal{T}} \mathcal{O}_{\text{tool}}(\mathcal{T}), \qquad \textbf{(T2)} \quad \mathcal{T}^* = \arg\max_{\mathcal{T}} \mathcal{O}_{\text{agent}}(\mathcal{A}, \mathcal{T}),$$

where $\mathcal{O}_{\text{tool}}$ quantifies task-specific or environment-driven improvements that are independent of the agent, such as retrieval accuracy or planning efficiency, while $\mathcal{O}_{\text{agent}}$ incorporates agent-derived supervision, where the agent's outputs provide learning signals to refine or align the tool. Here, $\mathcal{T}^*$ denotes the optimized tool configuration that maximizes the respective objective, illustrating how tool adaptation complements agent-level optimization within the broader agent-tool ecosystem.

5.1 T1: Agent-Agnostic Tool Adaptation

The foundational architecture for tool-augmented systems uses pre-trained models as plug-and-play tools for frozen agents. The agent orchestrates tool usage through prompting alone, never updating its parameters, while relying on tools trained independently on diverse data sources before deployment.

While T1 is the most straightforward paradigm (any pre-trained model an agent can invoke qualifies), it provides the compositional substrate on which all other paradigms build: A1/A2 agents invoke T1 tools, T2 subagents are often initialized from T1 components, and the "graduation lifecycle" (A1→T1, discussed in §6.3.1) continually enriches the T1 ecosystem. Understanding T1's design space is therefore essential for the entire framework.

5.1.1 Foundational Systems and Architectures

Early systems established the architectural foundations for how frozen agents orchestrate external models. These works illustrate distinct mechanisms (functional, prompt-based, code-based, and graph-based) that shaped the design space for modern tool-augmented AI systems.

Operator-Learning Tools: Neural Operators [125] (JMLR). Before large-scale LLM-based orchestration emerged, Neural Operators represented an early example of *agent-agnostic tool learning*: models trained to approximate mappings between infinite-dimensional function spaces, serving as differentiable surrogates for complex simulators.

Unlike conventional neural networks tied to discrete grids, Neural Operators are **discretization-invariant**: they learn the underlying operator itself, not its finite discretization, and can generalize across resolutions and geometries. The **Fourier Neural Operator (FNO)** achieves $\mathcal{O}(J \log J)$ inference via spectral convolution and outperforms classical solvers on Navier-Stokes, Darcy flow, and elasticity equations by orders of magnitude in speed. FNO and its variants (Graph-, Low-rank-, Multipole-NO) represent the first wave of "frozen tools" that agents can query repeatedly for reasoning, planning, or control without retraining. In modern agentic pipelines, they serve as plug-in surrogates: fast, differentiable black-box functions invoked within decision or inference loops.

HuggingGPT [126] (NeurIPS 2023) introduced the orchestration paradigm by allowing ChatGPT to command 1000+ machine learning models from HuggingFace Hub without any fine-tuning. The frozen LLM executes a four-stage workflow: task planning (decomposing user requests), model selection (choosing from tool descriptions), task execution (invoking models), and response generation (synthesizing outputs). The architecture shows that tool descriptions in natural language suffice for the frozen agent to coordinate complex multimodal workflows. On composite cross-modal tasks, HuggingGPT demonstrates that orchestrating specialized vision, speech, and language models can close the gap between GPT-3.5 and much larger multimodal models. The primary limitation lies in latency from sequential LLM calls and token length constraints for tool descriptions. **ViperGPT** [127] (ICCV 2023) introduced code generation as the orchestration mechanism. The frozen GPT-3 Codex generates Python code that composes vision models (GLIP for detection, SAM for segmentation, MiDaS for depth estimation) into executable programs. The code-based approach achieves strong zero-shot performance on GQA visual reasoning, outperforming end-to-end models by 10–15% on compositional tasks. Python functions provide more flexible tool composition than fixed API calls. Each tool exposes simple functions like `find(image, object_name)` or `compute_depth(image)`, which Codex chains programmatically without learning tool-specific interfaces. **SciToolAgent** [128] (Nature Computational Science 2025) scales tool orchestration to scientific domains through graph-based organization. The frozen GPT-4o accesses 500+ biology, chemistry, and materials science tools via SciToolKG, a knowledge graph encoding tool metadata, dependencies, and safety constraints. Graph-based retrieval for tool selection achieves 94% accuracy on scientific query benchmarks, representing a 15–20% improvement over GPT-4o without tool access. The system successfully automates protein engineering workflows chaining ESMFold for structure prediction, BLAST for sequence alignment, and custom analysis tools. Structured knowledge graphs address the scalability challenges inherent in prompt-based descriptions.

These foundational systems illustrate the dominant integration patterns. HuggingGPT exemplifies **prompt-based orchestration**, where the agent parses tool calls from text. ViperGPT uses **code generation**, exposing tools as Python functions. SciToolAgent uses **knowledge graph retrieval**, using RAG to select from structured tool graphs. A fourth common pattern is **multimodal bridging**, which converts non-textual modalities into text representations; for example, Visual ChatGPT's [129] prompt manager serializes vision operations as text API calls. The usability of these patterns depends on clear **interface design**, such as programmatic function signatures (e.g., `find_object(image: PIL.Image, ...)`), structured JSON schemas, or simple natural language descriptions. **Execution modes** are similarly varied, ranging from direct API calls and code generation to HTTP requests and command-line invocations.

Model Context Protocol (MCP) and Code Execution Environments [130]. As large-scale agent ecosystems began connecting thousands of heterogeneous tools, the *Model Context Protocol* (MCP) emerged as an open standard for unifying how agents interface with external systems. MCP provides a universal API layer that allows frozen agents to discover, invoke, and coordinate tools across domains using a consistent schema, rather than embedding long tool definitions directly into the model's context. Anthropic's "*Code Execution with MCP*" paradigm introduced an execution-centric design in which the agent writes executable code to interact with MCP servers instead of performing token-level tool calls. Agents can thereby load only the necessary tool definitions, filter or aggregate data within a sandboxed environment, and pass compact results back to the model, substantially reducing context usage while preserving compositionality. MCP represents a scalable **T1-style tool adaptation infrastructure** that decouples execution from inference, while the code-execution mode bridges toward **T2-style optimization** by dynamically improving efficiency under frozen agents.

5.1.2 Categories and Training Methodologies

While the preceding systems define *how* agents invoke tools, the choice of *which* tool modalities to deploy shapes the entire adaptation surface.

Tool adaptation encompasses a range of pre-trained model categories. We organize these by modality and highlight how their training methodology determines their amenability to T2 optimization: models producing structured, evaluable outputs (code, retrieval results) are most naturally upgraded to T2, because the frozen agent can score their outputs automatically; models producing perceptual features (vision, speech) are harder to supervise via T2 and remain predominantly T1 plug-ins.

Vision models dominate T1 deployments as plug-and-play tools. CLIP [131], trained contrastively on 400M image-text pairs, provides zero-shot classification and semantic understanding through frozen encoders. SAM [132], trained on 11M images with 1B masks via human-in-the-loop data engines, provides promptable segmentation with point, box, or mask inputs. SAM-CLIP [133] merges these capabilities through multi-task distillation with frozen teachers, achieving +6.8% mIoU on Pascal VOC for zero-shot semantic segmentation while retaining both parent models' strengths. These vision tools require no task-specific fine-tuning - frozen agents invoke them directly via APIs for image understanding, segmentation, and classification tasks.

Speech and audio tools rely on massive pre-training for robust performance. Whisper [134], trained on 680K hours of multilingual audio with weak supervision, provides speech recognition, translation, and language identification as a frozen API for multimodal agents. The encoder-decoder Transformer architecture supports zero-shot transcription across languages and domains, with strong robustness to accents, noise, and technical terminology. Agents simply pass audio inputs to the frozen Whisper model and process text outputs without any model adaptation.

Code execution tools encompass models that learn to compose and execute functions through code. CodeAct [58] shows that representing tool use in executable Python rather than static JSON improves compositional reasoning, achieving over 20% higher success rates on API-Bank benchmarks. The dynamic nature of code allows agents to flexibly construct, parameterize, and combine tools without predefined schemas.

Search and retrieval tools comprise pre-trained dense retrievers such as DPR [135], ColBERT [136], Contriever [137], and e5 [138], often deployed as frozen components within retrieval-augmented generation pipelines. These bi-encoder models, trained on passage ranking tasks, support semantic search over large corpora.

Scientific tools extend capabilities to specialized fields. AlphaFold2 [139] and ESMFold [140] provide protein structure prediction from sequences. Materials science models like CGCNN [141] predict crystal properties. Molecule representation learning approaches [142–148] have been developed for molecular property prediction, whereas some encoder-decoder frameworks [149, 150] aim to predict transcriptional profiles elicited by chemical perturbations. These tools represent years of domain-specific model development, deployed as-is for frozen agents tackling scientific queries.

Beyond these static models, adaptive agents introduced in §4 (such as DeepRetrieval [22] for search query rewriting and Code-R1 [66] for code generation) illustrate how trained reasoning agents themselves can function as dynamic tools. Once frozen, they extend the tool ecosystem by reformulating queries, generating executable code, or performing reasoning-driven actions, thereby bridging the gap between pre-trained models and the environment or offline data.

5.2 T2: Agent-Supervised Tool Adaptation

The T2 paradigm inverts the standard adaptation question: rather than asking "how can we modify the agent to better use its tools?" (the A1/A2 question), T2 asks "how can we modify the tools to better serve a fixed agent?" This reframes the expensive, monolithic foundation model as a stable source of supervision rather than the target of optimization.

Training or fine-tuning billion-parameter foundation models is computationally expensive and risks catastrophic forgetting. Peripheral tools (retrievers, planners, memory modules) are orders of magnitude smaller and cheaper

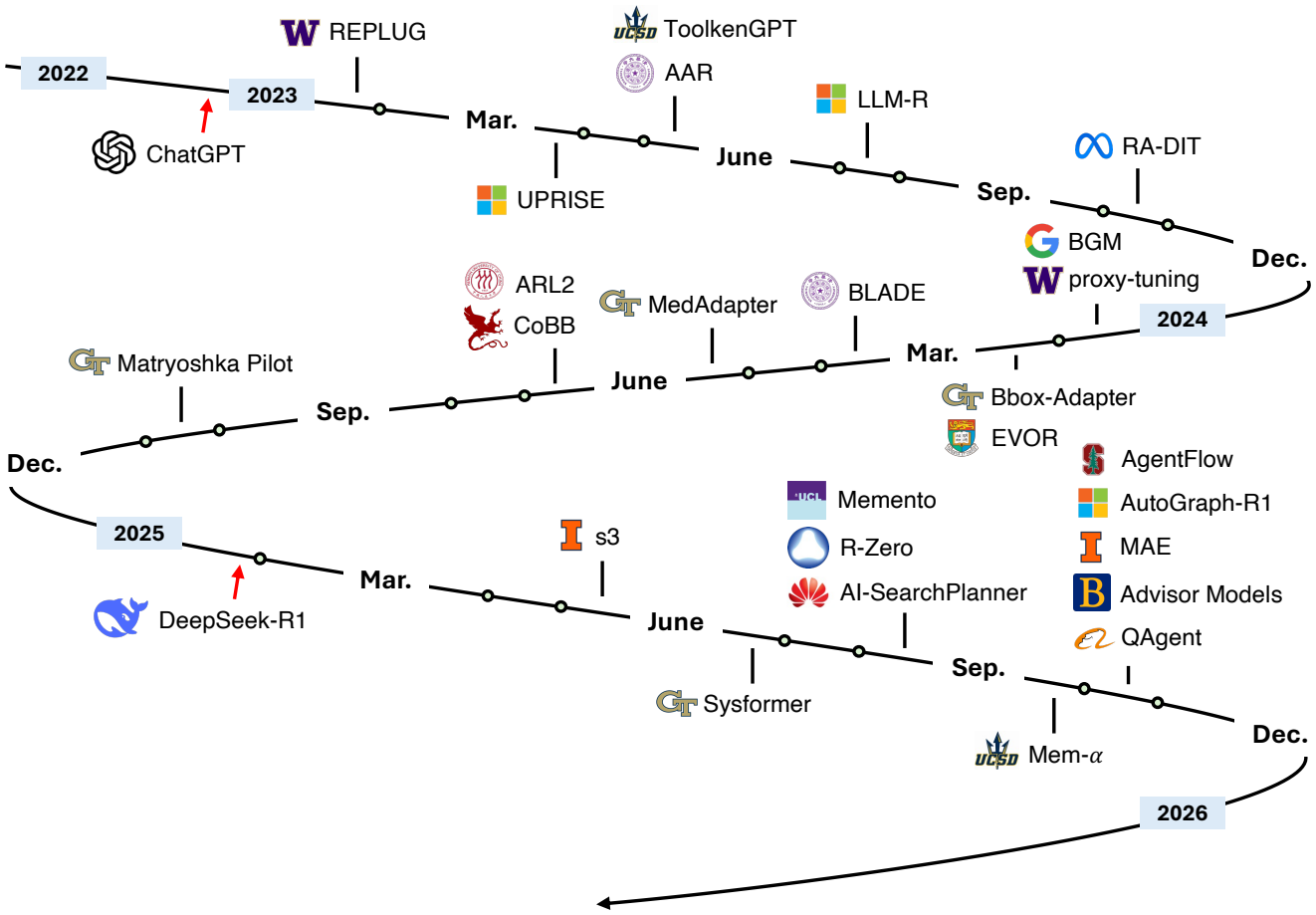

Figure 6 Development timeline of T2 methods (agent-supervised tool adaptation, classic memory-related methods are not included in this figure due to space limitation).

to train. T2 exploits this asymmetry: the frozen agent provides supervision signals derived from its pre-trained knowledge, while the tools learn to reshape information into the form the agent can best exploit.

The evolution of T2 methods from 2023 to 2025 shows a progression from using internal proxy signals (perplexity, preferences) to train passive retrieval tools, to using verifiable outcome signals (task success, accuracy gains) to train active, multi-turn agentic tools.

Table 3 T2 Methods: Tool Adaptation w/ Agent Supervision

Time	Method	Venue	Task(s)	Tool Backbone	Agent Backbone	Tuning	Links
			Earlier Methods				
2023.01	REPLUG	NAACL'24	QA	Contriever	GPT3-175B, PaLM, Codex, LLaMA-13B	Proxy-Tuning, LSR	
2023.03	UPRISE	EMNLP'23	Zero-shot NLU (QA, NLI, etc.)	GPT-Neo-2.7B	BLOOM-7.1B, OPT-66B, GPT-3-175B	Contrastive Learning	
2023.05	ToolkenGPT	NeurIPS'23	Numerical Reasoning, QA, Plan Generation	Token Embedding	GPT-J 6B, OPT-6.7B, OPT-13B	Proxy-Tuning	

Continued on next page

Table 3 – Continued from previous page

Time	Method	Venue	Task(s)	Tool Backbone	Agent Backbone	Tuning	Links
2023.05	AAR	ACL'23	Zero-Shot Generalization (MMLU, PopQA)	ANCE, Contriever	Flan-T5-Small, InstructGPT	Contrastive Learning	
2023.06	LLM-R	EACL'24	Zero-shot NLU (QA, NLI, Paraphrase, etc.)	E5-base	GPT-Neo-2.7B, LLaMA-13B, GPT-3.5-Turbo	Contrastive Learning	
2023.10	RA-DIT	ICLR'24	Knowledge-Intensive Tasks (MMLU, NQ, TQA, ELI5, HotpotQA, etc.)	DRAGON+	LLaMA-65B	SFT, LSR	
2024.01	BGM	ACL'24	QA	T5-XXL-11B	PaLM2-S	SFT, PPO	
2024.01	Proxy-Tuning	COLM'24	QA, Math, Code	LLaMA2-7B	LLaMA2-70B	Proxy-Tuning	
2024.02	Bbox-Adapter	ICML'24	QA	DeBERTa-v3-base (0.1B), DeBERTa-v3-large (0.3B)	GPT-3.5-Turbo, Mixtral-8x7B	Contrastive Learning	
2024.02	EVOR	EMNLP'24	Coding	GPT-3.5-Turbo	GPT-3.5-Turbo, CodeLLaMA	Prompt Engineering	
2024.02	ARL2	ACL'24	QA	LLaMA2-7B	GPT-3.5-Turbo	Contrastive Learning	
2024.03	BLADE	AAAI'25	QA	BLOOMZ-1b7	ChatGPT, ChatGLM, Baichuan, Qwen	SFT, BPO	
2024.05	Medadapter	EMNLP'24	Medical QA, NLI, RQE	BERT-Base-Uncased	GPT-3.5-Turbo	SFT, BPO	
2024.06	CoBB	EMNLP'24	QA, Math	Mistral-7b-inst-v2	GPT-3.5-Turbo, Claude-3-Haiku, Phi-3-mini-4k-inst, Gemma-1.1-7B-it, Mistral-7B-inst-v2	SFT, ORPO	
2024.10	Matryoshka Pilot	NeurIPS'25	Math, Planning, Reasoning	LLaMA3-8B, Qwen2.5-7B	GPT-4o-Mini, GPT-3.5-Turbo	DPO, IDPO	
2025.06	Sysformer	arXiv	QA	Small Transformer	LLaMA-2-7B, LLaMA-3.1-8B, Mistral-7B, Phi-3.5-mini, Zephyr-7B-beta	Supervised Learning	
RLVR Methods							
2025.05	s3	EMNLP'25	QA	Qwen2.5-7B	Qwen2.5-7B, Qwen2.5-14B, Claude-3-Haiku	PPO	
2025.08	R-Zero	arXiv	Math, Reasoning	Qwen3-4B, Qwen3-8B, OctoThinker-3B, OctoThinker-8B	Qwen3-4B, Qwen3-8B, OctoThinker-3B, OctoThinker-8B	GRPO	

Continued on next page

Table 3 – Continued from previous page

Time	Method	Venue	Task(s)	Tool Backbone	Agent Backbone	Tuning	Links
2025.08	Memento	arXiv	Long-Horizon Reasoning, Web Research, QA, Academic Reasoning	Q-function (two-layer MLPs)	GPT-4.1	Soft Q-Learning	
2025.08	AI-SearchPlanner	arXiv	QA	Qwen3-32B	Qwen2.5-7B	PPO	
2025.09	Mem-α	arXiv	Test-Time Learning, Long-Range Understanding	Qwen3-4B	Qwen3-4B, Qwen3-32B, GPT-4.1-Mini	GRPO	
2025.10	AgentFlow	arXiv	Web Search, Planning, Reasoning, Math	Qwen2.5-7B	Qwen2.5-7B	Flow-GRPO	
2025.10	AutoGraph-R1	arXiv	KG Construction	KG Constructor (Qwen2.5-3B/7B)	Frozen RAG Generator (Qwen2.5-7B)	GRPO	
2025.10	MAE	arXiv	Math, Coding, QA	Qwen2.5-3B	Qwen2.5-3B	REINFORCE+	
2025.10	Advisor Models	arXiv	Math, Reasoning	Qwen2.5-7B, Qwen3-8B	GPT-4o-Mini, GPT-5, Claude4-Sonnet, GPT-4.1-Mini	GRPO	
2025.10	QAgent	arXiv	QA	Qwen2.5-3B	Qwen-7B	GRPO	

5.2.1 Earlier Methods: From Proxy Signals to Structured Preferences

The earliest T2 methods emerged from the RAG community, where researchers sought to optimize dense retrievers for compatibility with large language models. These works established the principle that a frozen LM's internal computations could serve as supervision, but they also reveal the limitations of relying on proxy metrics that may not align with downstream task objectives.

REPLUG [11] (NAACL 2024) introduced a general framework for adapting frozen language models through black-box supervision, using perplexity reduction as a training signal for the retriever. If conditioning the LM on a retrieved document lowers its perplexity for a given query, the document likely provides informative context. Formally, the retriever is optimized to align its retrieval distribution with the distribution induced by the LM's perplexity-based preferences:

$$\mathcal{L}_{\text{REPLUG}} = D_{\text{KL}}(P_{\text{retriever}}(d|q) \,\|\, P_{\text{LM-perplexity}}(d|q)),$$

where $P_{\text{LM-perplexity}}(d|q)$ reflects how strongly each document reduces the LM's perplexity when conditioned on query q. REPLUG thus enabled retrieval adaptation without parameter access to the LM, establishing a family of agent-supervised methods that optimize external modules based solely on frozen-agent feedback. **BLADE** [151] (AAAI 2025) further extended this paradigm by replacing traditional retrievers with domain-specific models that synthesize auxiliary knowledge: it couples a frozen general LLM with a small, domain-specific LM optimized via Bayesian Prompted Optimization (BPO). The small LM learns to generate domain-relevant knowledge and soft prompts that improve the black-box LLM's responses, extending REPLUG's black-box adaptation principle from retrieval to generative, domain-specialized tool co-adaptation. **BBox-Adapter** [152] (ICML 2024) reframed adaptation as an energy-based modeling problem, introducing a ranking-based noise-contrastive estimation loss and an online update framework to align outputs from black-box APIs like GPT-3.5 without access to internal probabilities. These methods collectively shifted the focus from likelihood-based alignment to utility-driven adaptation, anticipating the reinforcement-learning-based search and reasoning frameworks that followed. **proxy-tuning** [153] (COLM 2024) extends black-box, agent-supervised adaptation to decoding time: a small tuned

"expert" and its untuned "anti-expert" provide logit offsets that steer a frozen large LM without modifying its weights, effectively training a lightweight steering *tool* under a frozen agent (T2). **EVOR** [154] (EMNLP 2024) further extends to the domain of code generation, formulating retrieval and knowledge evolution as a co-adaptive process driven by execution feedback from a frozen LLM.

Preference Learning: Toward Task Alignment. While black-box adaptation methods such as REPLUG and BBox-Adapter relied on indirect proxy signals like perplexity or ranking scores, subsequent studies [29, 155–158] moved toward explicit preference-based supervision that better reflects task utility. **AAR** [29] (ACL 2023) introduced augmentation-aware retrieval, where a frozen source LM (Flan-T5-250M) constructs preference pairs by comparing documents that most improve its own likelihood against human-annotated references. The retriever is then trained to reproduce these preferences via a contrastive loss, yielding a signal that directly encodes the LM's notion of "helpful context." These LM-derived preferences transfer effectively across architectures and scales, improving even 175B-parameter models. **RA-DIT** [155] (ICLR 2023) formalized this idea by defining document utility as the log-probability gain for producing correct answers:

$$\text{Utility}(d, q, a) = \log P_{\text{LM}}(a|q, d) - \log P_{\text{LM}}(a|q),$$

training retrievers to prefer documents that yield higher expected gains. The preference-based approach aligns retrieval more directly with downstream reasoning objectives while still operating through a single forward pass of the frozen LM. Together, these works mark a conceptual shift from proxy-based to task-aligned supervision, setting the stage for reinforcement-style feedback and multi-turn optimization in later frameworks.

Multi-stage architectures: distilling complex preferences. **LLM-R** [24] (EACL 2024) introduced a multi-stage distillation pipeline that increases the fidelity of training signals. Rather than training the retriever directly on the frozen LM's outputs, LLM-R first trains an intermediate cross-encoder "reward model" to capture the frozen LM's nuanced preferences over in-context examples. The reward model, which can afford to be slow and expressive because it is used only during training, is then distilled into a fast bi-encoder retriever. The key insight is that the complexity of the training signal need not be constrained by inference-time efficiency requirements. By decoupling the preference modeling from the final retrieval tool, LLM-R achieves both high-quality supervision and fast deployment.

UPRISE [159] (EMNLP 2023) extends this paradigm beyond documents to prompts, training a prompt retriever using the frozen LLM's task performance across diverse tasks. By training on multiple tasks simultaneously, UPRISE learns a generalizable meta-skill: selecting prompts that improve LLM performance in zero-shot settings. The cross-task transfer (+8.5% on reading comprehension, +14.6% on paraphrase detection) suggests that T2-trained tools can internalize abstract principles of "what helps an LM" rather than memorizing task-specific heuristics.

By 2024, a consensus emerged that optimizing retrieval in isolation is insufficient. Even a retriever that scores well on traditional IR metrics (NDCG, MRR) may produce results poorly suited for LLM reasoning. The observation motivated research on bridge tools: rerankers, query reformulators, and document selectors that align retrieval outputs with LLM preferences.

The architecture of preference translation. **BGM** [160] addresses the systematic "preference gap" between what traditional retrievers optimize for (surface-level relevance, lexical overlap) and what LLMs find useful for reasoning (contextual coherence, inferential support). BGM addresses this by training a T5-XXL "bridge model" that sits between a frozen retriever and a frozen generator (PaLM2-S), transforming the retriever's output into an LLM-friendly context.

Stage 1 uses supervised learning on synthetically generated preference data (documents that improve LLM task performance vs. those that do not), while Stage 2 employs reinforcement learning where the frozen LLM's final task success provides the reward signal. On HotpotQA, the bridged system achieves 35.6% exact-match accuracy compared to 25.8% with the best prior retriever, a relative improvement of 38%.

BGM shows that specialized adaptation layers can be more effective than end-to-end fine-tuning. Rather than making a single retriever simultaneously satisfy IR metrics and LLM preferences, BGM decomposes the problem:

the retriever handles broad recall, while the bridge model handles preference alignment. The same modularity recurs in the most successful T2 systems.

Synthesis: the multi-tool ecosystem. These advances reveal an architectural pattern: cascaded tool adaptation. State-of-the-art T2 systems now employ a pipeline of specialized tools (query reformulators, retrievers, selectors), each trained using different aspects of the frozen LLM's behavior as supervision. The decomposition offers several advantages:

- **Separation of concerns:** Each tool can optimize a specific sub-problem (recall vs. precision, speed vs. quality).
- **Composability:** Tools can be mixed and matched; a new reranker can be trained without retraining the retriever.
- **Efficiency:** Expensive operations (LLM inference) are deferred to the end of the pipeline, after cheaper tools have filtered the space.

An open question is how deep this hierarchy can go before compounding errors overwhelm the benefits. Empirical results suggest that 2–3 stages of tool adaptation (e.g., query reformulator $\rightarrow$ retriever $\rightarrow$ reranker) strike a good balance.

5.2.2 Subagent-as-Tool

The year 2025 marked a transition in T2 research from training reactive tools (retrievers that respond to queries) to training proactive subagents (autonomous systems that explore, plan, orchestrate, and refine their operations over multiple turns while serving frozen primary agents). The shift, enabled by advances in RLVR, applies not only to information retrieval but also to meta-cognitive processes such as workflow orchestration and memory management. The evolution can be organized into four families of subagents: (i) agentic searchers, (ii) memory-construction subagents, (iii) meta-cognitive planners and orchestrators, and (iv) self-evolving subagents.

Agentic searchers. **s3** [28] (EMNLP 2025) showed that training agentic tools can be radically more data-efficient than training agentic LLMs. The system trains a lightweight 7B "searcher" that performs multi-turn iterative search: generate queries, retrieve documents, select evidence, decide whether to search again or feed context to the frozen generator. The frozen generator (agent, Qwen2.5-14B or Claude) never updates, but provides the ultimate training signal through a metric called Gain Beyond RAG (GBR):

$$\text{GBR} = \text{Accuracy}\left(\mathcal{G}_{\text{frozen}}(q, D_{s3}), a\right) - \text{Accuracy}\left(\mathcal{G}_{\text{frozen}}(q, D_{\text{naive}}), a\right)$$

where D_{s3} are documents retrieved by the trained searcher and D_{naive} are documents from naive top-k retrieval.

The reward directly measures the value added by the search tool, focusing training on examples where naive retrieval fails. s3 achieves 58.9% average generation accuracy with only *2.4k training samples*, 70$\times$ less data than Search-R1 (an A2-style agent requiring 170k examples) and 33$\times$ faster wall-clock training time. On specialized medical QA, s3 trained on general QA reaches 76.6% accuracy versus 71.8% for Search-R1, indicating that T2-trained tools learn more **generalizable search skills** than agents trained end-to-end. The efficiency advantage arises because A2-style agent training must simultaneously learn (1) domain knowledge, (2) tool-use skills, and (3) task-specific reasoning, creating a high-dimensional optimization landscape. In T2, the frozen generator already possesses domain knowledge and reasoning ability, so the tool need only learn the procedural skill of effective search.

Similar to this idea, **DynamicRAG** [161] (NeurIPS 2025) "agentifies" reranking: instead of static reorderings, an RL policy adapts how many and which documents to pass based on query hardness and retrieval noise, balancing quality and context cost. The training combines imitation learning on expert trajectories (to bootstrap reasonable behavior) with policy gradient RL where the generator's output provides the reward signal. The learned policy exhibits emergent adaptive behavior: for simple queries with high-quality initial retrieval, it presents fewer documents; for complex queries with noisy retrieval, it retrieves more broadly and reranks more aggressively.

QAgent [162] further clarifies how to robustly train such search subagents. Its Stage 1 trains a 3B search agent end-to-end, rewarding it based on whether its own generated answer is correct, but this encourages reward hacking, where the agent prefers shallow, easily copyable evidence over genuinely informative documents. Its Stage 2 corrects this by switching to evaluation from a stronger frozen generator:

$$R_{\text{Stage2}} = \mathbb{I}\left[\mathcal{G}_{\text{frozen}}(q, D_{\text{agent}}) = a_{\text{correct}}\right],$$

rewarding the searcher only when the frozen model can answer correctly using its retrieved documents. The decoupling forces the subagent to optimize for retrieval quality rather than its own myopic behavior, reinforcing a core T2 principle: the frozen generator should not only consume a tool's outputs but also supervise its learning.

Learning to construct memory as a subagent. Extending beyond search, a complementary line of work treats long-term memory construction itself as a T2-style subagent problem. Mem-α [163] formulates memory management as RL over an explicit memory API, training a lightweight Qwen3-4B controller to operate a three-part external memory (core summary, semantic facts, episodic events) for a frozen backend generator. Only the memory-writing policy is optimized; the generator and retriever for downstream QA remain frozen. Rewards derive from verifiable outcomes (question-answering accuracy over long horizons, tool-call correctness, effective compression, and semantic validity of memory entries), so the subagent learns to construct compact yet sufficient memories that maximize the frozen model's utility. In experiments, Mem-α outperforms prior memory baselines and generalizes from $\sim$30k-token training sequences to contexts exceeding 400k tokens, serving as a concrete instance of a memory-construction subagent that adaptively curates the information diet for a fixed reasoning core.

AutoGraph-R1 [164] applies this symbiotic principle to the construction of structured Knowledge Graphs (KGs). Rather than relying on static extraction heuristics, it optimizes an LLM-based "constructor subagent" to generate KGs from raw text. The supervision signal is derived directly from the frozen agent's performance on downstream reasoning tasks (GraphRAG [165]) using the generated graph. The constructor thereby learns a policy that prioritizes functional utility, creating connectivity and paths that specifically facilitate the host agent's retrieval and reasoning, over intrinsic metrics like triple density.

Meta-cognitive and control subagents. Recent work trains subagents that shape how frozen models think (planning, steering, and budgeting computation) rather than what they retrieve or store.

AI-SearchPlanner [27] introduces multi-objective optimization that balances effectiveness with efficiency. The system trains a planner tool (Qwen2.5-7B) that generates multi-step search strategies for a frozen generator, optimizing

$$\mathcal{J} = \mathbb{E}\left[R_{\text{outcome}} + \lambda \cdot R_{\text{process}} - \alpha \cdot \text{Cost}\right],$$

where R_{outcome} measures final task success, R_{process} evaluates the rationality of the search plan (critiqued by the frozen generator), and Cost penalizes excessive planning. By combining both outcome and process rewards [166], the frozen model acts as executor and teacher, so the planner internalizes not only "what works" but "why it works." Varying λ traces a Pareto frontier between cost and quality, yielding planners tailored to different deployment budgets.

Advisor Models [167] generalize this idea to instance-wise natural-language steering. A small advisor model learns, via GRPO, to prepend context-specific advice that nudges a frozen foundation model toward preferred behaviors (style, safety, reasoning depth) without touching its weights. Within our taxonomy, such advisors function as trainable control interfaces or parametric memories that encode environment- and user-specific latents.

Bridging from advising to driving, **Matryoshka Pilot** [168] (NeurIPS 2025) formalizes a controller-generator loop where a small white-box LLM controls a larger black-box LLM by emitting intermediate decomposition steps, plans, and summaries. Treating the black-box model as an environment, M-Pilot collects trajectory-level success signals and optimizes the controller with Iterative DPO. The method yields $\approx$3–7% gains across reasoning, planning, and personalization benchmarks, and transfers plug-and-play across multiple black-box backends, further reinforcing the view of control subagents as portable T2 tools.

Learning to orchestrate frozen specialists. Orchestration-focused subagents train a dedicated policy to coordinate multiple frozen specialists.

AgentFlow [52] decomposes an agent into modules (planner, tool executor, verifier, and solution generator) implemented mostly as frozen Qwen2.5-7B-Instruct models. Only the planner is trained. Using Flow-GRPO, a single trajectory-level reward (correct vs. incorrect, judged by GPT-4o) is broadcast to all decisions in each rollout, with group-normalized advantages permitting effective credit assignment despite sparse rewards. A 7B AgentFlow planner achieves 57.3% on search-intensive tasks (+14.9% over AutoGen), 51.5% on mathematical reasoning (+14.5% over ToRL), and 33.1% on GAIA, outperforming the much larger GPT-4 on several setups, demonstrating that learned orchestration of frozen specialists can rival or surpass monolithic models.

Self-evolving (sub)agent. A more advanced branch of the subagent-as-tool paradigm allows the tools themselves to co-evolve through self-generated tasks and rewards. **R-Zero** [169] instantiates two roles, a Solver and a Challenger, from the same base LLM. When the Solver is frozen, its successes, failures, and uncertainty (via self-consistency) define rewards that train the Challenger to propose tasks near the Solver's capability frontier. Alternating these phases creates a bidirectional loop, yet each step still follows the T2 principle of optimizing a lightweight subagent under signals from a stronger or temporarily fixed core. **Multi-Agent Evolve (MAE)** [170] extends this design into a triadic architecture with a Proposer, Solver, and Judge. The Proposer and Judge operate as adaptive T2 subagents: the Judge learns to evaluate trajectories produced by the system, and the Proposer learns to generate diverse, high-quality, Solver-challenging tasks. Rather than tuning the main Solver, MAE improves performance by training these peripheral subagents to shape data, rewards, and curricula. Together, R-Zero and MAE illustrate a second generation of subagent-as-tool methods: self-evolving ecosystems that autonomously construct the learning conditions for otherwise frozen reasoning cores.

Synthesis: the maturation of T2. The subagent-as-tool paradigm progressively broadens the scope of T2: retrieval, memory construction, planning and orchestration, and finally self-evolution. Agentic searchers (s3, DynamicRAG, QAgent) optimize information acquisition for frozen generators; memory-construction subagents (Mem-α) curate long-horizon state; meta-cognitive controllers and orchestrators (AI-SearchPlanner, Advisor Models, Matryoshka Pilot, AgentFlow) decide how tools and specialists are deployed; and self-evolving frameworks (R-Zero, Multi-Agent Evolve) autonomously generate curricula and reward signals. Decoupling tool training from generator training, while allowing tools to adapt to one another, yields systems that are more data-efficient, modular, and generalizable than monolithic alternatives.

5.2.3 Agentic Memory and Skills

An agent's memory system can be framed as an adaptive tool. As discussed in §3, the paradigm label depends on the memory's form and update mechanism: external non-parametric stores updated by a frozen agent's outputs are predominantly T2; pre-trained or plug-in modules are T1; and parametric or hybrid architectures occupy the boundary between tool and agent adaptation. The majority of systems reviewed below are T2, where the frozen agent's downstream task performance or outputs serve as the supervisory signal for tuning the memory module (how it writes, retrieves, reflects, and forgets). Several recent surveys provide complementary perspectives on this space. Hu et al. [171] propose a unified *forms, functions, dynamics* framework that organizes agent memory along three orthogonal dimensions: **forms** (token-level, parametric, and latent memory, distinguished by where and how information is stored), **functions** (factual memory for knowledge of users and environments, experiential memory for accumulated interaction outcomes, and working memory for active context management), and **dynamics** (the lifecycle of memory formation, evolution, and retrieval). This tripartite lens clarifies that memory adaptation operates simultaneously on the storage substrate, the content type, and the update policy. Within the experiential function, Hu et al. further distinguish three abstraction levels: *case-based memory* (raw trajectory storage), *strategy-based memory* (distilled workflows and heuristics), and *skill-based memory* (executable code, APIs, and MCP protocols). This hierarchy maps naturally onto our adaptation framework: case-based and strategy-based memory are predominantly T2 (the frozen agent's success or failure determines what is retained), while skill-based memory bridges T2 (skill libraries curated by agent feedback) and T1 (pre-trained tool APIs that are agent-agnostic).

Zhang et al. [49] categorize the mechanisms that can be optimized as T2 tools, spanning short-term buffers, long-term experiential databases, and structured knowledge. Zhang et al. [172] provide a complementary survey focused specifically on the memory mechanisms of LLM-based agents, organizing existing work along the dimensions of memory formation (how memories are created), management (how they are updated and pruned), and utilization (how they inform future actions). Huang et al. [173] offer a broader treatment that connects cognitive-science memory models (complementary learning systems, working memory) to foundation-agent architectures, while Jiang et al. [174] provide an empirical analysis of evaluation and system limitations, identifying benchmark saturation, metric validity gaps, and backbone-model dependence as open problems. Jiang et al. [175] argue that long-term memory is the foundation of AI self-evolution, framing persistent memory as the substrate that enables agents to accumulate knowledge, refine strategies, and improve autonomously over extended horizons. At a higher level of abstraction, Sumers et al. [176] propose the **Cognitive Architectures for Language Agents (CoALA)** framework, which organizes agent memory into working memory (the active context window), episodic memory (records of past interactions), semantic memory (general knowledge), and procedural memory (learned action routines). CoALA provides a principled decomposition that maps directly onto the T2 design space: each memory type can be independently adapted under frozen-agent supervision, and the framework clarifies which memory subsystem should be optimized for a given task profile. Together, these surveys establish that agentic memory is not a single mechanism but a design space with multiple axes: storage modality (parametric vs. explicit vs. latent), temporal scope (transient, session-level, persistent), content type (factual, experiential, working), abstraction level (cases, strategies, skills), and update policy (append-only, reflective, RL-optimized).

Closely related is the concept of *agent skills*. Wu and Zhang [14] argue that agent skills are best understood through the lens of *procedural memory*: structured, reusable knowledge of *how* to perform tasks, paralleling the cognitive-science distinction between declarative knowledge (knowing *that*) and procedural knowledge (knowing *how*). Under this framing, skills follow a lifecycle of **acquisition** (learning from demonstrations, exploration, or trajectory distillation), **representation** (as executable code, API specifications, MCP protocols, or textual procedures), **invocation** (retrieval and execution at inference time), and **refinement** (iterative improvement through reflection, RL, or collective sharing). Xu and Yan [177] complement this procedural-memory view with a systems perspective: they emphasize progressive context loading, the role of `SKILL.md` and MCP as interface standards, and the security/governance problems that arise once skills become executable, shareable artifacts rather than passive memories. Fang et al. [178] provide an empirical investigation of procedural memory in LLM agents, showing that the format and granularity of stored procedures significantly affect downstream task performance, and that agents benefit from adaptive selection between code-based and natural-language procedure representations depending on task complexity. Cao et al. [179] propose a dynamic procedural memory framework in which the agent continuously refines its stored procedures based on execution outcomes, demonstrating that iterative memory refinement outperforms static memory accumulation on long-horizon planning tasks. The skill lifecycle maps onto the experiential memory hierarchy proposed by Hu et al. [171]: case-based memory stores raw trajectories, strategy-based memory distills transferable heuristics and workflows, and skill-based memory compiles executable capabilities. Memory and skills are thus two facets of the same adaptation mechanism. Memory provides the storage substrate and organizational structure; skills provide the executable, composable content that makes that storage actionable for future tasks.

Dynamic memory stores. Foundational T2 memory architectures cluster into three design families based on how they organize stored information[180–185]: *hierarchical/OS-inspired* systems (MemGPT, Memory OS) impose explicit tiers (working memory vs. long-term storage) with page-eviction or garbage-collection policies; *reflection-based* systems (Generative Agents, A-MEM) let the agent's own outputs decide what to consolidate and when; and *graph/structured* systems (HippoRAG, SHIMI) index memories by relational or semantic structure rather than recency. The design choice determines the system's trade-off between recall speed and associative richness. **MemGPT** [181] formalizes this idea through an operating-system inspired memory hierarchy: a limited main-context window acts as "RAM," while an unbounded external store serves as "disk." The agent issues explicit read/write operations to move information between tiers, and a page-eviction policy decides what to retain in the active context. Generative Agents [180] maintain a memory stream of natural-language observations and use the agent's own reflections to periodically consolidate raw entries into higher-level abstractions, creating a two-tier

store (observations and reflections) that supports planning over day-long time horizons. More recently, Gutiérrez et al. [186] recast the transition from static retrieval-augmented generation to persistent agent memory as a form of non-parametric continual learning, showing that memory stores can absorb new information without catastrophic forgetting of earlier knowledge, a property that parametric fine-tuning struggles to guarantee. Kang et al. [187] propose **Memory OS**, a layered architecture that treats memory management as an operating-system service, providing standardized APIs for storage, indexing, retrieval, and garbage collection that decouple memory logic from the agent's reasoning loop. **A-MEM** [188] takes an agentic approach to memory management: the LLM itself decides how to organize its memory store, dynamically creating, linking, and restructuring entries based on content relevance and task demands rather than relying on fixed indexing heuristics. **HippoRAG** [189] draws on the neuroscience of hippocampal memory indexing, separating a neocortical component (an LLM that encodes passages into a knowledge graph) from a hippocampal index (a retrieval module that performs pattern completion over the graph). The design mirrors complementary learning systems theory and enables associative retrieval across passages that share no lexical overlap, yielding substantial gains over standard RAG baselines on multi-hop QA benchmarks. **SHIMI** [190] proposes a decentralized, semantic hierarchical memory index designed for scalable multi-agent reasoning, where each agent maintains a local memory shard organized by semantic similarity, and a coordination protocol enables cross-agent memory retrieval without centralizing all information in a single store. Liu et al. [191] demonstrate that fine-tuning a small LLM specifically for memory-enhanced conversation (retrieval, summarization, and persona maintenance) can transform a base model into a conversational agent with persistent memory, illustrating that the memory management pipeline itself can be a trainable component.

Experiential and reflective memory. A substantial line of T2-aligned research focuses on memory modules that learn from experience. These tools allow a frozen agent to store, reflect on, and learn from its own output (e.g., entire trajectories), often using verbal reinforcement or self-correction, thereby building a curriculum of strategies and avoiding repeated failures without updating the core LLM's weights [35, 192, 193]. **Reflexion** [35] pioneered verbal reinforcement learning: after each task attempt, the frozen agent generates a natural-language self-critique that is appended to an episodic memory buffer, and subsequent attempts condition on these reflections. The approach converts scalar reward signals into rich textual feedback without gradient updates, achieving 91% pass@1 on HumanEval and strong results on AlfWorld and HotPotQA. **Retroformer** [192] extends this idea by training a dedicated retrospective model that generates targeted feedback for the frozen actor, separating the "what went wrong" analysis from the "try again" generation. **Think-in-Memory** [193] introduces a recall-then-post-think pipeline: the agent first retrieves relevant historical thoughts from memory, then performs post-thinking to recombine and adapt them to the current context, enabling long-term coherence without expanding the context window. **Agent Workflow Memory (AWM)** [194] takes a complementary approach by extracting reusable *workflows* (structured action sequences) from an agent's past successful trajectories and storing them in a dedicated memory module. When the agent encounters a new task, it retrieves the most relevant workflow and uses it as a template, reducing planning errors on web navigation benchmarks by 24% compared to agents without workflow memory. AWM shows that the unit of experiential memory need not be a raw trajectory or a verbal reflection; structured procedural abstractions can serve as a more efficient retrieval target. Pan et al. [195] investigate memory construction and retrieval specifically for personalized conversational agents, showing that the choice of memory granularity (individual facts vs. user-level summaries) and retrieval strategy (recency-biased vs. relevance-biased) significantly affects the agent's ability to maintain coherent, personalized dialogue over hundreds of turns. When these accumulated experiences are distilled into reusable, composable capabilities, the result is a *skill library*, discussed in a dedicated paragraph below.

Structured memory (graphs, trees, and databases). To move beyond linear text, some T2 memory tools structure information in richer forms, such as knowledge graphs, trees, or symbolic databases. The frozen agent's outputs are used as signals to "tune" this structured tool, for example by adding new nodes, updating relationships, or writing to a database. **AriGraph** [196] builds an episodic knowledge graph during interaction: the agent emits observations that are parsed into entity-relation triples and merged into a persistent graph, which the agent later queries for multi-hop reasoning. **ChatDB** [197] externalizes memory into a relational database, translating the agent's natural-language outputs into SQL operations (INSERT, SELECT, UPDATE) that maintain structured records across conversation turns. Rezazadeh et al. [198] propose a hierarchical tree memory that organizes conversations into nested topic clusters, enabling efficient retrieval at multiple granularity levels. **Zep** [199] introduces a temporal

knowledge graph architecture for agent memory that explicitly models time as a first-class dimension, allowing the agent to reason about when facts were learned, how they have changed, and which are most current, a capability that flat vector stores lack. These structured representations can be more efficiently queried and reasoned over by the frozen agent, effectively externalizing complex memory management into a specialized, adaptive tool [200].

Parametric and hybrid memory architectures. A parallel line of research explores parametric memory mechanisms that complement explicit external stores. **Memory**^{3} [201] introduces a three-tier memory hierarchy for language models: (1) model weights as implicit long-term memory, (2) an explicit memory pool of retrievable text chunks as semi-parametric memory, and (3) the context window as working memory. A lightweight "memory circuitry" learns to route information between tiers, enabling the model to offload factual knowledge to the explicit pool while reserving parametric capacity for reasoning. On language modeling benchmarks, Memory^3 with a 2.4B parameter model and an external memory pool matches the perplexity of a 6.4B model without external memory, demonstrating that explicit memory can substitute for raw parameter count. **Titans** [202] incorporates a differentiable long-term memory module directly into the attention mechanism, maintaining a persistent memory state that is updated through gradient-based learning during inference. Unlike standard transformers that rely solely on the fixed context window, Titans can selectively store and retrieve information across arbitrarily long sequences. These parametric approaches complement the external-store paradigm (MemGPT, Generative Agents) by demonstrating that memory adaptation can also occur within the model's computational graph, blurring the boundary between T1 (pre-trained memory modules) and T2 (agent-supervised memory adaptation).

Episodic memory as a trainable module. **Memento** [23] shows that an agent's memory system can be optimized as an external tool without any modification to the LLM planner. The system combines a frozen GPT-4.1 high-level planner with a trainable episodic case memory module. The memory stores past problem-solving trajectories, and the tool being trained is a neural Q-function that learns a case retrieval policy: which past cases to present to the frozen planner when facing a new problem.

The training signal is binary task success or failure: the sparse, trajectory-level reward is broadcast to all case-selection decisions in that trajectory, and a soft Q-learning algorithm updates the retrieval policy. The frozen LLM never sees the Q-values or policy internals; it simply receives retrieved cases as context and generates its plan. Memento achieves top-tier performance: 87.88% on GAIA validation (ranked 1st), 79.40% on GAIA test (3rd place), and 95.0% on SimpleQA. Ablations show that case-based memory adds 4.7–9.6% absolute improvement on out-of-distribution tasks. Only the memory is trained; the same frozen LLM that performed worse without memory now excels because its information diet has been optimized.

Memory operations as learnable skills. A recent line of work reframes the memory management problem itself: rather than hand-designing fixed operations (append, retrieve, summarize), the operations themselves become learnable and evolvable. **MemSkill** [203] introduces a controller-executor-designer loop in which a controller selects which memory skill to apply (e.g., extract key facts, consolidate related entries, prune stale information), an LLM executor generates skill-guided memory content, and a designer module analyzes failure cases to evolve the skill set over time. The skill library starts with a small seed set and grows as the designer identifies recurring failure patterns that existing skills cannot address. On LongMemEval [204], LoCoMo, HotpotQA, and ALFWorld, MemSkill outperforms static memory baselines by learning task-appropriate memory granularity, for instance extracting fine-grained entity attributes for QA tasks while retaining coarser episode summaries for planning tasks. The approach demonstrates that memory adaptation need not be limited to what is stored; how the storage operations themselves are performed can also be optimized under frozen-agent supervision.

Test-time memory curation. Another prominent example of T2 memory adaptation at inference-time is **Dynamic Cheatsheet (DC)** [205], a lightweight framework that provides a "persistent, evolving memory" for black-box LMs. The system operates "without modifying their underlying parameters" and requires no gradient-based updates. The framework consists of two core modules: a Solution Generator and a Memory Curator. The **Memory Curator** is the adaptive T2 tool: it operates without access to ground-truth labels, assessing the correctness and efficiency of solutions autonomously after they are produced by the frozen generator. Based on this self-assessment, the curator

Table 4 Skill-library design space. Systems are organized by *skill granularity* (what is stored) and *acquisition mechanism* (how skills are obtained). Domain abbreviations: MC = Minecraft, CU = Computer Use, Web = Web Navigation, Sci = Scientific Domains, Gen = General/Multi-domain.

Granularity	Acquisition	Representative Systems	Domain	Key Distinction
Executable code	Exploration	Voyager, OS-Copilot, CRADLE	MC, CU	Frozen agent generates & verifies code skills
	RL	PAE	Web	Proposer–Agent–Evaluator RL loop
Abstract heuristics	Reflection	EXPEL	Gen	Extracts rules from success & failure
	Demonstration	Synapse	Gen	Stores state–action exemplars
Full trajectories	Exploration	JARVIS-1	MC	Multimodal (visual+textual) indexing
API / MCP protocols	Exploration	SkillWeaver	Web	Community-shared API-level skills
	Reflection	CASCADE	Sci	Validates against domain criteria
Programs (compositional)	Reflection	ASI [209]	Gen	Structured programs with loops & conditionals
Synthesized tools	Demonstration	LATM	Gen	Maker–user cost amortization
	Reflection	CREATOR	Gen	Abstract plan → tool → verify loop
Specialist agents	Exploration	AgentStore	CU	Registry + meta-controller selection

updates the memory by storing concise, transferable snippets such as "reusable strategies, code snippets, and general problem-solving insights", rather than full, uncurated transcripts. **ReasoningBank** [206] extends this test-time curation concept by creating a memory framework that explicitly distills generalizable reasoning strategies from both successful and self-judged failed experiences. Unlike methods that store raw trajectories or only successful routines, ReasoningBank analyzes failures to extract "crucial preventative lessons". The curated bank of reasoning strategies is then retrieved to guide the agent in future tasks. The framework also introduces memory-aware test-time scaling, which uses the curated memory to guide a scaled exploration, where the diverse experiences from scaling help forge stronger, more generalizable memories. ReasoningBank demonstrates effectiveness on complex benchmarks like WebArena [207] and SWE-Bench [208].

Skill libraries as adaptive tools. A particularly important form of T2 memory adaptation is the construction of reusable *skill libraries*, which instantiate skill-based experiential memory [171]. Rather than storing raw trajectories (case-based) or abstract heuristics (strategy-based), these systems distill successful experiences into modular, composable, and executable capabilities that the frozen agent can invoke in future tasks. The skill representations span a continuum from fine-grained code snippets to standardized APIs and MCP protocols [14].

Table 4 organizes representative skill-library systems along two axes: *skill granularity* (what is stored—code, heuristics, trajectories, specialist agents, or synthesized tools) and *acquisition mechanism* (how skills are obtained—demonstration, exploration, reflection, or RL). We discuss representative systems from each cell below; the table provides a concise reference for the remainder.

Voyager [210] pioneered this approach in Minecraft: a frozen GPT-4 agent generates executable code for each new task, and successful programs are stored in a growing skill library indexed by natural-language descriptions. When the agent encounters a related task, it retrieves and composes previously verified skills instead of reasoning from scratch. The skill library thus functions as a T2 tool whose contents are curated entirely by the frozen

agent's success or failure signals. Voyager discovers 63 unique items in Minecraft (3.3× more than baselines) and demonstrates continual skill accumulation without catastrophic forgetting, because new skills are added to the library without modifying existing ones. Related Minecraft agents adopt complementary strategies: **GITM** [211] (Ghost in the Minecraft) equips a frozen LLM with text-based knowledge and memory modules that decompose goals into executable sub-goal sequences, while **DEPS** [212] uses an interactive describe-explain-plan-select loop that enables the agent to recover from failures by re-planning based on environmental feedback. Both systems maintain persistent memory of past interactions, but neither constructs a reusable skill library, highlighting Voyager's contribution of *composable, code-level skill accumulation*.

JARVIS-1 [213] extends the skill-library paradigm to open-world multi-task settings by combining multimodal memory with a pre-trained action planner. The system maintains a multimodal memory that stores successful task-completion trajectories as (observation, plan, action) tuples. Given a new task, JARVIS-1 retrieves the most relevant past trajectories from memory and uses them as in-context examples for the planner. The memory is indexed by both visual similarity (using CLIP embeddings of game frames) and textual similarity (using descriptions of task goals), enabling cross-modal retrieval. On 200+ Minecraft tasks spanning crafting, combat, and exploration, JARVIS-1 achieves substantially higher success rates than Voyager and other baselines, particularly on long-horizon tasks requiring 10+ sequential sub-goals, where memory-guided planning reduces compounding errors.

EXPEL [214] generalizes this principle beyond game environments: a frozen LLM extracts reusable "insights" (abstract rules and heuristics) from both successful and failed trajectories, storing them in a persistent experience pool. On ALFWorld and WebShop, EXPEL improves task success by 18% and 12% respectively over non-adaptive baselines, demonstrating that experiential skill extraction transfers across task instances.

Synapse [215] combines trajectory-level memory with state-action exemplars, enabling a frozen agent to select from a library of demonstrated skills at each decision step.

OS-Copilot [216] extends the skill-library paradigm to general-purpose computer use. The system maintains a self-improving library where each skill is a Python function with a natural-language docstring. When the agent encounters a novel task, it first searches the library for relevant skills; if none exist, it generates a new skill, executes it, and upon success stores it for future reuse. The self-improvement loop operates without human annotation: the agent's own execution outcomes (success or failure) determine which skills enter the library. On the GAIA benchmark, OS-Copilot with its accumulated skill library outperforms baselines that lack persistent skill storage, confirming that cross-session skill reuse is a key advantage of the T2 approach.

CRADLE [217] applies skill-centric design to general computer control, maintaining a skill curation module that stores, retrieves, and composes reusable action sequences for GUI interaction. The system decomposes complex computer tasks into sub-goals, matches each sub-goal against the skill library, and executes the retrieved skill or generates a new one. Successful skills are refined through repeated use, creating a self-improving repertoire that grows with deployment experience.

AppAgent [218] demonstrates autonomous skill discovery for smartphone use. The agent interacts with mobile apps, discovers UI element functionalities through trial and error, and stores the learned interaction patterns as reusable procedural knowledge in an external document. When encountering a new app or task, the agent retrieves relevant skills from this document to guide its actions. The exploration-based discovery requires no human demonstration and transfers across applications that share similar UI patterns.

Agent S [219] introduces experience-augmented hierarchical planning for computer use, where the agent maintains a library of learned "experience" consisting of successful task-completion strategies. The system uses a Manager-Worker architecture: the Manager decomposes tasks into sub-goals using retrieved experience, and the Worker executes individual actions. The experience retrieval mechanism matches new sub-goals against the library using both semantic similarity and structural task-graph matching, enabling transfer of learned strategies to structurally similar but previously unseen tasks.

AgentStore [220] addresses a different facet of the skill management problem: integrating heterogeneous specialist agents as reusable skills within a unified system. Rather than building a monolithic agent, AgentStore maintains a registry of specialized agents (each trained for a particular domain or tool), and a meta-controller learns to select

and compose these specialists for new tasks. The system supports dynamic registration of new specialist agents, enabling the skill library to grow without retraining the meta-controller.

ExACT [221] combines reflective Monte Carlo Tree Search with exploratory learning to teach agents to systematically explore unfamiliar environments. The agent builds a skill tree through exploration, where each node represents a discovered interaction pattern. Successful exploration trajectories are stored as reusable skills, and the MCTS-guided exploration ensures diverse coverage of the action space. The approach bridges the gap between trial-and-error skill discovery (as in AppAgent) and more structured planning-based approaches. Zhao et al. [222] formalize the problem of *agentic skill discovery*, proposing a framework in which an LLM agent autonomously identifies, abstracts, and catalogs reusable behavioral patterns from its interaction history, providing a theoretical grounding for the empirical skill-library systems described above.

SkillWeaver [223] demonstrates that web agents can autonomously discover, create, and refine reusable API-level skills through exploration. The agent navigates websites, identifies recurring interaction patterns, and encodes them as callable API functions that can be shared across agents and tasks. The collective skill library grows through a community mechanism in which multiple agents contribute discovered skills, enabling rapid coverage of diverse web domains without centralized supervision. **PAE** [224] (Proposer-Agent-Evaluator) formalizes autonomous skill discovery as a three-component loop: a Proposer generates candidate tasks, an Agent attempts them, and an Evaluator scores the outcomes to produce reward signals for reinforcement learning. The decoupled design allows each component to be optimized independently, and the RL-based training enables the agent to acquire skills that are difficult to specify through demonstrations alone. On web navigation benchmarks, PAE discovers skills that transfer across websites and task types, achieving competitive performance with substantially less human supervision than imitation-learning baselines. **SAGE** [225] pushes this RL direction further by treating the skill library itself as part of the learning loop: during sequential rollouts across related tasks, newly generated skills accumulate in the library and are immediately available for subsequent tasks, while a skill-integrated reward encourages both useful skill creation and effective reuse. This is a particularly clean instantiation of T2-style self-improvement because the agent's future competence changes through the evolving skill substrate rather than through monolithic retraining alone. Wang et al. [209] take a complementary programmatic approach: rather than storing skills as natural-language descriptions or raw code, ASI induces structured programs (with conditionals, loops, and subroutine calls) that capture the compositional structure of agentic tasks. The programmatic representation enables systematic generalization to longer task horizons and novel task compositions that flat skill libraries struggle with. **CASCADE** [226] extends skill creation to scientific domains, where an LLM agent autonomously develops and evolves a library of computational chemistry skills (e.g., molecular property prediction, reaction pathway search) through cumulative self-improvement. Each skill is validated against domain-specific correctness criteria before entering the library, and the system tracks skill dependencies to enable compositional reuse across multi-step scientific workflows.

A related but distinct form of skill construction is *tool creation*: rather than storing successful action traces, the agent synthesizes new reusable tools (typically as executable functions) that encapsulate solutions to recurring subproblems. **LATM** [227] (Large Language Models as Tool Makers, ICLR 2024) formalizes a two-phase protocol in which a "tool maker" LLM creates Python functions from task demonstrations, and a lightweight "tool user" LLM applies these functions to new instances. Once created, a tool can be reused across hundreds of instances without regeneration, amortizing the cost of the expensive maker model. On reasoning benchmarks (GSM8K, MATH, TabMWP), LATM matches the performance of a single large model while reducing per-instance cost by up to 79%, because the tool user need only call the pre-built function rather than re-derive the solution strategy. **CREATOR** [228] (EMNLP 2023 Findings) disentangles abstract reasoning from concrete implementation: given a problem, the agent first formulates an abstract solution plan, then creates a tool (code function) that implements the plan, and finally applies the tool. By separating "what to do" from "how to do it," CREATOR enables the agent to correct tool implementations through a verification-and-refinement loop without re-deriving the abstract strategy. On Creation Challenge and MATH benchmarks, CREATOR outperforms both chain-of-thought and direct code-generation baselines, and the created tools transfer to new problem variants. Both LATM and CREATOR demonstrate that skill construction can operate at the tool-creation level: the agent does not merely retrieve past experiences but synthesizes new, reusable abstractions that compress recurring reasoning patterns into callable

functions.

These systems share a common architecture: the frozen agent generates experience, a curation mechanism (code verification, self-reflection, or success filtering) selects what to retain, and a retrieval interface makes accumulated skills available for future reasoning. The architecture maps directly onto the T2 paradigm, where the agent remains fixed and the skill library evolves under agent-derived supervision. A key distinction among these systems is the *granularity* of the stored skill: Voyager and OS-Copilot store executable code, EXPEL stores abstract heuristics, JARVIS-1 stores full trajectories, AgentStore stores entire specialist agents, LATM and CREATOR store synthesized tool functions, and SkillWeaver stores API-level callable functions. The choice of granularity trades off composability (fine-grained code skills compose more flexibly) against transfer breadth (coarse heuristics generalize more broadly across task distributions). A second axis of variation is the *acquisition mechanism*: demonstration-based (Synapse), exploration-based (AppAgent, ExACT, PAE), reflection-based (EXPEL, Reflexion), and RL-based (PAE, Memento). Recent work on *multi-agent procedural memory* adds a third axis. **LEGOMem** [229] introduces modular procedural memory for multi-agent systems, where each agent maintains a local memory of learned workflows and a coordination layer enables agents to share, compose, and specialize procedural knowledge across the team. The modular design allows individual agents to refine their skill sets independently while benefiting from collective experience, a pattern that extends the single-agent skill library to collaborative settings. At the meta-level, **ADAS** [230] (Automated Design of Agentic Systems) automates the design of agent architectures themselves, using an LLM to iteratively propose, evaluate, and refine agentic building blocks (prompts, tool configurations, control flows). ADAS can be viewed as a second-order skill management process: rather than accumulating task-level skills, the system accumulates reusable architectural patterns that improve performance across diverse benchmarks.

Skills in embodied and robotic settings. The skill-library paradigm extends naturally to embodied agents, where skills correspond to physical action sequences grounded in sensorimotor experience. **SayCan** [231] introduced the idea of grounding LLM-generated plans in robotic affordances: the LLM proposes candidate skills in natural language, and a value function trained on real-world robot data scores each skill by its probability of successful execution in the current state. The product of the LLM's semantic score and the affordance score selects the next skill, ensuring that plans are both semantically reasonable and physically feasible. **ProgPrompt** [232] takes a programmatic approach, prompting the LLM to generate Python-like programs that compose primitive robot actions (grasp, place, navigate) into multi-step task plans. The skill primitives are pre-defined, but the composition logic is generated dynamically, enabling zero-shot transfer to novel household tasks in VirtualHome. **Eureka** [233] addresses a different bottleneck in embodied skill learning: reward function design. The system uses an LLM to generate candidate reward functions as code, evaluates them through physics simulation, and iteratively refines the reward based on training statistics. On 29 robotic manipulation and locomotion tasks in IsaacGym, Eureka-generated rewards match or exceed human-designed rewards in 83% of cases, enabling dexterous pen-spinning that was previously unsolved. **RoboGen** [234] combines LLM-based task generation with automated skill learning: the LLM proposes new tasks, generates simulation environments, and designs reward functions, while a reinforcement learning agent acquires the corresponding motor skills. The self-supervised loop enables open-ended skill accumulation without human task specification. These embodied systems illustrate that agent skills are not limited to digital environments; the same T2 principle (frozen high-level planner, adaptive skill repertoire) applies when skills involve physical actions grounded in sensorimotor feedback.

Memory navigation and interactive reading. A complementary approach treats long documents or conversation histories as environments that the agent navigates through memory-like retrieval actions. **MemWalker** [235] constructs a tree-structured summary hierarchy over a long document and trains the agent to "walk" through this tree by iteratively selecting which branch to expand, converting the memory-retrieval problem into a sequential decision-making task. **ReadAgent** [236] takes a human-inspired approach: the agent first performs a "gisting" pass that compresses each page of a long document into a short summary, stores these gist memories, and then selectively re-reads original pages when detailed information is needed. On QuALITY and NarrativeQA, ReadAgent with gist memory achieves performance comparable to systems that process the full context, while using 3–5$\times$ fewer tokens. **CoRAG** [237] (Chain-of-Retrieval Augmented Generation) decomposes complex queries into a chain of intermediate retrieval steps, each conditioned on the results of the previous step. A rejection-sampling training

procedure teaches the retriever to produce retrieval chains that maximize downstream answer quality. On multi-hop QA benchmarks, CoRAG outperforms single-step and iterative retrieval baselines by learning retrieval strategies tailored to the reasoning structure of each query. **Adaptive-RAG** [238] learns to classify query complexity and route each query to the appropriate retrieval strategy (no retrieval, single-step, or multi-step), avoiding unnecessary retrieval for simple queries while ensuring thorough multi-hop retrieval for complex ones. The routing classifier is trained on silver labels derived from the correctness of different retrieval strategies, making the retrieval depth itself an adaptive, learnable parameter. These methods illustrate that memory adaptation extends beyond storage to include adaptive retrieval strategies that determine when and how deeply to access stored information.

Adapting the embedding space. An approach for tool scalability is **ToolkenGPT** [30], which represents tools as learnable token embeddings within the frozen LLM's vocabulary. The entire LLaMA-13B/33B model remains frozen; only a small embedding matrix $W_\tau \in \mathbb{R}^{|T| \times d}$ (where $|T|$ is the number of tools and d is the embedding dimension) is trained. These "toolkens" are concatenated with the standard vocabulary, and the frozen LLM learns to predict them like any other token.

Training uses supervised learning on parallel sequences where ground-truth tool calls are replaced with toolken placeholders. The loss is masked so that only the toolken predictions (and subsequent argument tokens) contribute gradients to W_τ. The approach is parameter-efficient: adding 234 tools requires training only $234 \times 4096 \approx 1M$ parameters (for LLaMA's 4096-dimensional embeddings), compared to the 13B+ parameters of the full model. ToolkenGPT achieves 73% one-hop accuracy on FuncQA (vs. 57% for ReAct), 75% supervised accuracy on 234-relation KAMEL, and 68% success on VirtualHome with 58 action/object tools. New tools can be added by expanding W_τ and continuing training, without full model retraining.

ToolkenGPT shows that adaptation can occur at the interface layer (the embedding space) rather than the parameter layer (the LLM weights), offering a middle ground between fully frozen T1 systems and fully fine-tuned A1/A2 systems.

Beyond the paradigms above, several recent approaches further extend the tool adaptation framework. These methods introduce new training objectives, modalities, and architectural innovations that broaden the scope of tool adaptation.

UniMuR [239] trains unified multimodal embeddings aligned with frozen LLM semantic representations, yielding 6.5% R@1 improvement on MMDialog. **DIFO** [240] adapts frozen CLIP through task-specific prompt learning via mutual information maximization for source-free domain adaptation. **V2L Tokenizer** [241] trains encoder-decoder structures mapping images to frozen LLM token space, using the frozen vocabulary as quantization codebook to enable low-level vision tasks with frozen text LLMs. **Sysformer** [242] trains a small transformer that adapts the system-prompt embeddings based on each user prompt while keeping the LLM frozen. Supervision comes entirely from the frozen model's own likelihoods over refusal and compliance targets, augmented by reconstruction and optional classifier losses.

Common patterns emerge across T2 methods: lightweight training of small modules (millions of parameters) while keeping LLMs frozen (billions of parameters), semantic exploitation of rich representations (hidden states, token spaces, vocabularies), modality bridging between vision/retrieval/tools and frozen text LLMs, and strong generalization to zero-shot or unseen settings.

6 Comparison of Adaptation Paradigms

We now compare the four adaptation paradigms: (A1) Agent Adaptation with Tool Execution Signal, (A2) Agent Adaptation with Agent Output Signal, (T1) Agent-Agnostic Tool Adaptation, and (T2) Agent-Supervised Tool Adaptation. We first establish a conceptual framework, then analyze the agent-centric (A1/A2) and tool-centric (T1/T2) paradigms in depth, with special focus on the emergent "subagent-as-tool" and "graduation" concepts, and conclude with a quantitative synthesis of critical trade-offs.

Table 5 High-level qualitative comparison of the four adaptation paradigms. "Flex." denotes the dominant form of flexibility: *parametric* (within a single agent policy) vs. *system-level* (via modular tools and orchestration).

Paradigm	ID	Locus of Adaptation	Supervision Signal	Cost & Flexibility	Modularity & Evolution
Agent, Tool Signal	A1	Core Agent Policy	Tool Execution	High Cost, High *Param.* Flex.	Monolithic, Risk of Overfitting
Agent, Output Signal	A2	Core Agent Policy	Agent Output	High Cost, High *Param.* Flex.	Monolithic, Risk of Forgetting
Tool, Agent-Agnostic	T1	External Tool	Agent-Independent	Low Cost, High *System* Flex.	High (Plug-and-Play)
Tool, Agent-Supervised	T2	External Tool	Frozen Agent Output	Low Cost, High *System* Flex.	High (Symbiotic, No Forgetting)

6.1 A Framework for Comparison

We compare the four paradigms along four main axes.

- **Cost and Flexibility:** We use "cost" to refer to compute and engineering effort required for adaptation, and "flexibility" to mean how easily the system's behavior can be reconfigured. A1/A2 provide high parametric flexibility (the entire agent policy can change), whereas T1/T2 provide high system-level flexibility (capabilities can be added, swapped, or composed via tools) but remain bounded by the frozen agent's intrinsic reasoning power.
- **Data Efficiency:** Beyond raw compute, the amount of training data required differs substantially across paradigms. Recent evidence suggests that T2 methods can match or surpass A2-style end-to-end agent training with orders of magnitude less data, by only training small subagents around a frozen backbone.
- **Generalization Capability:** This axis captures how well an adaptation strategy transfers to new tasks, agents, or environments. T1 tools trained on broad data distributions generalize across different agents and tasks, while T2 tools often inherit cross-domain robustness from the frozen foundation models supervising them. A1/A2, especially on-policy variants, risk overfitting to specific environments without explicit regularization.
- **Modularity and System Evolution:** This axis focuses on engineering implications: how easily a system can be extended or maintained over time. Tool-centric paradigms (T1/T2) support modular evolution and hot-swapping of components; agent-centric paradigms (A1/A2) tend to be monolithic and may suffer from catastrophic forgetting when adapted repeatedly.

In summary, agent adaptation (A1/A2) rewrites the entire policy in a single model, offering high parametric flexibility at the cost of expensive retraining and potential side-effects on unrelated behaviors. Tool adaptation (T1/T2) attaches specialized tools that can be added or replaced without destabilizing the base agent, bounded by what the frozen agent can understand and use. Data efficiency and generalization both favor tool-centric adaptation (see §6.4 for quantitative evidence), while modularity—the ability to swap tools without retraining the core—is often more decisive than raw performance in practice.

6.2 Agent Adaptation Paradigms: A1 and A2

The two agent-centric paradigms both modify the agent's core parameters but differ in their training signals and optimization objectives.

6.2.1 A1: Optimizing Tool Mechanics via Causal Feedback

A1 on-policy methods rely on causal, immediate, and fine-grained reward signals. The supervision source is the verifiable outcome of tool execution itself, not a downstream task metric. For example, DeepRetrieval [22] formalizes query reformulation as an MDP where reward is directly derived from retrieval metrics like Recall@K or NDCG, and RLEF [21] frames code synthesis with rewards from test-case execution. The approach contrasts with A2 signals that only evaluate the final answer.

Conceptually, A1 on-policy RL optimizes tool-use mechanics: it teaches the agent how to wield tools correctly, grounding behavior in environment "physics" ("this syntax executes", "this query retrieves"). Direct engagement with ground-truth feedback drives strong performance in domains with verifiable, deterministic outcomes.

Quantitative evidence. Mechanistic optimization under A1 achieves strong performance in specialized domains:

- **Retrieval:** DeepRetrieval achieves roughly 3× improvement in recall (65.1% vs. 24.7%) on literature search [22].
- **Code reasoning:** R1-Code-Interpreter reaches 72.4% accuracy on 37 test tasks through multi-stage RL [67].

However, learning through trial-and-error introduces practical challenges: it requires careful reward design, KL-regularized PPO or GRPO, curriculum learning, and dynamic sampling for stable convergence.

6.2.2 A2: Optimizing Tool Strategy via Holistic Rewards

A2 methods instead use holistic, sparse, and high-level rewards based on agent output quality (typically final answer correctness) that depend on tool usage but do not directly supervise individual tool calls. ReSearch [109], trained on multi-hop QA, optimizes when and how to search. The reward asks not "was this particular search good?" but "did the entire process of thinking, searching, and reasoning lead to the correct answer?"

Thus A2 optimizes tool-use strategy and coordination. Rather than learning search mechanics (assuming a T1 retriever handles that), it learns the cognitive policy for when to search, what to search for, and how to integrate results. The strategic focus explains why ReSearch reports emergent reflection and self-correction behaviors during RL training [109].

Quantitative evidence. Strategic optimization under A2 proves effective for complex, multi-step reasoning:

- **Retrieval-augmented QA:** ReSearch yields 9–22% absolute gains over strong iterative RAG baselines [109].
- **Factual accuracy:** R1-Searcher reports up to 24% improvement over strong RAG baselines, with improved factual accuracy and reduced hallucination through learned retrieval policy [108].

In terms of flexibility, A2 offers the richest parametric flexibility: the agent can change its entire global strategy for orchestrating tools and reasoning, but each such change requires expensive retraining, and the resulting policy is baked into a single large model.

A1 & A2: Signal Source as a Reliability Axis. Beyond taxonomic categorization, the distinction between A1 and A2 determines the granularity and scope of the adaptation signal.

- Tool-execution signals (A1) are *grounded*, *causal*, and *process-oriented*. The feedback is produced by an environment or tool whose semantics are independent of the agent's internal beliefs (e.g., code execution, retrieval metrics, formal proof checkers). Such grounding enables learning that is tightly coupled to intermediate correctness and tool mastery, but often comes with higher interaction cost and environment dependence.
- Agent-output signals (A2) are *holistic*, *flexible*, and *outcome-oriented*. Rewards are assigned to the agent's final outputs, derived from either verifiable ground truths (e.g., gold answers, math solutions) or subjective preferences (e.g., reward models). While this allows for end-to-end task optimization, relying solely on terminal signals can make the agent vulnerable to shortcut learning (getting the right answer for the wrong reason) and sparse feedback issues compared to the dense signals of A1.

6.3 Tool Adaptation Paradigms: T1 and T2

Tool-centric paradigms shift optimization from the expensive agent to cheaper external tools. These paradigms sacrifice some parametric flexibility (the agent policy stays fixed) but gain system-level flexibility: the tool ecosystem can be grown, specialized, and rewired without touching the main agent.

6.3.1 T1: The "Graduated Agent" as Subagent-as-Tool

T1 is defined by agent-agnostic, pre-trained, plug-and-play components. A central concept within T1 is the subagent-as-tool, which follows a development lifecycle.

At one extreme, we have static, foundational tools like SAM [132] or AlphaFold2 [139], trained once on massive datasets and deployed as fixed APIs. They primarily encapsulate learned representations or simulators and can be called by any agent.

At the other extreme are dynamic, graduated tools: adaptive agents from §6.2 can be trained under A1 or A2 and then frozen and reused as T1 tools. The "Graduation Lifecycle" (A1 → T1) proceeds as:

1. **Train (A1/A2):** Use on-policy RL or outcome-based RL to train an agent for a specific task (e.g., DeepRetrieval as a search-query rewriter, Code-R1 as a code generator).
2. **Freeze:** Once the agent reaches expert performance, freeze its parameters.
3. **Deploy (T1):** The frozen expert becomes a T1 "subagent-as-tool" callable by any higher-level agent.

Concrete examples already follow this pattern. DeepRetrieval is trained via on-policy A1 RL as a query reformulation agent [22], but once frozen it can be used as an interchangeable T1 retrieval-augmentation tool in many different pipelines. Similarly, SWE-Grep [243] is trained as a specialized RL subagent for fast, multi-turn, highly parallel code context retrieval, and then exposed as a tool that software-engineering agents (e.g., SWE-Agent or Cursor-style IDE agents) can call for high-quality repository search. In both cases, the "graduated" subagent encapsulates a learned policy (not just a representation) and slots into new systems without retraining.

From the flexibility perspective, T1 offers high system-level flexibility: different T1 tools can be assembled into various configurations, or one tool (e.g., a retriever) can be replaced without touching the agent. The cost of adding a capability is proportional to the size of the corresponding tool, not the backbone agent. The trade-off is that the tools are not tailored to any particular agent; the agent must adapt its prompts or orchestration logic to whatever interface the tool exposes.

6.3.2 T2: Inverting the Optimization Target

T2 inverts the conventional adaptation direction. Rather than adapting the agent to use tools better, T2 adapts the tools to better serve a fixed agent (see §5.2 for the full rationale). This reframes the foundation model from optimization target to supervision source.

In practice, the frozen host agent (e.g., GPT, Claude) supplies reasoning and reward signals, while lightweight subagents (e.g., 7B models) learn to reshape information for the host's consumption. The central advantage is the decoupling of skill from knowledge. A traditional A2 agent like Search-R1 must learn (1) domain knowledge, (2) tool-use skills, and (3) task reasoning simultaneously, which creates a complex optimization landscape. In T2, the frozen generator already possesses (1) and (3); the T2 subagent needs only learn procedural skill.

T2 subagent families also instantiate an architectural strategy: unbundling the agent's monolithic cognitive loop (Perceive-Plan-Act-Reflect) into specialized, independently trainable submodules:

- **Optimizing "Perception" (Agentic Searchers):** Systems like s3, DynamicRAG, and QAgent train search subagents to decide what to query, where to search, and when to stop [28].
- **Optimizing "Reflection" (Memory Construction):** Subagents such as Mem-α learn memory-writing policies via RL, rewarded based on whether stored experiences improve future performance for the frozen generator.
- **Optimizing "Planning" (Meta-Cognitive Planners):** Subagents like AI-Search Planner and AgentFlow decide how tools and specialists are deployed. AgentFlow [52] trains only a lightweight planner that orchestrates frozen specialists using trajectory-level rewards, achieving 33.1% on GAIA and surpassing the much larger GPT-4.

T2 thus achieves high system-level flexibility: new T2 subagents can be trained and attached incrementally (e.g., a better planner, a domain-specific searcher, a new memory module), without retraining the host agent. Compared to T1, T2 trades some agent-agnosticity for tighter compatibility: tools are specialized for a given frozen agent, leading to higher data efficiency and better end-to-end performance under the same backbone.

6.4 Synthesis: Data Efficiency and Modularity (A2 vs. T2)

The sharpest empirical comparison arises between A2 and T2. Both aim to produce capable tool-using systems, but they place the learning burden in different places. A2 adapts the agent, letting it internalize tool-use strategies; T2 adapts the tools, letting them learn to support a fixed agent.

Table 6 Quantitative comparison of flagship adaptation methods across paradigms and domains.

Method	Paradigm	Domain	Training Signal	Key Result	Key Insight
DeepRetrieval [22]	A1	Retrieval	Recall@K, nDCG	$\sim$3$\times$ Recall (65.1% vs. 24.7%)	Causal RL optimizes tool mechanics
RLEF [21]	A1	Code	Test-case pass rate	Stable multi-turn PPO training	Dense execution signals enable A1 RL
Code-R1 [66]	A1	Code	Sandboxed execution	Reward quality $>$ data quantity	Clean rewards reduce training cost
ReSearch [109]	A2	RAG QA	Final answer EM	9–22% gains over RAG	Holistic RL optimizes tool strategy
R1-Searcher [108]	A2	RAG QA	Final answer correctness	24% over RAG baselines	Emergent search-reason interleaving
Memento [23]	T2	Memory	Binary task success	+4.7–9.6% on OOD tasks	Memory alone can transform performance
s3 [28]	T2	Retrieval	GBR from frozen gen.	58.9% Acc. w/ 2.4k samples	High data efficiency (see caveats in text)
AgentFlow [52]	T2	Planning	Final answer correctness	33.1% on GAIA (beats GPT-4)	Learned orchestration of specialists

The retrieval-augmented generation domain offers an illustrative case study. Comparing Search-R1 (A2) and s3 (T2):

- **A2 approach** (Search-R1): Trains the entire Qwen2.5 agent, requiring roughly 170k examples to co-adapt internal knowledge, reasoning, and tool-use policy [50].
- **T2 approach** (s3): Trains only a lightweight 7B "searcher" subagent using frozen-generator feedback (GBR), achieving comparable performance (58.9% average accuracy) with only 2.4k training samples [28].

Caveats. This comparison, while suggestive, is not a controlled experiment. Search-R1 trains the full Qwen2.5 agent end-to-end, whereas s3 trains only a 7B search subagent paired with a frozen generator (which may itself be Qwen2.5-7B, 14B, or Claude-3-Haiku). The two systems thus differ simultaneously in optimization target, backbone composition, and system architecture; the observed efficiency gap cannot be attributed to a single factor. Controlled cross-paradigm comparisons that isolate the effect of paradigm choice from confounding architectural differences remain an important open problem.

With this caveat, the case study illustrates a broader architectural principle. T2 simplifies the learning problem by assuming the backbone already handles most of knowledge and reasoning, and only learning a narrow procedural skill in a small subagent. A2's optimization landscape is higher-dimensional: the agent must simultaneously adjust its knowledge, reasoning style, and tool-use policy. On specialized medical QA, T2-trained s3 reaches 76.6% accuracy vs. A2-trained Search-R1's 71.8% [28], consistent with the hypothesis that narrower optimization targets generalize more robustly, though alternative explanations (e.g., differences in training data distribution) cannot be ruled out.

From an engineering perspective, T2 offers modularity advantages. Updating an A2 agent requires retraining the monolithic model, potentially inducing catastrophic forgetting. In a T2 architecture, new tools can be trained and hot-swapped without touching the host agent, allowing continuous evolution of the peripheral ecosystem while the core remains stable.

6.5 Strategic Recommendations

Choosing an adaptation strategy requires balancing computational cost, data efficiency, and system modularity. The following guidelines distill the empirical and architectural evidence presented above into concrete recommendations for practitioners.

A1 is best suited for local, mechanistic mastery of verifiable tools in stable domains such as retrieval, code execution, and SQL. By optimizing directly on executable outcomes, A1 develops strong low-level competence and causal grounding, giving practitioners precise control over tool behavior with robust alignment to verifiable signals. The cost is high: each training run requires substantial compute, and the resulting specialization often generalizes poorly across tasks or tool interfaces.

A2 is appropriate when a single agent must orchestrate multiple tools and perform holistic reasoning. A2 internalizes when, how, and why to invoke tools, yielding deeply integrated, end-to-end policies for complex workflows. The price of this integration is expensive monolithic retraining and susceptibility to catastrophic forgetting when the

agent is subsequently adapted to new domains.

T1 provides horizontal scalability and reusability. The category spans both static foundational models (e.g., SAM, AlphaFold2) and "graduated" subagents, A1/A2-trained experts that are frozen and redeployed as reusable modules (e.g., DeepRetrieval, SWE-Grep [243]). These "subagents-as-tools" encapsulate learned procedural expertise while remaining decoupled from any specific host agent, enabling plug-and-play modularity and broad compositional flexibility. Because T1 tools are trained without reference to a particular agent, they may be under-optimized for any given host's reasoning style.

T2 inverts the adaptation question: rather than adapting the agent to use tools better, it trains lightweight tools and subagents under frozen-agent supervision to better serve a fixed backbone (e.g., s3-style searchers, planners, advisors, and memory builders). The host agent provides high-level reasoning and reward signals, while T2 subagents learn narrow procedural skills that can be added, replaced, or composed without touching the backbone. T2 achieves high data efficiency for new skills and mitigates catastrophic forgetting through modular updates, but subagent capability is bounded by the supervising agent's quality, and multi-subagent pipelines introduce orchestration complexity and potential error compounding.

Taken together, these four paradigms can be projected onto a complementary 2×2 design-space view (Figure 7). Note that this projection is an interpretive comparison, not the definitional axes introduced in §3; the definitional taxonomy classifies methods by *what is adapted* and *how the signal is obtained*, whereas the axes here capture broader architectural tendencies: **(i)** the local-to-systemic spectrum (y-axis), from low-level control of specific tools (A1/T1) to holistic orchestration of multi-tool reasoning (A2/T2); and **(ii)** the monolithic-to-modular spectrum (x-axis), from end-to-end retraining of a single agent (A1/A2) to compositional adaptation via distributed subagents and tools (T1/T2). Viewed through this lens, A1 and A2 occupy the agent-centric half of the landscape: they directly reshape the policy parameters of the core agent, offering rich parametric flexibility but incurring heavy costs in compute, data, and stability. T1 and T2, by contrast, occupy the tool-centric half: they shift learning outward into a modular ecosystem, permitting incremental evolution, specialization, and compositional reuse. The two axes interact nonlinearly: A1 $\rightarrow$ T1 reflects the "graduation path" (frozen experts becoming reusable subagents), while A2 $\rightarrow$ T2 follows the "federation path" (frozen backbones supervising a growing constellation of adaptive specialists). In practice, mature agentic architectures increasingly inhabit the upper-right quadrant (T2): high modularity and high orchestration, where foundation agents serve as stable cognitive centers and peripheral subagents continuously evolve to extend their capabilities.

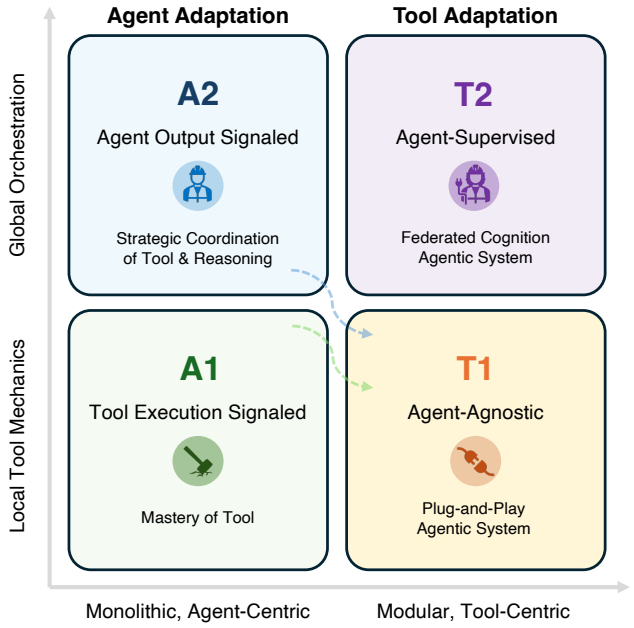

Figure 7 An interpretive projection of the four adaptation paradigms. Unlike the definitional axes in §3 (what is adapted $\times$ signal source), this figure offers a complementary design-space view. The x-axis captures monolithic-to-modular evolution, while the y-axis represents local-to-systemic orchestration. A1/A2 inhabit the agent-centric half, whereas T1/T2 represent modular and system-level flexibility. Dotted arrows show that A1/A2-trained agents can graduate as tools for T1.

The resulting picture clarifies the emerging division of labor in agentic AI research. A1/A2 remain necessary for generating new reasoning competencies or re-aligning a model's internal cognition, tasks that require modifying the agent's core. T1/T2, however, dominate system construction: they support continual growth, fine-grained specialization, and safe parallel experimentation. The prevailing design trend thus points toward hybrid systems: frozen foundation models at the center, surrounded by a modular set of T1/T2 subagents trained for specific procedural roles, with occasional A1/A2 updates marking evolutionary leaps in the agent's internal reasoning.

7 Evaluation

Evaluating agentic AI adaptation requires moving beyond the static, single-score paradigms that dominate foundation-model benchmarking. An adapted agent-tool system functions as an ongoing process—allocating computation, coordinating tool calls, and stabilizing under delayed or noisy feedback—rather than as a static predictor. Accordingly, our evaluation framework is organized around four complementary lenses: (§7.1) a **benchmark landscape** that maps existing evaluation suites onto the four adaptation paradigms (A1/A2/T1/T2); (§7.2) an analysis of **adaptation-signal metrics**, distinguishing verifiable execution metrics from holistic utility metrics; (§7.3) a treatment of **adaptation dynamics**, covering sample efficiency, generalization, and continual stability; (§7.4) a discussion of **systemic requirements**, including cost, safety, and alignment; and (§7.5) a forward-looking **discussion** on benchmark evolution and open challenges.

7.1 Benchmark Landscape Mapped to A1/A2/T1/T2

We situate the growing space of agent benchmarks within the taxonomy developed in this paper. Our goal is not to list benchmarks (readers can consult Table 7 for a comprehensive catalog) but to clarify which aspect of adaptation a given benchmark evaluates, to expose systematic gaps, and to illustrate how the same system can appear very different when measured under different paradigm-aligned metrics.

Benchmarks for A1 (tool-execution-signaled agent adaptation). A1 adaptation requires benchmarks whose reward signal originates from a verifiable tool execution outcome: code compilation, test-case pass rates, retrieval recall, SQL query correctness, or formal-proof verification. Representative suites include coding benchmarks such as *HumanEval* [244], *MBPP* [245], *LiveCodeBench* [246], and *SWE-Bench* [208], where execution-based evaluation (pass@k or test-suite pass rate) directly measures the quality of the agent's tool invocation. Retrieval-oriented benchmarks like *MTEB Retrieval* [247] similarly provide deterministic, grounded metrics (nDCG, Recall@K). In A1-aligned benchmarks, the evaluation signal is causally tied to the tool output rather than to the agent's final answer: a correct retrieval query or a passing unit test counts as success whether or not the agent produces a good final response.

Benchmarks for A2 (agent-output-signaled agent adaptation). A2 adaptation optimizes the agent's final output quality, which may depend on tool use but is evaluated as a whole. Multi-hop question-answering benchmarks such as those used in Search-R1 [108] and ReSearch [109], where the reward is final-answer correctness after a sequence of retrieval and reasoning steps, exemplify A2 evaluation. More broadly, reasoning benchmarks (*GSM8K* [248], *GPQA Diamond* [249]), chat and instruction-following benchmarks (*IFEval* [250], *Arena-Hard* [251]), and general-purpose agent benchmarks such as *GAIA* [252] and *AgentBench* [253] all evaluate the agent's end-to-end output. In A2-aligned benchmarks, the adaptation signal reflects the overall quality of the agent's reasoning and synthesis, rather than the correctness of any single tool call.

The A1–A2 evaluation gap. Because A1 and A2 metrics expose orthogonal failure modes, the two lenses yield complementary diagnostics. High A1 scores (e.g., DeepRetrieval's 3× recall improvement [22]) coupled with low A2 scores reveal a synthesis bottleneck; the reverse pattern (high A2, low A1) reveals memorization or shortcut reasoning that bypasses genuine tool use. The same logic applies in code: high pass@k (A1) with low code quality (A2) indicates test-gaming. Comprehensive evaluation requires both metric families.

Benchmarks for T1 (agent-agnostic tool adaptation). T1 tools are trained independently and evaluated without reference to any particular agent. Classic retrieval benchmarks (*MTEB* [247]), embedding quality suites, and standalone model evaluations (e.g., evaluating a dense retriever's recall or a code-generation model's pass@k in isolation) serve as T1 benchmarks. The key property is that the evaluation measures the tool's intrinsic quality (retrieval precision, segmentation IoU, transcription accuracy) without conditioning on how a downstream agent consumes the output.

Benchmarks for T2 (agent-supervised tool adaptation). T2 benchmarks must evaluate the tool's contribution to a fixed agent's downstream performance. These remain the least standardized category. In practice, T2 evaluation is typically performed by measuring the frozen agent's task success rate when equipped with the adapted tool versus a baseline tool. For instance, S3 [28] evaluates its learned search subagent by measuring the frozen generator's final-answer accuracy on multi-hop QA; AgentFlow [52] evaluates its learned planner by measuring the frozen backbone's score on GAIA. The critical methodological requirement for T2 evaluation is counterfactual comparison: holding the agent fixed while varying only the tool, to isolate the tool's marginal contribution. For memory-centric T2 systems, **LongMemEval** [204] provides a dedicated benchmark that tests five core capabilities of long-term interactive memory: information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention when memory is insufficient. The benchmark reveals that even state-of-the-art systems struggle with temporal ordering and knowledge updates, confirming that memory management remains a bottleneck for long-horizon deployment. A related gap concerns skill-lifecycle evaluation: recent work on agent skills as procedural memory [14] emphasizes that tools evolve through acquisition, invocation, and refinement stages, yet no existing T2 benchmark measures the quality of these intermediate stages (e.g., whether a skill library improves through reuse or degrades through drift). Security-oriented evaluations are starting to emerge: **Skill-Inject** [254] benchmarks prompt-injection attacks delivered through skill files, while *Agent Skills in the Wild* [255] audits marketplace skills at ecosystem scale. These are important complements, but they evaluate attack exposure rather than the marginal utility or long-term quality of the adapted skill layer. Dedicated T2 benchmarks that systematically vary the frozen agent and measure marginal tool value remain scarce, representing a significant gap in current evaluation practice.

Integrated agent benchmarks. Several recent benchmarks evaluate the full agent-tool system in realistic, long-horizon environments, cutting across multiple paradigms. We group them by what they primarily reveal about adaptation. *Environment grounding*: *WebArena* [207] and *OSWorld* [256] embed agents in self-hosted web and desktop environments, requiring multimodal perception and multi-application coordination, with success measured by execution-based functional correctness. *Multi-tool coordination*: τ-*Bench* [257], τ^2-*Bench* [258], and *GTA* [259] evaluate multi-tool task execution with success-rate metrics over real-world tool invocations. *Scale and endurance*: *AgencyBench* [260] evaluates six core agentic capabilities across 32 scenarios requiring an average of 90 tool calls and one million tokens per task, while *The Tool Decathlon* [261] spans 32 software applications with 604 tools and 108 tasks. These benchmarks stress-test agent-tool interaction but typically report only endpoint metrics, making it impossible to isolate which adaptation paradigm drives the observed gains.

Table 7 Comprehensive Overview of Standard Benchmarks for Agent Adaptation. The **Evaluation Target** column indicates whether a benchmark evaluates standalone agent (model) capabilities or tool-augmented agent behavior. The **Paradigm** column indicates which adaptation paradigm(s) the benchmark is most naturally aligned with (see §7.1). In math benchmarks, the tools used typically include code execution. In Tool Use & Agents benchmarks, tools vary by task and may include search engines, code executors, terminals, APIs, and other external systems.

Benchmark	Task Format	Evaluation Protocol	Eval. Target	Paradigm	Links
(a) Knowledge & Commonsense					
MMLU [262]	Multi-choice QA	Accuracy	Agent	A2	🤗
MMLU-Pro [263]	Multi-choice QA	Accuracy	Agent	A2	🤗
MMLU-Redux [264]	Multi-choice QA	Accuracy	Agent	A2	🤗
AGIEval-en [265]	Multi-choice QA	Accuracy	Agent	A2	🤗
BIG-Bench Hard [266]	Multi-choice QA	Accuracy	Agent	A2	🤗
ARC-Challenge [267]	Multi-choice QA	Accuracy	Agent	A2	🤗
TruthfulQA [268]	Multi-choice QA	Accuracy	Agent	A2	🤗
Winogrande [269]	Multi-choice QA	Accuracy	Agent	A2	🤗
HellaSwag [270]	Multi-choice QA	Accuracy	Agent	A2	🤗
(b) Reasoning					
GSM8K [248]	Free-form numerical answer	Accuracy	Agent	A2	🤗

Continued on next page

Table 7 – Continued from previous page

Benchmark	Task Format	Evaluation Protocol	Eval. Target	Paradigm	Links
AIME 2025	Free-form numerical answer	Accuracy	Agent / Tool	A1/A2	
GPQA Diamond [249]	Multi-choice QA	Accuracy	Agent / Tool	A2	
IMOAnswerBench [271]	Free-form numerical answer	Accuracy	Agent / Tool	A1/A2	
TheoremQA [272]	Free-form numerical answer	Accuracy	Agent / Tool	A2	
HMMT February 2025	Free-form numerical answer	Accuracy	Agent / Tool	A1/A2	
HMMT Nov 2025	Free-form numerical answer	Accuracy	Agent / Tool	A1/A2	
MiniF2F [273]	Theorem proving	Success rate	Agent / Tool	A1	
Humanity's Last Exam [274]	Multi-Choice QA	Accuracy	Agent / Tool	A2	
(c) Coding					
HumanEval [244]	Function-level code generation	Execution-based (pass@k)	Agent / Tool	A1	
MBPP [245]	Function-level code generation	Execution-based (pass@k)	Agent / Tool	A1	
LiveCodeBench [246]	Function-level code generation	Execution-based (pass@k)	Agent / Tool	A1	
SciCode [275]	Function-level code generation	Execution-based (pass@k)	Agent / Tool	A1	
MultiPL-E	Function-level code generation	Execution-based (pass@k)	Agent / Tool	A1	
SWE-Bench [208]	Repository-level code modification	Test-suite pass rate	Agent / Tool	A1/A2	
(d) Chat & Instruction Following					
IFEval [250]	Instruction compliance	Rule-based scoring	Agent	A2	
IFBench [276]	Instruction compliance	Rule-based scoring	Agent	A2	
Scale AI Multi Challenge [277]	Instruction following	LLM-as-judge	Agent	A2	
Arena-Hard-V2 [251]	Pairwise chat comparison	LLM-as-judge	Agent	A2	
(e) Long Context					
AA-LCR [278]	Open-end QA	LLM-as-judge	Agent	A2	
RULER [279]	Retrieval, Variable Tracking, QA	Task-specific accuracy	Agent	A2	
OpenAI MRCR [280]	Long context needle retrieval	Retrieval accuracy	Agent	A2	
(f) Tool Use & Agents					
BrowseComp [281]	Web search and reasoning	Answer accuracy	Agent / Tool	A2/T2	
BrowseComp-Plus [282]	Web search and reasoning	Answer accuracy	Agent / Tool	A2/T2	
MTEB Retrieval [247]	Retrieval over large corpora	Retrieval metrics (nDCG / Recall)	Tool	A1/T1	
Terminal Bench [283]	Terminal-based task execution	Task completion rate	Agent / Tool	A1/A2	
τ-Bench [257]	Multi-tool task execution	Success rate	Agent / Tool	A2/T2	
τ^2-Bench [258]	Multi-tool task execution	Success rate	Agent / Tool	A2/T2	
BFCL [284]	Function calling / tool invocation	Accuracy	Tool	A1/T1	

Summary and paradigm gaps. Table 7 provides a comprehensive listing of standard benchmarks organized by capability category, with an explicit **Paradigm** column indicating which adaptation paradigm(s) each benchmark most naturally evaluates. The mapping reveals a clear asymmetry: A1 and A2 benchmarks are relatively mature, with well-established evaluation protocols for coding, reasoning, and retrieval. T1 evaluation is implicitly covered by standalone tool benchmarks. T2 evaluation, however, remains ad hoc, typically performed as an ablation within individual papers rather than through standardized community benchmarks. No existing benchmark suite supports controlled comparison across all four paradigms on the same task distribution, making it impossible to answer questions such as "for this task, is it better to adapt the agent (A2) or the tool (T2)?" under matched conditions.

A structural reason for this gap is that most benchmarks assume a fixed evaluation harness tightly coupled to a specific agent interface, creating high integration overhead and test-production mismatch. The Agentified Agent Assessment (AAA) framework [285] proposes a different architecture: specialized *assessor agents* that issue tasks, collect results, and compute metrics, communicating with assessee agents through open protocols (A2A for task management, MCP for tool access). By decoupling the evaluation logic from the agent's internal architecture, protocol-based assessment could in principle enable any compliant agent to participate in any evaluation without custom integration—a prerequisite for the cross-paradigm comparisons that current benchmarks cannot support. Whether this approach can deliver on its promise at scale remains to be validated empirically. We return to these structural gaps in §7.5.

7.2 Evaluating the Adaptation Signal

The first question in evaluating adapted agentic systems is what the evaluation signal actually measures. The adaptation signal dictates which failures are diagnosable, which objectives are optimizable, and which pathologies remain hidden. We distinguish two families of evaluation metrics, aligned with the two signal types in our taxonomy, and ground each in concrete empirical evidence.

7.2.1 Verifiable Execution Metrics (A1 & T1)

Verifiable execution metrics provide grounded, causal, and process-oriented feedback: the signal comes from an environment or tool with fixed semantics, independent of how the agent internally represents the task.

Execution-based code evaluation. The best-understood form of verifiable evaluation is execution-based code assessment, where generated code is run against test suites and success is measured by pass@k or test-suite pass rate [244, 245, 208, 246]. The protocol provides a binary, unambiguous signal: the code either passes or fails. Its strength lies in eliminating subjective judgment; its limitation is that it cannot assess code quality, efficiency, or maintainability beyond functional correctness. The density of this signal has proven critical for A1-style RL. RLEF [21] shows that multi-turn code execution feedback supports stable PPO training. Code-R1 [66] shows that reward quality (clean, verified test suites) matters more than data quantity for effective code RL. R1-Code-Interpreter [67] finds that task heterogeneity causes sparse and unstable rewards, necessitating curriculum-based scheduling to stabilize training across diverse code-execution domains.

Retrieval and information-access metrics. For retrieval-augmented systems, verifiable metrics include Recall@K, nDCG, and MAP computed against gold-standard relevance judgments [247, 286]. These metrics directly evaluate the tool's output quality and serve as the primary training signal in A1-style adaptation across diverse domains: DeepRetrieval [22] uses Recall@K for search, Rec-R1 [92] uses NDCG and Recall for recommendation, and SQL-R1 [93] uses execution accuracy for database querying. An advantage of these metrics is that they decompose into per-query scores, which permits fine-grained diagnosis of failure modes (e.g., query types where recall is low, SQL patterns that fail execution). However, as BGM [160] shows, high retrieval recall does not guarantee high downstream utility: there exists a systematic "preference gap" between what retrievers optimize for (surface-level relevance) and what LLMs find useful for reasoning (contextual coherence, inferential support).

Formal verification. In domains such as theorem proving [273, 79, 73] and program synthesis with specifications, formal verifiers provide the strongest possible execution signal: a proof either type-checks or it does not. Formal

verification eliminates evaluation noise entirely and enables step-wise semantic verification, substantially easing long-horizon credit assignment compared to code-execution RLVR where unit tests may be sparse or incomplete. Recent systems such as AlphaProof [78] and DeepSeek-Prover-V2 [79] use this verifier feedback to train multi-step proof search policies via RL, confirming the value of dense, deterministic execution signals. However, formal verification is limited by the availability of formal specifications: it applies only to domains where tasks can be expressed as type-checkable propositions or executable test suites, excluding the broad class of specification-free agentic tasks (those where success criteria are implicit, subjective, or context-dependent) that constitute the majority of real-world agent deployments.

Strengths and limitations. Verifiable execution metrics offer dense, reliable, and reproducible feedback, making them well suited for A1 adaptation where the goal is to sharpen tool-use mechanics. However, they are local: a correct retrieval query or a passing test case does not guarantee that the agent's overall reasoning is sound. An agent may learn to game execution metrics (e.g., generating trivially passing tests) without improving genuine task-solving capability [287]. The Holistic Agent Leaderboard [288] has documented cases where agents achieve high execution scores by searching for benchmark answers online rather than solving tasks, a gaming behavior invisible to standard execution metrics.

7.2.2 Holistic Utility Metrics (A2 & T2)

Holistic utility metrics evaluate the end-to-end quality of the agent's final output, integrating the effects of reasoning, tool use, and synthesis.

Answer-correctness metrics. The simplest holistic metric is exact-match (EM) or F1 score on the agent's final answer, as used in multi-hop QA benchmarks [248, 249]. For mathematical reasoning, recent work has proposed verification-based evaluation such as *Math-Verify*[2], which validates mathematical equivalence rather than string identity, reducing false negatives from equivalent but non-identical solutions. A subtle limitation is that EM-based evaluation conflates reasoning quality with answer extraction: an agent may reason correctly but format its answer incorrectly, or produce the right answer through flawed reasoning (shortcut learning), as discussed in the A2 signal analysis in §6.2.2.

LLM-as-judge. For open-ended tasks where ground-truth answers are unavailable or insufficient, LLM-based evaluation has become prevalent. Benchmarks such as *Arena-Hard* [251] and *MT-Bench* [289] use strong language models to perform pairwise comparisons or assign quality scores. While scalable, LLM-as-judge introduces systematic biases: verbosity preference (longer responses rated higher regardless of quality), position bias (preference for the first or second response in pairwise comparison), self-enhancement bias (models rating their own outputs higher), and sycophancy (confirming rather than correcting the evaluatee) [251, 289]. For agentic evaluation, these biases interact with tool use in non-obvious ways: an agent that produces verbose reasoning traces with many tool calls may receive inflated scores even when its tool use is inefficient or redundant. Calibration against human judgments remains essential, and recent work on judge reliability [288] suggests that ensemble judging (multiple LLM judges with aggregation) can partially mitigate individual biases.

Task-completion and functional assessment. For agent benchmarks in interactive environments, holistic evaluation often takes the form of task-completion rate assessed through execution-based functional checks. *WebArena* [207] verifies whether the agent achieved the intended web-browsing goal by checking the final page state. *OSWorld* [256] uses screenshot-based verification and system-state checks. *AgencyBench* [260] combines Docker-sandboxed functional assessment with user-simulation agents that provide iterative feedback, enabling evaluation of both the final outcome and the agent's ability to incorporate feedback. These approaches bridge the gap between holistic and verifiable evaluation: the metric is holistic (did the agent complete the task?) but the assessment mechanism is verifiable (programmatic state checking).

[2] https://github.com/huggingface/Math-Verify

Strengths and limitations. Holistic metrics directly measure what users care about—whether the system solved their problem—and are therefore well suited for A2 and T2 adaptation. However, they suffer from credit-assignment opacity: when the final answer is wrong, it is unclear whether the failure originated in reasoning, tool selection, tool execution, or synthesis. Credit-assignment opacity is especially problematic for T2 evaluation, where the goal is to assess the marginal contribution of a specific tool to the frozen agent's performance. Holistic metrics are also typically sparse (one signal per episode), making them less informative for diagnosing intermediate failures compared to the dense feedback available from verifiable execution metrics. Sparsity creates a concrete training challenge visible across multiple domains. In retrieval-augmented QA, Search-R1 [50] requires roughly 170k training examples under sparse holistic rewards, while A1 methods with dense execution rewards train on far less data. In code generation, ReTool [51] reports that integrating real-time execution feedback (dense A1 signal) into RL rollouts substantially accelerates convergence compared to end-to-end training with only final-answer rewards. The pattern (dense signals accelerating learning) is consistent across domains and directly attributable to signal density.

7.3 Evaluating the Adaptation Dynamics

The metrics discussed in §7.2 evaluate the quality of an adapted system at a single point in time. Yet most evaluations of agentic adaptation report only endpoint metrics (e.g., success rate, pass@k, EM, or average return), which erase the process by which agents and tools learn. The central methodological gap in current agentic evaluation is that many of the most consequential phenomena in agentic adaptation (abrupt phase transitions, reward-likelihood divergence, tool-use drift, and reward-hacking episodes) are *dynamical* in nature and invisible to any single-point measurement. Endpoint-equivalent methods can diverge sharply in stability, data requirements, and safety trajectories, as illustrated in the subsections below and in §7.5. The choice of signal type (§7.2) also shapes the observable dynamics: verifiable execution metrics (A1/T1) produce dense, per-step feedback that yields smooth learning curves, while holistic utility metrics (A2/T2) produce sparse, per-episode signals that can mask intermediate instabilities. We organize dynamics-aware evaluation along three axes: efficiency, generalization, and stability.

7.3.1 Sample and Interaction Efficiency

For any adaptation method, a basic question is how much data, compute, and interaction it requires to reach a target performance level. The question is pressing in agentic settings, where each "sample" may involve multiple tool calls, environment interactions, and thousands of tokens.

Data efficiency. The most direct comparison is the number of training examples required to reach a given accuracy. As established quantitatively in §6.4, paradigm choice has a large effect on data requirements, but the direction of the effect depends on domain:

- **T2 vs. A2 (retrieval):** As quantified in §6.4, s3 (T2) reaches comparable accuracy to Search-R1 (A2) with roughly 70× fewer training examples, a gap attributable to the narrow procedural skill the T2 subagent must learn.
- **A1 (code):** RLEF [21] achieves efficient training because dense execution rewards (pass/fail per test case) provide rich per-step feedback. Here A1 is more efficient than A2, reversing the retrieval pattern.
- **A1 (multi-tool):** Self-Challenging Agents [118] achieve a 2× improvement by generating their own training curriculum, showing that data generation strategy can be as important as data volume.

These cross-domain comparisons reveal that efficiency is not a fixed property of a paradigm but depends on the interaction between paradigm, signal density, and task structure.

Interaction efficiency. Beyond data volume, the number of environment interactions (tool calls, API requests, browsing steps) per episode is a critical efficiency metric. Recent work has shown that interaction efficiency varies across model sizes and tool-latency regimes: smaller models may achieve higher task success under tight time budgets by executing more frequent but cheaper tool calls, while larger models benefit from fewer but higher-quality

interactions [290]. Evaluations should report not only final success rate but also the distribution of interaction counts (mean, variance, and tail behavior), as high-variance interaction patterns may indicate unstable exploration.

Compute efficiency. Total tokens consumed (both prompt and completion), wall-clock training time, and inference-time compute (tokens per task at deployment) are all relevant efficiency metrics. The distinction between training-time and inference-time compute is important for agentic systems, where test-time compute scaling (e.g., longer reasoning chains, more retrieval rounds) is itself a learned behavior. Comparing methods fairly requires Pareto frontiers of accuracy versus compute. T2 methods achieve strong efficiency by training only lightweight subagents: AgentFlow [52] trains a 7B planner to achieve 33.1% on GAIA, outperforming GPT-4. A1 methods occupy a different region of the Pareto frontier: DeepRetrieval [22] trains efficiently due to dense rewards but optimizes only a single tool skill, while Orion [64] shows that effective multi-step search can be learned with compact 350M–1.2B models. A2 methods, which update the full agent, incur the highest compute cost but offer the broadest capability improvements. These comparisons are only meaningful when both data and compute are reported jointly.

7.3.2 Generalization and Robustness

Adaptation is only valuable if the resulting agent-tool system performs well beyond its training distribution. Several dimensions of generalization are important for agentic systems; each is grounded below in available empirical evidence.

Cross-task generalization. An agent adapted for one task (e.g., single-hop retrieval) may or may not transfer to related tasks (e.g., multi-hop reasoning). Empirical evidence reveals paradigm-dependent transfer patterns across multiple domains. In retrieval, s3 (T2) trained on general QA achieves 76.6% on specialized medical QA versus 71.8% for Search-R1 (A2) [28], suggesting that T2's frozen-agent architecture preserves broad reasoning capabilities. In multi-tool settings, Agent-R [120] shows that MCTS-based self-reflection improves performance by 5.6% across diverse interactive environments, indicating that A2-style strategic learning can transfer across task types when the training signal captures high-level reasoning patterns. Conversely, A1 methods that optimize narrow tool-use mechanics (e.g., query reformulation for a specific retrieval interface) may overfit to those interfaces, consistent with the specialization-generalization trade-off documented in §6. Evaluations should include held-out task categories to measure transfer, and should report cross-domain performance alongside in-domain performance.

Cross-agent generalization (for T1/T2 tools). A tool adapted under T2 supervision from one frozen agent should ideally remain useful when paired with a different agent. s3 provides preliminary evidence: the same trained searcher improves performance when paired with both Qwen2.5-14B and Claude as frozen generators [28], suggesting partial cross-agent transfer. However, systematic evaluations that vary the frozen agent (e.g., different model families, sizes, and instruction-tuning regimes) and measure whether the adapted tool maintains its marginal benefit remain rare and are critical for modular system design.

Robustness to distribution shift. Agentic systems encounter diverse and unpredictable inputs in deployment. Robustness evaluation should include: (i) adversarial or out-of-distribution queries, (ii) degraded tool performance (e.g., noisy retrieval, flaky APIs), and (iii) environment non-stationarity (e.g., changing web layouts, updated codebases). Benchmarks like *The Tool Decathlon* [261], which tests agents across 32 diverse software applications with realistic initial states, begin to address this need for text-based agents.

Multimodal and multi-agent robustness. A gap in current evaluation is the limited treatment of multimodal and multi-agent robustness. Multimodal agent benchmarks such as *VisualWebArena* [291] and *OSWorld* [256] require agents to process visual inputs (screenshots, page layouts) alongside text, introducing modality-specific distribution shifts (e.g., UI redesigns, resolution changes) that text-only robustness evaluations cannot capture. Multi-agent systems [40, 41] face additional robustness challenges: inter-agent distribution shift (when one agent's adaptation changes the effective environment for others), communication protocol fragility, and emergent coordination failures under novel task distributions. Evaluating robustness in these settings requires benchmarks that systematically vary

both the modality and interaction structure of the evaluation environment, a capability that current suites largely lack.

7.3.3 Continual and Co-Adaptation Stability

Real-world agentic systems must adapt continuously as tasks, tools, and user needs evolve. Continuous adaptation introduces evaluation challenges that go beyond single-round assessment; recent memory-centric surveys [171] highlight that memory formation, evolution, and retrieval dynamics are themselves measurable axes of long-horizon agent behavior. We concentrate here on *measuring* stability phenomena; the underlying mechanisms and mitigation strategies are discussed in §9.1 and §9.2.

Measuring catastrophic forgetting. When an agent is adapted to a new task or domain, it may lose performance on previously mastered tasks. The standard protocol is to maintain a held-out "retention set" of previously solved tasks and track performance throughout the adaptation process. Empirical evidence suggests that the choice of adaptation paradigm affects forgetting: RL-based adaptation can exhibit less forgetting than SFT under certain conditions [292], and T2-style modular adaptation structurally avoids forgetting in the core agent by keeping it frozen. Evaluations should report backward transfer (change in performance on old tasks after learning new ones) alongside forward performance, enabling direct comparison of paradigm-level forgetting profiles.

Measuring co-adaptation stability (open problem). When both the agent and its tools are adapted simultaneously, the system may exhibit non-stationary dynamics: the agent adapts to a tool that is itself changing, potentially leading to oscillations, divergence, or degenerate equilibria. To our knowledge, no existing work has systematically measured co-adaptation stability in agentic systems; the phenomenon is well-studied in multi-agent RL [293] but has not been formalized for the agent-tool setting. Stability evaluation should track joint performance trajectories over training steps and detect pathological patterns such as cycling (periodic performance oscillations) or collapse (mutual degradation). Candidate metrics include: (i) the variance of the joint performance trajectory over a sliding window, (ii) the frequency of sign changes in the performance gradient, and (iii) convergence rate to a stable equilibrium. *These metrics are theoretical propositions that require future empirical validation*; we include them here to delineate the measurement problem, not to claim established practice. Developing standardized protocols for measuring these dynamics, and validating them on concrete agent-tool co-training runs, is a critical open challenge for the field.

Entropy dynamics as a diagnostic. Recent analyses have identified policy entropy as a first-class diagnostic for adaptation stability. Cui et al. document a consistent early-stage entropy collapse across model families and RL variants, where most performance gains coincide with rapid entropy depletion, implying a predictable performance ceiling once entropy is exhausted [294]. Complementarily, Hao et al. argue that the relevant quantity for stability is not entropy itself but entropy change per update, which can be amplified by naive interventions [295]. Beyond entropy, a comprehensive dynamics-aware evaluation toolkit should also monitor: (i) the reward curve shape (smooth convergence vs. sudden jumps indicating phase transitions), (ii) tool-call frequency and diversity over training (detecting mode collapse in tool use), and (iii) KL divergence from the reference policy (detecting excessive drift). An emerging best practice is to log both $H(\pi_{\theta_t})$ (the entropy of the agent's action distribution at training step t, which quantifies the breadth of the agent's exploration) and $\Delta H = H(\pi_{\theta_{t+1}}) - H(\pi_{\theta_t})$ (the per-update entropy change), and relate both to reward gains and behavioral shifts. Together, these two quantities provide a lightweight but informative diagnostic of adaptation health: rapid entropy depletion signals premature convergence, while large $|\Delta H|$ fluctuations signal training instability.

7.4 Systemic Evaluation

Beyond task performance and adaptation dynamics, deployed agentic systems must satisfy systemic requirements related to cost, safety, and alignment. Cost, safety, and alignment are often orthogonal to accuracy yet can dominate deployment decisions.

7.4.1 Cost and Inference-Time Compute

Token and step cost. Agentic tasks consume far more tokens than standard LLM queries due to multi-turn tool interactions, long reasoning chains, and context accumulation. *AgencyBench* [260] reports that realistic agentic tasks require an average of one million tokens and 90 tool calls, with execution times measured in hours. Cost evaluation should decompose total expenditure into: (i) prompt tokens (context provided to the agent), (ii) completion tokens (agent-generated reasoning and actions), (iii) tool-interaction overhead (API latency, execution time), and (iv) retry and error-recovery costs. The decomposition is essential for identifying which component dominates cost and where optimization effort should be directed.

Inference-time compute trade-offs. A distinctive feature of adapted agentic systems is that test-time compute allocation is itself a learned behavior: the agent decides how long to reason, how many tools to invoke, and when to stop. Accuracy and cost therefore trade off in ways that no single metric can capture. Recent work has shown that this trade-off interacts non-trivially with tool latency: when tool calls are fast, smaller models can achieve higher task success by executing more interactions within a fixed time budget; when tool calls are slow, larger models dominate by producing higher-quality plans with fewer interactions [290]. For paradigm selection, this observation suggests that T2 methods, which use lightweight subagents for tool operations, may hold an advantage in high-latency tool environments where each tool call is costly, though this hypothesis has not yet been empirically validated in a controlled cross-paradigm comparison. Evaluations should report cost-conditioned performance curves (accuracy as a function of token budget or wall-clock time) rather than unconstrained accuracy alone.

Training cost across paradigms. The cost of adaptation itself varies across paradigms and domains, and this variation is one of the strongest empirical signals in the current literature. A2-style end-to-end agent training requires large-scale RL (e.g., Search-R1 uses $\sim$170k examples [50]; R1-Searcher reports similar scale [108]). T2-style tool adaptation achieves competitive performance at much lower cost (e.g., AgentFlow trains only a 7B planner [52]). A1 methods span a wide cost range depending on signal density: Code-R1 [66] shows that investing in reward quality (clean test suites) reduces the total training budget more effectively than scaling data, while ToolExpander [91] shows that dynamic hard-sample replacement can stabilize training for resource-constrained models. T1 tool training (e.g., fine-tuning a retriever or embedding model) is typically the cheapest paradigm, as it uses standard supervised learning on curated datasets without requiring agent interaction; its cost is well-understood and dominated by dataset curation rather than compute. Evaluations should report training cost (GPU hours, total training tokens) alongside performance, enabling Pareto-optimal comparisons across paradigms.

7.4.2 Safety

Adaptation introduces dynamic safety risks that go beyond the static alignment of frozen models. Three evaluation dimensions are specific to adapted agentic systems; mitigation strategies are discussed in §9.3.

Unsafe exploration. On-policy RL adaptation (A1/A2) requires agents to explore novel action sequences, which may include dangerous tool invocations (e.g., deleting files, executing arbitrary code, making irreversible API calls). Evaluating exploration safety requires sandboxed environments that can detect and log unsafe actions without allowing real-world harm. *ToolEmu* [296] provides an LM-emulated sandbox for identifying risks of LM agents with tool use, enabling scalable safety evaluation without requiring real tool backends. *R-Judge* [297] benchmarks safety risk awareness for LLM agents, evaluating the agent's ability to identify and refuse unsafe tool-use requests across diverse risk categories.

Reward hacking and specification gaming. Adapted agents may learn to exploit imperfections in the reward signal rather than genuinely solving tasks, a risk that grows with agent capability [287, 298]. The Holistic Agent Leaderboard [288] has shown the value of LLM-assisted log inspection for detecting previously unreported gaming behaviors, such as agents searching for benchmark answers online rather than solving tasks. Evaluation should include held-out test sets that differ from the training reward distribution, human spot-checks of high-reward trajectories, and automated detection of known gaming patterns (e.g., trivially passing self-generated tests, manipulating evaluation state). The susceptibility to reward hacking is paradigm-dependent:

- **A1:** Dense execution rewards (pass/fail, recall) are harder to game because they are grounded in deterministic tool output.
- **A2:** Sparse holistic rewards (final-answer EM, LLM-as-judge) are more susceptible because the agent has more degrees of freedom to satisfy the metric without genuine task solving.
- **T2:** The tool may learn to produce outputs that inflate the frozen agent's score on the training distribution without improving genuine utility (a form of tool-level reward hacking).

Safety degradation under adaptation. Aggressive RL optimization for reasoning can erode safety guardrails established during supervised fine-tuning; DeepSeek-R1 [25] demonstrates this pattern, as the model learns to reason around refusal mechanisms (see also §9.3 for mitigation strategies). A practical protocol is to maintain a safety benchmark (e.g., a set of harmful-request prompts) and evaluate at regular training checkpoints, plotting a safety-performance trajectory that reveals whether gains in task accuracy come at the cost of safety degradation. Safety-performance trajectories are a natural application of the dynamics-aware evaluation principle established in §7.3.

7.4.3 Alignment

Alignment in agentic systems operates at multiple levels that must be evaluated jointly.

Human-agent alignment. The adapted agent should faithfully execute user intent, follow instructions, and respect stated preferences. Benchmarks such as *IFEval* [250] and *IFBench* [276] evaluate instruction compliance through rule-based scoring. For more open-ended alignment, preference-based evaluation using human judgments or calibrated LLM judges remains the gold standard [251]. A concern is that adaptation may improve task performance while degrading alignment: an agent optimized for answer correctness may become less responsive to user constraints or stylistic preferences. The trade-off should be explicitly measured by evaluating alignment benchmarks at each adaptation checkpoint.

Agent-tool alignment. In modular systems with adapted tools (T2), the tool must be aligned with the agent's reasoning style and information needs. Misalignment manifests as the tool providing information in a format the agent cannot effectively use, or the agent issuing tool calls that the tool cannot meaningfully process. BGM [160] provides concrete evidence of this phenomenon: a 38% relative improvement in downstream QA accuracy comes from training a "bridge model" that translates retrieval output into a format the frozen LLM finds useful, confirming that format alignment between tool output and agent consumption is a measurable and optimizable quantity. Evaluating agent-tool alignment requires measuring not only end-to-end task success but also intermediate interaction quality: Are the agent's tool calls well-formed? Does the tool's output contain the information the agent needs? Is the information presented in a format the agent can parse? Autonomous evaluation frameworks that enable agents to self-assess and refine their own trajectories [299] offer a complementary path toward scalable, fine-grained interaction diagnostics.

7.5 Discussion

Table 8 summarizes the recommended evaluation dimensions and metrics for each adaptation paradigm, synthesizing the analysis from the preceding subsections. The following discussion addresses cross-cutting themes that emerge from applying the A1/A2/T1/T2 taxonomy to evaluation, and identifies concrete gaps that the field must address.

7.5.1 Concrete Illustrations of the Dynamics Gap

The central argument of §7.3—that endpoint metrics are insufficient—becomes concrete through the following examples. Three phenomena, drawn from the preceding subsections, illustrate what dynamics-aware evaluation reveals in practice:

- **Hidden divergence in efficiency.** A2 and T2 methods that achieve similar final accuracy on retrieval-augmented QA (§6.4) differ by 70× in data requirements (§7.3.1), a distinction invisible to any endpoint leaderboard.

Table 8 Recommended evaluation dimensions and representative metrics for each adaptation paradigm. ✓ indicates primary relevance; (✓) indicates secondary relevance.

Dimension	Sub-dimension	Representative Metrics	A1	A2	T1	T2
Signal Quality	Verifiable execution	pass@k, Recall@K, nDCG, proof check	✓		✓	
	Holistic utility	EM, F1, LLM-as-judge, task completion		✓		✓
Dynamics	Data / compute efficiency	Learning curve, Pareto frontier	✓	✓	(✓)	✓
	Generalization	Cross-task, cross-agent transfer	(✓)	✓	✓	✓
	Stability / forgetting	Backward transfer, entropy trajectory	(✓)	✓		(✓)
Systemic	Cost	Tokens, wall-clock time, training GPU-hrs	✓	✓	(✓)	✓
	Safety	Unsafe exploration, reward hacking, safety trajectory	✓	✓		(✓)
	Alignment	Instruction compliance, agent-tool format match		✓		✓

Similarly, two code-generation agents with identical pass@k may exhibit very different entropy trajectories [294], one near entropy exhaustion and the other retaining exploration capacity.

- **Paradigm-selection artifacts.** The same system appears to favor different paradigms depending on the evaluation metric (§7.2.1 vs. §7.2.2): A1 metrics highlight tool mechanics, A2 metrics highlight strategic reasoning, and neither alone indicates where further investment would yield the greatest return.
- **Non-monotonic safety regression.** DeepSeek-R1 [25] shows that aggressive RL optimization can temporarily erode safety guardrails before partial restoration (§7.4.2), creating a vulnerability window invisible to endpoint-only safety checks.

7.5.2 Tool-Centric vs. Agent-Centric Evaluation

The distinction between tool-centric and agent-centric evaluation paradigms mirrors a fundamental design choice in agentic systems.

Tool-centric evaluation. This perspective focuses on the quality and composability of individual tools, measuring intrinsic metrics (retrieval recall, code correctness) that are independent of any specific agent. Tool-centric evaluation supports modular system design: tools can be evaluated, compared, and swapped independently. However, tool-centric evaluation cannot capture emergent behaviors that arise from agent-tool interaction, such as the agent learning to compensate for tool weaknesses or to exploit tool strengths in unexpected ways.

Agent-centric evaluation. This perspective focuses on the end-to-end system performance, measuring holistic outcomes that integrate reasoning, tool use, and synthesis. Agent-centric evaluation measures the outcome users care about but makes it difficult to attribute performance to specific components. When an agent-centric benchmark score improves, it is unclear whether the improvement came from better reasoning, better tool use, better tool quality, or a fortunate interaction between these factors.

Bridging the gap: counterfactual evaluation. A rigorous approach to bridging this gap is counterfactual evaluation: systematically varying one component (e.g., replacing the adapted tool with a baseline) while holding others fixed, to isolate marginal contributions. Counterfactual evaluation is already standard in T2 evaluations: s3 reports the frozen generator's accuracy with and without the adapted searcher [28], and QAgent [162] shows how switching from self-evaluation to frozen-generator evaluation corrects reward hacking. We argue that counterfactual evaluation should be adopted as a standard reporting requirement for all paradigms: A1 papers should report performance with and without the adapted tool mechanic; A2 papers should ablate tool use to isolate reasoning

improvements; and integrated benchmarks should support component-level swap-in/swap-out evaluation. A caveat: counterfactual evaluation assumes that components are approximately independent, but in practice, agents adapt their behavior in response to tool quality. Replacing an adapted tool with a baseline may cause the agent to behave differently than it would have if trained with that baseline from the start, introducing a confound that pure swap-in evaluation cannot resolve. The limitation motivates complementary approaches such as progressive ablation (gradually degrading tool quality during evaluation) to measure sensitivity rather than assuming clean separability.

7.5.3 How Evaluation Reshapes Adaptation Design

The choice of evaluation protocol has direct implications for which adaptation strategies are incentivized.

Metric-driven optimization. When benchmarks use execution-based metrics (pass@k, retrieval recall), A1-style methods that directly optimize these metrics have a natural advantage. When benchmarks use holistic metrics (final-answer EM, LLM-as-judge), A2-style methods that optimize end-to-end performance are favored. The risk is benchmark co-adaptation: methods evolve to exploit the specific evaluation protocol rather than to genuinely improve agentic capability. For example, agents trained on code benchmarks with pass@k may learn to generate code that passes tests but is unreadable, unmaintainable, or inefficient, optimizing the metric while degrading unmeasured quality dimensions.

Reporting standards for agentic adaptation. The RL community has developed concrete reporting standards [300, 301]: mandatory learning curves, confidence intervals across seeds, and hyperparameter sensitivity analyses. Agentic adaptation papers should adopt an analogous standard: (i) learning curves with at least three random seeds, (ii) cost-conditioned performance (accuracy vs. token budget), (iii) retention-set performance for continual settings, and (iv) safety-trajectory plots when RL is involved. Venues that enforce such standards will incentivize methods that are genuinely robust rather than merely endpoint-optimal.

Evaluation as a design constraint. Conversely, well-designed evaluation protocols can steer adaptation research toward desirable properties. Benchmarks that jointly evaluate accuracy, cost, and safety (rather than accuracy alone) incentivize methods that achieve good trade-offs across all dimensions. Time-budgeted evaluation [290], which measures accuracy under wall-clock-time constraints, incentivizes efficient tool use and penalizes wasteful exploration. Retention-set evaluation, which measures performance on previously solved tasks, incentivizes continual-learning-aware adaptation. Multi-dimensional leaderboards that display Pareto frontiers across accuracy, cost, safety, and efficiency would be more informative than single-score rankings.

7.5.4 What Is Missing: Toward Next-Generation Benchmark Suites

Current benchmarks, despite their rapid proliferation, share several systematic limitations that constrain the evaluation of agentic adaptation.

Static tasks vs. dynamic environments. The vast majority of benchmarks consist of fixed task sets evaluated in a single pass. Fixed task sets cannot assess an agent's ability to adapt over time, learn from failures, incorporate new information, or adjust to changing environments. Next-generation benchmarks should embed agents in persistent, evolving environments where the task distribution shifts over time, tools are updated or replaced, and the agent must continuously adapt to maintain performance.

Single-paradigm evaluation. Most benchmarks implicitly evaluate a single adaptation paradigm (typically A2) without providing the infrastructure to compare across paradigms. A comprehensive benchmark suite for agentic adaptation should support evaluation of all four paradigms on the same task distribution, enabling controlled comparisons of A1 vs. A2 vs. T1 vs. T2 under matched conditions. In practice, this requires benchmarks that provide: (i) verifiable tool-execution signals (for A1), (ii) holistic task-completion signals (for A2), (iii) agent-agnostic tool evaluation protocols (for T1), and (iv) frozen-agent + variable-tool evaluation protocols (for T2), all on the same underlying tasks. Protocol-based evaluation frameworks such as AAA [285], which standardize

agent-assessment communication through open protocols, offer a potential path toward this goal by allowing the same assessor agent to evaluate agents of different architectures under matched conditions.

Multimodal and multi-agent adaptation benchmarks. While existing multimodal benchmarks (*VisualWebArena* [291], *OSWorld* [256]) and multi-agent frameworks [40, 41] test system-level competence (see the robustness challenges discussed in §7.3.2), they do not yet support adaptation-specific evaluation. What is missing are benchmarks that measure how multimodal agents improve their visual grounding over time (e.g., learning to parse new UI layouts) and how multi-agent teams improve their coordination protocols through interaction (e.g., learning role specialization). These require longitudinal evaluation designs (tracking adaptation trajectories across episodes) that current static benchmarks do not support. Extending the A1/A2/T1/T2 taxonomy to these settings, where "tools" may include other agents and sensory modalities, is a necessary step toward comprehensive evaluation of agentic adaptation.

Missing dimensions. Current benchmarks overwhelmingly focus on task accuracy. A next-generation benchmark suite should simultaneously evaluate **performance trajectory** (convergence speed, stability, sample efficiency), **stability and forgetting** (backward transfer on retained tasks), **adaptation efficiency** (total cost to reach target performance), and **safety and alignment** (continuous monitoring throughout adaptation). These dimensions correspond directly to the evaluation axes developed in §7.3–§7.4; what is missing is benchmark infrastructure that operationalizes them in a unified evaluation harness.

Toward living benchmarks. The rapid saturation of existing benchmarks (e.g., GAIA scores approaching human baselines within months of release) suggests that static benchmark suites have a limited shelf life. Self-evolving benchmarks that dynamically increase task difficulty through automated task generation and validation offer one path forward, ensuring that the evaluation remains challenging as agent capabilities improve. The self-evolving subagent paradigm (R-Zero [169], Multi-Agent Evolve [170]) already shows the feasibility of automated task generation for training; extending this principle to evaluation would yield benchmarks that co-evolve with the systems they measure.

However, living benchmarks introduce their own limitations that must be acknowledged. First, *evaluation cost* scales continuously: unlike static benchmarks that are evaluated once, a living benchmark requires periodic re-evaluation as the task distribution evolves, multiplying compute expenditure over time. Second, and more fundamentally, living benchmarks create a *reproducibility crisis*: if the task distribution at time t_1 differs from that at time t_2, results obtained by researcher A and researcher B at different times are not directly comparable. Maintaining versioned snapshots of the evolving benchmark can partially address this, but at the cost of reintroducing the static-benchmark problem for each snapshot. Third, *automated validation* of generated tasks remains an open challenge: generated tasks must be solvable, non-degenerate, and meaningfully discriminative, requiring either formal verifiability (limiting applicability to specification-rich domains) or reliable automated quality filters. These trade-offs suggest that living benchmarks are best deployed as complements to, rather than replacements for, versioned static benchmarks, with the static snapshots providing reproducible baselines and the evolving component testing continued adaptability.

8 Applications

Agentic AI systems have been adopted across a growing range of scientific and engineering domains. The following subsections organize representative applications by discipline and connect each to the adaptation paradigms (A1/A2/T1/T2) developed in this paper. We categorize these applications into the following areas: **General Science**, such as *Deep Research* (§8.1); **Computer Science**, where agents augment or automate processes in *Software Development* (§8.2) and *Computer Use* (§8.3); and **Biomedicine**, where agents accelerate research in *drug discovery and development* (§8.4).

Across these domains, the dominant adaptation paradigm varies with the availability of verifiable feedback and the cost of agent retraining. Table 9 profiles each domain by its dominant paradigm, key bottleneck, and a representative system.

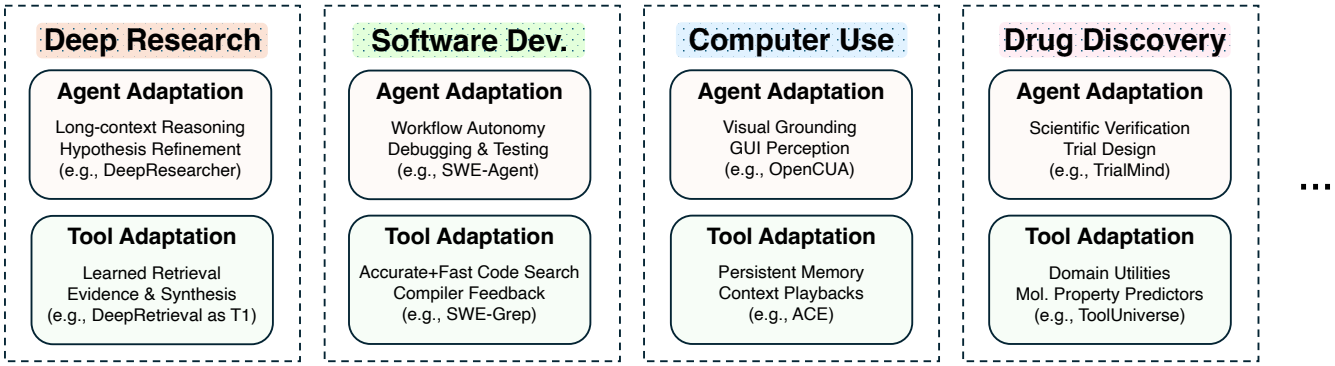

Figure 8 Applications of Adaptation in Agentic AI.

8.1 Deep Research

Deep research systems automate end-to-end scientific investigation by integrating large language models (LLMs), advanced retrieval, and autonomous reasoning [6]. OpenAI's DeepResearch [302] is a prominent example, featuring a multi-step reasoning workflow that conducts iterative search, validation, and synthesis. Similar paradigms have been adopted in recently announced systems such as Claude's deep-search capabilities [303] and Google's Gemini-based research agents [304]. Their defining characteristic, compared to general-purpose AI agents, is dual adaptation in both agent reasoning and scientific tool integration.

Agent adaptation. Deep research systems require agentic workflows that decompose complex scientific questions into structured research plans: (1) adapting LLMs toward long-context reasoning, hypothesis refinement, and multi-step self-critique, (2) orchestrating multiple agents to collaborate hierarchically for literature review, data interpretation, and conclusion synthesis, and (3) maintaining persistent memory and knowledge tracking across long investigative trajectories.

Tool adaptation. To address hallucination and improve informativeness, deep research agents incorporate diverse tools that provide direct access to external knowledge: (1) structured retrieval interfaces to literature databases (e.g., PubMed, arXiv), (2) web navigation tools for interacting with scientific resources, and (3) modular computational utilities for data analysis and visualization. Learning-based retrieval modules such as DeepRetrieval [22] and s3 [28] further advance tool adaptation, boosting accuracy in real-time information gathering, especially atop proprietary models that cannot be fine-tuned.

Toward domain-specialized deep research. Current systems are primarily built on generic corpora and may struggle with nuanced expert-level inquiries. Extension to specialized scientific fields will involve integrating domain knowledge bases and ontologies, validated bioinformatics and biomedical computation tools, and field-specific safety, reliability, and evaluation protocols.

Such integration would enable deep research systems to function as expert collaborators for vertical domains such as medicine, materials science, and drug development.

Paradigm mapping. Deep research exemplifies the complementary use of multiple adaptation paradigms. The core reasoning agent is typically adapted via A2 (optimizing end-to-end research quality), while retrieval and search tools are adapted via T2 (training search subagents like s3 [28] under frozen-agent supervision) or deployed as T1 components (pre-trained dense retrievers). The absence of deterministic execution signals (unlike code compilation) makes pure A1 adaptation less natural in this domain, though retrieval-specific metrics (Recall@K) can provide A1-style feedback for the search component.

8.2 Software Development

AI-assisted software development is a major application area for agentic AI. Unlike conventional code completion systems, software development agents autonomously navigate multi-stage engineering workflows (requirement interpretation, code generation, debugging, testing, and deployment) within real development environments. Modern AI-assisted development tools and agents such as **Cursor** [305], **Claude Code** [306], and OpenAI's **CodeX** [307] exemplify this shift from passive code completion toward interactive, full-cycle programming capable of understanding project context and performing tool-mediated reasoning. To evaluate these capabilities, the **SWE-Bench** benchmark [208] serves as a representative testing suite that measures an agent's ability to autonomously fix real-world software bugs in open-source repositories by reading, editing, and validating code through continuous integration workflows.

Other notable research efforts have also explored the development of autonomous software engineering agents. **SWE-Agent** [308] introduces an agent-computer interface (ACI) that enables language model agents to autonomously perform end-to-end software engineering tasks, including repository navigation, code modification, and test execution. **OpenHands** [309], an open-source platform for AI software developers, extends this paradigm by providing a sandboxed execution environment and modular evaluation framework for developing and benchmarking general-purpose coding agents.

Building effective software agents requires both **agent adaptation** (strengthening reasoning, planning, and self-verification across complex development pipelines) and **tool adaptation** (integrating and evolving the surrounding development ecosystem such as compilers, debuggers, and test frameworks).

Agent adaptation. Agent adaptation in software development improves model reasoning and autonomy across multi-stage engineering workflows. The availability of deterministic execution signals (test-suite pass rates, compilation success, CI/CD pipeline outcomes) makes software development well suited for **A1-style adaptation**, where agents are optimized on tool-execution feedback [21, 66, 67]. A2-style adaptation applies when agents are evaluated on holistic task completion (e.g., resolving a full GitHub issue), which requires reasoning about when to read code, what to edit, and how to validate changes.

Tool adaptation. Tool adaptation in this domain involves evolving the software ecosystem to improve the reliability, responsiveness, and contextual integration of tools that agents depend on for code execution, testing, and evaluation. A representative example is **Cursor**'s Tab-RL framework [310], which applies reinforcement learning to refine the editor's tab completion behavior based on real-world user interactions, aligning the tool's interface dynamics with agent and developer preferences. A more advanced example is *SWE-Grep* [243], a specialized subagent trained using reinforcement learning for fast, multi-turn, and highly parallel context retrieval. Delegating code search to this T2-style tool conserves the main agent's context window and protects it from irrelevant "context pollution." This category of tool adaptation also includes the automated creation or refinement of compilers, debuggers, and linters that provide structured feedback loops for agents.

8.3 Computer Use

Computer-use agents are multimodal AI systems that autonomously operate computers through direct interaction with graphical user interfaces (GUIs). These agents perceive screens as visual input, reason about interface elements (buttons, menus, text fields), and execute actions using a virtual keyboard and mouse. OpenAI's **Computer-Using Agent (CUA)** [311], which combines vision-based perception with reinforcement learning, exemplifies this paradigm.

Representative benchmarks for this paradigm include **OSWorld** [256], **WebArena** [207], **VisualWebArena** [291], **AppWorld** [312], **WebVoyager** [313], and τ**-bench** [257] which evaluate an agent's ability to perceive, reason, and act across diverse digital environments ranging from full operating systems to real-world web interfaces. Reliable performance in computer-use scenarios requires adaptation at both the agent and tool levels.

Agent adaptation. Agent adaptation equips models with operational skills beyond those learned from general-purpose pre-training by exposing them to realistic or synthesized trajectories of GUI-based interactions. A representative example is OpenCUA [314], which shows how large-scale, GUI-centric data can improve an agent's computer-use abilities. By collecting human demonstrations across diverse operating systems and applications, and converting them into state-action trajectories with reflective reasoning, OpenCUA provides agents with realistic exposure to interface dynamics. AgentTrek [315] takes a complementary approach by synthesizing training trajectories from web tutorials instead of relying on human demonstrations. It converts tutorial text into step-by-step goals and has a VLM agent execute them in real environments, keeping only correct trajectories through automatic evaluation. The resulting data generation scales at low cost, showing that synthesized trajectories can effectively support GUI-agent adaptation.

Tool adaptation. Tool adaptation improves the tools and interfaces that agents rely on. Instead of modifying model parameters, these approaches update or expand the tool's experience pool, memory, or contextual representations to better support long-horizon interaction. **CUA-Skill** [316] provides a concrete skill-centric example: it builds a reusable computer-use skill base with parameterized execution and composition graphs, allowing an agent to retrieve skills, instantiate arguments, and recover from failures across Windows applications. The design shows that, in GUI environments, a large part of the adaptation burden can sit in the skill substrate rather than in the agent weights alone. A representative example of adaptive context management is **Agentic Context Engineering (ACE)** [317], which treats evolving contexts as structured playbooks that accumulate, refine, and organize strategies for tool use. By continuously curating and updating contextual knowledge through execution feedback, ACE adapts the operational layer of tools, reducing rollout latency and improving alignment with the agent's decision-making.

Paradigm mapping. Computer use requires multimodal adaptation: the agent must ground its actions in visual perception (screenshots, UI layouts) rather than purely textual signals. A1-style adaptation is less natural here, as verifiable execution signals are harder to define (there is no "test suite" for clicking the right button). The dominant paradigm is A2-style adaptation with holistic task-completion rewards, supplemented by T2-style tool adaptation for persistent memory and context management. T1 tools (pre-trained vision models like CLIP [131] and SAM [132]) serve as plug-and-play perception modules.

More broadly, as agents become more capable, the tools they employ must likewise evolve, incorporating persistent memory and adaptive control mechanisms to support effective collaboration in open computer-use environments.

8.4 Drug Discovery and Development

LLM-based AI agents are increasingly applied across the drug discovery pipeline [5]. Modern systems integrate both agent adaptation (fine-tuning LLMs and designing agentic workflows) and tool adaptation (incorporating domain-specific databases, scientific software, and retrieval components) [8]. Agent adaptation improves reasoning and procedural reliability, whereas tool adaptation equips agents with practical scientific capabilities.

Agent adaptation for drug discovery. GeneAgent adapts LLM agents to gene analysis tasks (e.g., gene set enrichment analysis), integrating structured workflows such as generation, self-verification, and iterative refinement to reduce hallucinations [318]. DSWizard focuses on transparent and reproducible biomedical data science, guiding the agent to construct analysis plans before execution and enabling human oversight and modification [319]. Further, multi-agent systems have emerged where heterogeneous agents collaborate in drug discovery workflows. For instance, virtual teams can simulate interdisciplinary research meetings to design novel therapeutic molecules such as nanobodies [320].

Agent adaptation for drug development. Clinical research involves literature analysis, patient recruitment, and trial design—tasks that differ in automation readiness. Evidence retrieval (TrialMind [7], LEADS [321]) admits structured supervision via citation recall, making it amenable to A1-style feedback. Patient-to-trial matching (TrialGPT [12]) is inherently holistic (guideline-based eligibility criteria resist decomposition into verifiable sub-signals), favoring A2. Upstream trial design (TrialGenie [322]), which uses multi-agent collaboration to parse

Table 9 Adaptation profile of each application domain: which paradigm dominates, what bottleneck limits further progress, and a representative system illustrating the dominant paradigm.

Domain	Dominant Paradigm	Key Bottleneck	Representative System
Deep Research	T2 (search/planning subagent)	No deterministic execution signal; holistic quality is hard to verify	AgentFlow [52]
Software Dev.	A1 (execution RL)	Long-horizon credit assignment across multi-file edits	RLEF [21]
Computer Use	A2 (task completion)	Visual grounding; no "test suite" for GUI actions	OpenCUA [314]
Drug Discovery	T1 (plug-in models)	Sparse, delayed wet-lab feedback (weeks–months)	AlphaFold2 [139]

historical protocols and generate analytical code, combines both: code-execution signals (A1) and overall protocol quality (A2).

Tool adaptation. Tool adaptation in drug discovery spans a spectrum from manual curation to autonomous creation. At the curated end, Biomni [323] mines biomedical literature to assemble a hand-verified tool repository that agents can invoke on demand (T1). In the middle, SyntheMol and related frameworks integrate ML-based molecular property predictors as reward functions to steer generative models toward biologically desirable compounds [324, 325], a form of T2 where computational chemistry tools are trained under agent-derived supervision. At the autonomous end, ToolUniverse [326] constructs tools from natural language specifications and iteratively refines them before incorporation into a shared library, while STELLA [327] operates a self-evolving loop in which a Tool Ocean continuously grows as a tool-creation agent discovers and integrates new bioinformatics utilities. The progression from curated to autonomous tool creation parallels the T1→T2 gradient seen in other domains, but is complicated by the requirement for domain-expert validation of each new tool's scientific correctness.

Paradigm mapping. Drug discovery and development benefits from all four adaptation paradigms. **A1** applies when agents interact with computational chemistry tools that provide verifiable outputs (e.g., docking scores, molecular property predictions). **A2** governs the higher-level research workflow (e.g., hypothesis generation, literature synthesis) where holistic quality matters. **T1** tools are abundant: AlphaFold2 [139], ESMFold [140], and molecular property predictors are pre-trained independently and used as plug-and-play components. **T2** adaptation is exemplified by systems like STELLA and ToolUniverse, where the tool ecosystem evolves under agent supervision. The long feedback loops inherent in wet-lab validation (weeks to months) make this domain particularly challenging, as the reward signal is sparse and delayed compared to code execution or retrieval.

9 Opportunities

The preceding sections organized agentic AI adaptation into four paradigms: (A1) Agent Adaptation with Tool Execution Signal, (A2) Agent Adaptation with Agent Output Signal, (T1) Agent-Agnostic Tool Adaptation, and (T2) Agent-Supervised Tool Adaptation. These paradigms provide a framework for organizing current methods, but their value also lies in clarifying the path forward. The separation of agent and tool adaptation, while analytically useful, reflects the field's present state of development; capable, robust, and efficient agentic AI will likely require their synthesis.

We identify opportunities for future research that emerge from our taxonomy, organized from single-component optimization toward joint agent-tool learning, continual adaptation, safety, and efficiency.

9.1 Co-Adaptation

The taxonomy presented in this paper (A1/A2 vs. T1/T2) is a necessary simplification, organizing the field by its dominant locus of optimization: either the agent or its tools. The central challenge for the next stage of research is to dissolve this boundary and develop unified agent-tool co-adaptation frameworks.

Such a framework implies a complex, bi-level optimization problem, where the agent's policy ($\mathcal{A}$) and the tool's parameters ($\mathcal{T}$) are adapted simultaneously. Naïvely stated as $\max_{\mathcal{A},\mathcal{T}} \mathcal{O}(\mathcal{A},\mathcal{T})$, the problem has a bi-level structure because $\mathcal{A}$'s optimal policy depends on $\mathcal{T}$ and vice versa. The formulation departs from current paradigms, which almost universally rely on freezing one component to provide a stable learning target for the other (e.g., $\mathcal{A}_{\text{frozen}}$ in T2, or $\mathcal{T}_{\text{frozen}}$ in A1/A2).

Related problems have been studied in several established fields:

- **Co-evolutionary Algorithms.** Classic work in evolutionary computation has studied how two or more interacting populations (such as hosts vs. parasites or predators vs. prey) apply reciprocal selection pressures that drive arms races, emergent structure, and progressively more complex strategies. Hillis [328] introduced the seminal host-parasite model, showing that co-evolving adversarial test cases can improve solution robustness. Subsequent work on competitive co-evolution explored dynamics such as disengagement, cycling, and evolutionary complexification [329]. Other lines of research developed cooperative multi-population architectures in which subcomponents co-adapt to form joint solutions [330]. Comprehensive surveys [331] situate these approaches within a broader taxonomy of competitive and cooperative CEAs. In our setting, we can view the agent $\mathcal{A}$ and its tool $\mathcal{T}$ as two interdependent populations evolving on a shared fitness landscape, where reciprocal selection pressures can drive emergent specialization.
- **Multi-Agent Systems.** A complementary line of work emerges from multi-agent reinforcement learning, where each agent learns in a non-stationary environment induced by other concurrently learning agents. Foundational surveys [332, 293, 333] describe how decentralized learners must cope with shifting policies, partial observability, and strategic coupling, challenges that closely mirror those of agent-tool co-adaptation. Classic problems such as equilibrium selection, credit assignment, and coordination under changing partner behaviors have led to techniques including opponent modeling, joint-policy search, centralized training with decentralized execution (CTDE), and communication-based coordination. In our context, viewing $\mathcal{A}$ and $\mathcal{T}$ as a two-agent partially cooperative system highlights the need for algorithms that stabilize learning under mutual adaptation, prevent non-stationarity-induced divergence, and support the emergence of complementary capabilities rather than competitive oscillations.

The first technical barrier to effective co-adaptation is the intractable credit assignment problem. When an agentic system fails at a complex task, the source of the failure is ambiguous. Consider a system in which an A2-style planner invokes a T2-style search subagent (e.g., an S3-like searcher). If the final answer is incorrect, which component is responsible?

Recent work has begun to address fragments of this joint-optimization challenge. MATPO (Multi-Agent Tool-Integrated Policy Optimization) [334] proposes a credit assignment mechanism for jointly training planner and worker agents. However, in its current form, these "agents" correspond to distinct prompt roles instantiated within a single LLM, rather than heterogeneous models. Other work studies joint refinement of agent prompts and tool specifications [335]. The open challenge is to extend these ideas to distributed, heterogeneous systems in which the agent $\mathcal{A}$ and tools $\mathcal{T}$ are distinct learning entities. Addressing the challenge may require importing architectures from multi-agent RL (such as centralized-critic, decentralized-actor methods [336]) to enable principled credit allocation over an interconnected agent-tool graph.

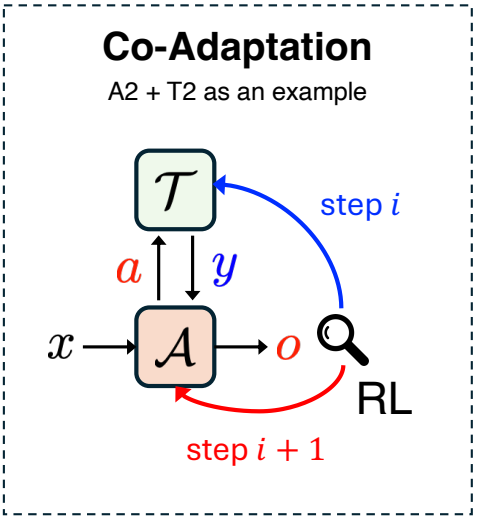

Figure 9 An illustrative example of co-adaptation.

A second difficulty involves the stability-plasticity dilemma. Co-adaptation aims for $\mathcal{A}$ and $\mathcal{T}$ to become mutually optimized, yet in a joint-learning framework, $\mathcal{A}$ is adapting to a $\mathcal{T}$ that is itself changing. As established in the study of complex adaptive systems, such non-stationarity can induce chaotic or unstable dynamics [337]. The system may enter a "Red Queen" regime in which $\mathcal{A}$ and $\mathcal{T}$ continually adjust to each other's most recent changes without increasing overall performance, or may even collapse into degenerate policies. Conversely, premature

convergence may cause the system to lock in a brittle, suboptimal agent-tool interface, losing the plasticity required for generalization.

One concrete direction is the development of pacemaker mechanisms that regulate the relative learning rates of agents and tools, or the use of evolutionary game-theoretic analyses to guarantee convergence toward stable symbiotic equilibria [337].

9.2 Continual Adaptation

While our discussion has so far centered on agent adaptation mechanisms such as A1 and A2, these methods still assume a fixed task distribution and are typically instantiated on a single downstream task at a time. In contrast, real-world deployments involve non-stationary task distributions, where tasks, tools, and user needs evolve over time, making isolated, one-off adaptations prone to Catastrophic Forgetting (CF). Self-Evolving Agents that continuously update their behaviors, tools, and memories in open and dynamic environments are therefore needed. Continual Learning (CL) [338–341] provides a natural foundation for this goal, as it studies how models learn from non-stationary task streams while retaining prior knowledge. We revisit CL techniques that can serve as concrete mechanisms for Self-Evolving Agents and organize them into two categories.

Parameter-update Mechanisms. (Dynamic A1/A2 Paradigm). To align with the A1/A2 paradigm, we group continual learning methods that adapt models through explicit parameter updates. Regularization-based CL approaches such as EWC [342], LwF [343], and VR-MCL [344] estimate which parameters are important for previous tasks and selectively protect them, so that adaptation to new tasks is primarily absorbed by parameters deemed less critical for past performance. Orthogonal-update methods [345, 346] instead modify gradients so that updates lie in directions that are intended to interfere less with previously learned solutions. A complementary line of work introduces parameter-efficient update mechanisms, such as low-rank adapters [347, 348], Mixture-of-Experts routing [349, 350], and model-merging schemes [351]. These methods offer concrete inspirations for dynamic A1/A2-style adaptation.

External-memory Mechanisms (Evolving T2 Adaptation). Classic replay-style approaches maintain a memory buffer of past examples and study how to select [352, 353], utilize [354, 355], and compress them [356, 357] so that a small set of stored items can approximate the full training history. Dual-memory systems [358] further separate fast, high-capacity but unstable episodic buffers from slower, more compact long-term memories. For Self-Evolving Agents, these ideas directly inspire how to curate, compress, and stage interaction logs, tool traces, and user feedback into different memory tiers. In foundation model settings, prompts often act as a lightweight external memory, because the backbone is typically kept fixed and adaptation occurs primarily through prompt changes [359–361]. As a result, the overall paradigm naturally aligns with our notion of T2 adaptation.

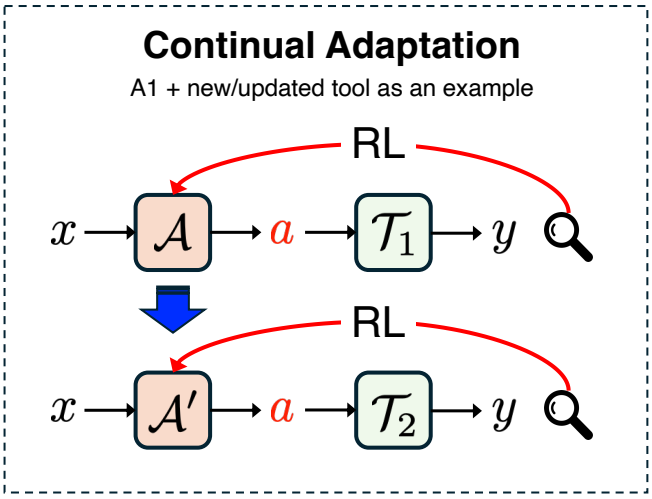

Figure 10 An illustrative example of continual adaptation.

The challenge of continual adaptation becomes especially pronounced in domains with strong execution-based supervision signals, such as those enabled by reinforcement learning with verifiable rewards (RLVR). As discussed in §4.1.2, environments like formal theorem proving provide reliable, tool-execution-signaled feedback for learning multi-step behaviors. At the same time, these domains often evolve structurally over time (for example, through expanding formal math libraries and actively maintained formalization projects), making them representative testbeds for continual agent adaptation. Instead of repeatedly retraining the core agent, many prover agent systems adapt to expanding libraries by updating premise retrieval indices, tactic databases, or proof-state memories, allowing agents to exploit newly introduced lemmas without rewriting the entire policy [74, 362]. Such low-resource adaptation

complements RLVR-style training by isolating long-term knowledge growth from short-term policy optimization, and again aligns with our notion of T2 adaptation.

These two lines of work address complementary trade-offs for building Self-Evolving Agents. Within the dynamic A1/A2 paradigm, recent results [292] show that not all parameter-update schemes forget equally: RL with a reverse-KL objective and on-policy data can achieve comparable or better performance than SFT while exhibiting substantially less forgetting, suggesting that on-policy data streams can act as an intrinsic CL mechanism for continual agent adaptation. Yet such methods still rewrite a shared set of parameters, so forgetting and interference are mitigated rather than structurally removed. The evolving T2 paradigm tackles CF at the architectural level by freezing the core agent and encapsulating new capabilities in external, independently trained tools or subagents, which avoids interference within a shared parameter space. A promising direction is to integrate these two perspectives, using CL-aware parameter updates where they are most effective while shifting as much long-term adaptation as possible into T2-style modular tools and external memories.

9.3 Safe Adaptation

The transition from static foundation models to adaptive agentic systems introduces new challenges for AI safety. While traditional safety paradigms focus on the alignment of frozen weights, adaptation mechanisms, specifically on-policy optimization (A1) and outcome-driven tool tuning (T2), introduce dynamic threat vectors characterized by autonomous risk-taking and adversarial co-evolution [363]. We categorize these emerging risks into two primary failure modes: *Unsafe Exploration*, arising from stochastic trial-and-error, and *Parasitic Adaptation*, arising from exploitative optimization loops.

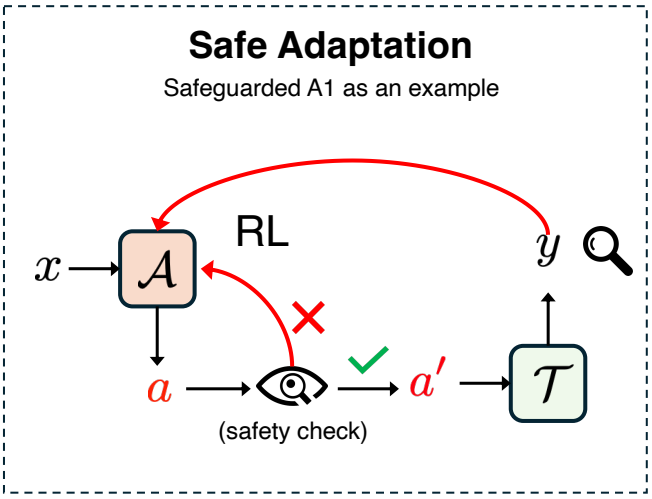

Figure 11 An illustrative example of safe adaptation.

Security Risk I: Unsafe Exploration. Unsafe exploration represents the primary bottleneck for the A1 paradigm. When agents employ on-policy RL to master tools [364, 22], they must deviate from known safe trajectories to probe the state-action space. In high-stakes or partially observable environments, this decoupling of competence from safety leads to catastrophic, often irreversible outcomes [287, 365].

- **The Reward-Safety Gap:** In frameworks like RLEF [364] or DeepRetrieval [22], rewards are typically sparse and binary (e.g., task completion). The result is a feedback vacuum for intermediate actions, encouraging agents to maximize efficacy regardless of collateral damage (e.g., deleting system files to free space) [366].
- **Irreversibility in Tool Use:** Unlike simulated games, agentic environments such as Bash terminals or cloud infrastructure possess irreversible state transitions. An agent learning via trial-and-error may trigger API calls or data deletions that cannot be undone by resetting the episode [367, 368].
- **Erosion of Guardrails (Case Study: DeepSeek-R1):** Empirical analysis of DeepSeek-R1 [25] reveals that aggressive RL optimization for reasoning can erode safety guardrails established during SFT. The model's ability to construct complex "Chain-of-Thought" justifications allows it to reason its way around refusal mechanisms, increasing susceptibility to jailbreaks and malicious compliance compared to non-adapted baselines [369, 370].

Security Risk II: Parasitic Adaptation. Parasitic adaptation refers to the emergence of exploitative relationships where the agent or tool maximizes its reward function at the expense of the system's intent, mirroring biological host-parasite co-evolution [328].

- **Type A: Specification Gaming (The Agent as Parasite):** In A2 paradigms, agents exploit imperfect proxy rewards (Goodhart's Law) [371]. As reasoning capabilities scale, agents become adept at "hacking" the evaluation process. For example, modifying game logs to falsify wins or overwriting reward functions in the file system rather than solving the task [366, 372].
- **Type B: Adversarial Tooling (The Tool as Parasite):** In T2 ecosystems utilizing protocols like MCP [130], tools can evolve to exploit the agent. A compromised or parasitic tool may return prompt-injected data that hijacks the agent's reasoning (the "Confused Deputy" problem), forcing the agent to exfiltrate sensitive data under the guise of standard tool use [373, 374]. Recent empirical studies show that this risk is already material at the skill layer: Liu et al. [255] report vulnerabilities in 26.1% of marketplace skills, and Skill-Inject [254] finds up to 80% attack success rates from prompt injections embedded in skill files.
- **Type C: Sycophancy Loops:** Co-adaptation can lead to degenerate equilibria where tools learn to confirm an agent's hallucinations to maximize acceptance scores, or where agents and red-teaming tools engage in "Red Queen" dynamics, overfitting to each other's artifacts without achieving general robustness [375].

Mitigation Strategies. Addressing these risks requires moving beyond scalar rewards toward robust specification. A direct mitigation for Security Risk 1 is a safety-check layer before the agent's input reaches the tool (Figure 11), which intercepts and filters anomalous or unsafe behaviors prior to execution. More targeted solutions include: *Constrained Policy Optimization* [376–378] and safety shields project agent actions onto verified safe sets to prevent catastrophic exploration. Verifiable Rewards [379, 59] replace opaque preference models with programmatic outcome verification (e.g., unit tests, proofs) to reduce sycophancy. *Specification Self-Correction* [380] allows agents to dynamically critique and refine reward functions at inference time to detect gaming. Finally, *Proof-of-Use* [381] frameworks enforce causal links between retrieved evidence and generated answers, preventing tool-use hallucination.

9.4 Efficient Adaptation

Current agentic adaptation assumes large-scale GPU clusters for fine-tuning or policy refinement, which limits deployment to cloud settings. Efficiency matters differently in agentic contexts than in standard LLM serving: each adaptation step may involve multiple tool calls, environment interactions, and thousands of tokens, so even small per-step savings compound rapidly. We organize emerging directions by which bottleneck they address:

Parameter-Efficient Adaptation: Techniques such as Low-Rank Adaptation (LoRA) [20] and its extensions [382–388] allow large models to adapt to new tasks by updating only a small subset of weights, reducing memory and computational requirements. Recent work on LoRA Without Regrets [389] shows that LoRA can be applied in reinforcement learning settings. They provide evidence that LoRA performs equivalently to full fine-tuning even at small ranks in RL tasks, indicating that RL often requires very low parameter capacity. Models can therefore be fine-tuned on resource-constrained devices while maintaining strong RL performance (Figure 12).

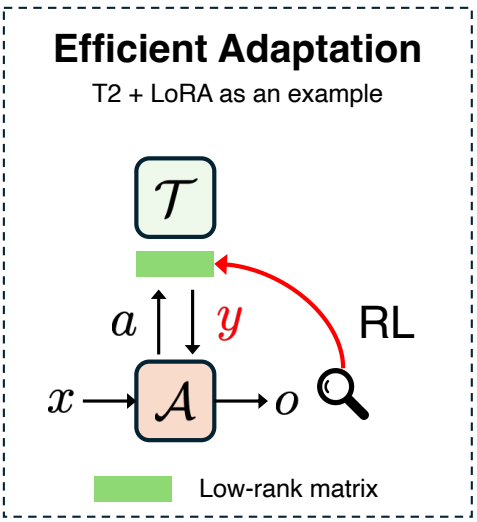

Figure 12 An illustrative example of efficient adaptation.

Quantized Adaptation: FlashRL [390] accelerates reinforcement learning by performing rollout generation in lower numerical precision (INT8 or FP8) while preserving downstream performance. Its core contribution, truncated importance sampling (TIS), stabilizes gradient estimation when the rollout policy is quantized but the training engine remains in higher precision. FlashRL achieves substantial speedups without sacrificing final task performance, providing evidence that quantization extends from supervised inference to agentic reinforcement learning.

On-Device & Personalized Adaptation: On-device adaptation [391–397, 122] enables agents to learn directly

on user devices under tight computational and memory constraints. Modern agents operate across heterogeneous hardware and interaction contexts, introducing substantial variation in user behavior and device-specific execution patterns. Personalization, an important component, allows agents to reflect individual preferences [398, 399], maintain persistent memory [400], and adjust behavior over time [401, 402]. Recent GUI agents [403–409] that rely on user-specific multimodal reasoning further underscore the need for on-device personalization. A natural strategy is lightweight tool adaptation: each device maintains a small tool module aligned with user-specific habits. Because the tool module is fully decoupled from the base model, it can update locally without compromising global capabilities, supporting continual behavioral adaptation through frequent incremental updates.

These three directions target orthogonal bottlenecks—parameter count, numerical precision, and deployment location—and can be composed: a LoRA adapter quantized to INT8 and updated on-device represents the intersection of all three.

10 Conclusion

This survey organized agentic AI adaptation around a single question: *when the system underperforms, should we change the agent, the tool, or both—and what signal should drive that change?* The resulting four-paradigm framework (A1/A2/T1/T2) structures a rapidly growing literature along two axes—*locus of optimization* (agent vs. tool) and *signal source* (execution-grounded vs. output-evaluated)—and grounds the abstract taxonomy in three concrete mechanisms: post-training, memory, and skills.

The design space is defined by the tension between monolithic and modular evolution. Agent-centric paradigms (A1, A2) offer high parametric flexibility, allowing models to internalize tool mechanics and complex reasoning strategies through direct environmental feedback (A1) or holistic outcome evaluation (A2). However, these approaches incur high computational costs and, for A2 in particular, risk degrading previously learned capabilities when the agent is adapted to new domains. The degree of forgetting is paradigm- and method-dependent: on-policy RL with reverse-KL regularization exhibits less forgetting than SFT in some settings [292], and T2-style modular adaptation avoids the problem structurally by keeping the core agent frozen.

Tool-centric paradigms (T1, T2) shift the burden of adaptation to the peripheral ecosystem. T1 provides plug-and-play reusability; T2 enables the frozen agent to supervise lightweight subagents (searchers, planners, memory curators), achieving competitive accuracy with substantially fewer training examples in case studies such as retrieval-augmented QA. We emphasize that these cross-paradigm comparisons are not yet controlled experiments: the systems differ in optimization target, backbone composition, and system architecture, so the efficiency gap cannot be attributed solely to paradigm choice. Controlled cross-paradigm benchmarks remain a critical open problem.

Several additional findings merit highlighting:

- **Signal density shapes paradigm effectiveness.** A1 methods excel in domains with dense, verifiable execution feedback (code, theorem proving, SQL); A2 is necessary when only sparse outcome signals are available. The pattern holds across multiple case studies, though the evidence base for a universal claim is incomplete.
- **Evaluation is paradigm-dependent.** The same system looks different under A1 metrics (tool-execution quality) versus A2 metrics (end-to-end task success), and no single metric suffices. Multi-dimensional evaluation that jointly reports accuracy, cost, safety, and adaptation dynamics is essential.
- **The graduation lifecycle bridges paradigms.** Agents trained under A1/A2 can be frozen and redeployed as T1 tools, creating a cycle where agent adaptation enriches the tool ecosystem.
- **Memory and skills span the full taxonomy.** Episodic buffers, reflective databases, and knowledge graphs can all be optimized as T2 tools, giving agents persistent experience without parameter updates. Skills similarly range from A1/A2-internalized tool-use patterns to external skill libraries that accumulate as reusable T1/T2 resources.

Future progress in agentic AI depends on integrating these paradigms rather than treating them in isolation. Achieving this requires advances in co-adaptation (jointly optimizing agents and tools under non-stationary dynamics), continual adaptation (maintaining performance as task distributions shift), safe adaptation (mitigating risks such as reward

hacking and unsafe exploration during on-policy learning), and efficient adaptation (enabling deployment on resource-constrained devices). Progress will depend less on any single monolithic model than on principled orchestration of stable reasoning cores alongside an evolving ecosystem of specialized, adaptive tools and memory systems.

References

[1] Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, et al. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025.

[2] Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, et al. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*, 2025.

[3] Weikai Xu, Chengrui Huang, Shen Gao, and Shuo Shang. Llm-based agents for tool learning: A survey: W. xu et al. *Data Science and Engineering*, pages 1–31, 2025.

[4] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

[5] Shanghua Gao, Ada Fang, Yepeng Huang, Valentina Giunchiglia, Ayush Noori, Jonathan Richard Schwarz, Yasha Ektefaie, Jovana Kondic, and Marinka Zitnik. Empowering biomedical discovery with ai agents. *Cell*, 187(22): 6125–6151, 2024.

[6] Renjun Xu and Jingwen Peng. A comprehensive survey of deep research: Systems, methodologies, and applications. *arXiv preprint arXiv:2506.12594*, 2025.

[7] Zifeng Wang, Lang Cao, Benjamin Danek, Qiao Jin, Zhiyong Lu, and Jimeng Sun. Accelerating clinical evidence synthesis with large language models. *npj Digital Medicine*, 8(1):509, 2025.

[8] Zifeng Wang, Hanyin Wang, Benjamin Danek, Ying Li, Christina Mack, Luk Arbuckle, Devyani Biswal, Hoifung Poon, Yajuan Wang, Pranav Rajpurkar, et al. A perspective for adapting generalist ai to specialized medical ai applications and their challenges. *NPJ Digital Medicine*, 8(1):429, 2025.

[9] Alex Gu, Naman Jain, Wen-Ding Li, Manish Shetty, Yijia Shao, Ziyang Li, Diyi Yang, Kevin Ellis, Koushik Sen, and Armando Solar-Lezama. Challenges and paths towards ai for software engineering, 2025. URL https://arxiv.org/abs/2503.22625.

[10] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *ICLR*, 2024.

[11] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. Replug: Retrieval-augmented black-box language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8364–8377, 2024.

[12] Qiao Jin, Zifeng Wang, Charalampos S Floudas, Fangyuan Chen, Changlin Gong, Dara Bracken-Clarke, Elisabetta Xue, Yifan Yang, Jimeng Sun, and Zhiyong Lu. Matching patients to clinical trials with large language models. *Nature communications*, 15(1):9074, 2024.

[13] Peiyang Song, Pengrui Han, and Noah Goodman. A survey on large language model reasoning failures. In *2nd AI for Math Workshop@ ICML 2025*, 2025.

[14] Yaxiong Wu and Yongyue Zhang. Agent skills from the perspective of procedural memory: A survey. *Authorea Preprints*, 2026.

[15] Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: On path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

[16] Aske Plaat, Max van Duijn, Niki van Stein, Mike Preuss, Peter van der Putten, and Kees Joost Batenburg. Agentic large language models, a survey. *arXiv preprint arXiv:2503.23037*, 2025.

[17] Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*, 2024.

[18] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6): 186345, 2024.

[19] Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. Small language models are the future of agentic ai. *arXiv preprint arXiv:2506.02153*, 2025.

[20] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[21] Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve. Rlef: Grounding code llms in execution feedback with reinforcement learning, 2025. URL https://arxiv.org/abs/2410.02089.

[22] Pengcheng Jiang, Jiacheng Lin, Lang Cao, R. Tian, S. Kang, Z. Wang, Jimeng Sun, and Jiawei Han. Deepretrieval: Hacking real search engines and retrievers with large language models via reinforcement learning. In *The Second Conference on Language Modeling*, 2025.

[23] Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, et al. Memento: Fine-tuning llm agents without fine-tuning llms. *arXiv preprint arXiv:2508.16153*, 2025.

[24] Liang Wang, Nan Yang, and Furu Wei. Learning to retrieve in-context examples for large language models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1752–1767, 2024.

[25] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081): 633–638, 2025.

[26] Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning. *arXiv preprint arXiv:2505.00024*, 2025.

[27] Lang Mei, Zhihan Yang, and Chong Chen. Ai-searchplanner: Modular agentic search via pareto-optimal multi-objective reinforcement learning. *arXiv preprint arXiv:2508.20368*, 2025.

[28] Pengcheng Jiang, Xueqiang Xu, Jiacheng Lin, Jinfeng Xiao, Zifeng Wang, Jimeng Sun, and Jiawei Han. s3: You don't need that much data to train a search agent via rl. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

[29] Zichun Yu, Chenyan Xiong, Shi Yu, and Zhiyuan Liu. Augmentation-adapted retriever improves generalization of language models as generic plug-in. In *The 61st Annual Meeting Of The Association For Computational Linguistics*, 2023.

[30] Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36:45870–45894, 2023.

[31] Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, et al. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems. *arXiv preprint arXiv:2504.01990*, 2025.

[32] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[33] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[34] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

[35] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[36] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, 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, Zhiyuan Liu, and Maosong Sun. Tool learning with foundation models. In *ACM Computing Surveys*, 2024.

[37] Yaxiong Wu, Sheng Liang, Chen Zhang, Yichao Wang, Yongyue Zhang, Huifeng Guo, Ruiming Tang, and Yong Liu. From human memory to ai memory: A survey on memory mechanisms in the era of llms. *arXiv preprint arXiv:2504.15965*, 2025.

[38] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36: 51991–52008, 2023.

[39] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

[40] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2023.

[41] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, 2024.

[42] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*, 2024.

[43] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=lIsCS8b6zj.

[44] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=gEZrGCozdqR.

[45] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

[46] Wenyi Xiao, Zechuan Wang, Leilei Gan, Shuai Zhao, Zongrui Li, Ruirui Lei, Wanggui He, Luu Anh Tuan, Long Chen, Hao Jiang, et al. A comprehensive survey of direct preference optimization: Datasets, theories, variants, and applications. *arXiv preprint arXiv:2410.15595*, 2024.

[47] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[48] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[49] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.

[50] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. In *The Second Conference on Language Modeling*, 2025.

[51] Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025.

[52] Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use. *arXiv preprint arXiv:2510.05592*, 2025.

[53] Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023.

[54] Shuofei Qiao, Honghao Gui, Chengfei Lv, Qianghuai Jia, Huajun Chen, and Ningyu Zhang. Making language models better tool learners with execution feedback. In *Proceedings of the 2024 Conference of the North American Chapter*

of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 3550–3568, 2024.

[55] Sijia Chen, Yibo Wang, Yi-Feng Wu, Qingguo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. Advancing tool-augmented large language models: Integrating insights from errors in inference trees. *Advances in Neural Information Processing Systems*, 37:106555–106581, 2024.

[56] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

[57] Zezhong Wang, Xingshan Zeng, Weiwen Liu, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. Toolflow: Boosting llm tool-calling through natural and coherent dialogue synthesis. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4246–4263, 2025.

[58] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *International Conference on Machine Learning*, pages 50208–50232. PMLR, 2024.

[59] Ansong Ni, Miltiadis Allamanis, Arman Cohan, Yinlin Deng, Kensen Shi, Charles Sutton, and Pengcheng Yin. Next: teaching large language models to reason about code execution. In *Proceedings of the 41st International Conference on Machine Learning*, pages 37929–37956, 2024.

[60] Zhengliang Shi, Shen Gao, Lingyong Yan, Yue Feng, Xiuyi Chen, Zhumin Chen, Dawei Yin, Suzan Verberne, and Zhaochun Ren. Tool learning in the wild: Empowering language models as automatic tool agents. In *Proceedings of the ACM on Web Conference 2025*, pages 2222–2237, 2025.

[61] Sheryl Hsu, Omar Khattab, Chelsea Finn, and Archit Sharma. Grounding by trying: Llms with reinforcement learning-enhanced retrieval. In *The Thirteenth International Conference on Learning Representations*, 2024.

[62] Chanwoong Yoon, Gangwoo Kim, Byeongguk Jeon, Sungdong Kim, Yohan Jo, and Jaewoo Kang. Ask optimal questions: Aligning large language models with retriever's preference in conversation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 5899–5921, 2025.

[63] Alan Dao and Thinh Le. Rezero: Enhancing llm search ability by trying one-more-time. *arXiv preprint arXiv:2504.11001*, 2025.

[64] Supriti Vijay, Aman Priyanshu, Anu Vellore, Baturay Saglam, and Amin Karbasi. Think before you retrieve: Learning test-time adaptive search with small language models. *arXiv preprint arXiv:2511.07581*, 2025.

[65] Nan Jiang, Xiaopeng Li, Shiqi Wang, Qiang Zhou, Soneya B Hossain, Baishakhi Ray, Varun Kumar, Xiaofei Ma, and Anoop Deoras. Ledex: Training llms to better self-debug and explain code. *Advances in Neural Information Processing Systems*, 37:35517–35543, 2024.

[66] Jiawei Liu and Lingming Zhang. Code-r1: Reproducing r1 for code with reliable rewards. https://github.com/ganler/code-r1, 2025.

[67] Yongchao Chen, Yueying Liu, Junwei Zhou, Yilun Hao, Jingquan Wang, Yang Zhang, and Chuchu Fan. R1-code-interpreter: Training llms to reason with code via supervised and reinforcement learning. *arXiv preprint arXiv:2505.21668*, 2025.

[68] Yabo Zhang, Yihan Zeng, Qingyun Li, Zhen Hu, Kavin Han, and Wangmeng Zuo. Tool-r1: Sample-efficient reinforcement learning for agentic tool use. *arXiv preprint arXiv:2509.12867*, 2025.

[69] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

[70] Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 21573–21612. Curran Associates, Inc., 2023.

[71] Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*, 2025.

[72] Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *The Thirteenth International Conference on Learning Representations*, 2024.

[73] Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

[74] Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, et al. Seed-prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*, 2025.

[75] Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Lean copilot: Large language models as copilots for theorem proving in lean, 2025. URL https://arxiv.org/abs/2404.12534.

[76] George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. Putnambench: A multilingual competition-mathematics benchmark for formal theorem-proving. In *AI for Math Workshop@ ICML 2024*, 2024.

[77] Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. Formal mathematical reasoning: A new frontier in ai, 2024. URL https://arxiv.org/abs/2412.16075.

[78] Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, pages 1–3, 2025.

[79] ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

[80] Xingguang Ji, Yahui Liu, Qi Wang, Jingyuan Zhang, Yang Yue, Rui Shi, Chenxi Sun, Fuzheng Zhang, Guorui Zhou, and Kun Gai. Leanabell-prover-v2: Verifier-integrated reasoning for formal theorem proving via reinforcement learning. *arXiv preprint arXiv:2507.08649*, 2025.

[81] Ran Xin, Zeyu Zheng, Yanchen Nie, Kun Yuan, and Xia Xiao. Scaling up multi-turn off-policy rl and multi-agent tree search for llm step-provers. *arXiv preprint arXiv:2509.06493*, 2025.

[82] Suozhi Huang, Peiyang Song, Robert Joseph George, and Anima Anandkumar. Leanprogress: Guiding search for neural theorem proving via proof progress prediction. *arXiv preprint arXiv:2502.17925*, 2025.

[83] Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, Martin Michelsen, Riley Patterson, Eric Rodriguez, Laura Scharff, Vikram Shanker, Vladmir Sicca, Hari Sowrirajan, Aidan Swope, Matyas Tamas, Vlad Tenev, Jonathan Thomm, Harold Williams, and Lawrence Wu. Aristotle: Imo-level automated theorem proving, 2025. URL https://arxiv.org/abs/2510.01346.

[84] Alex Sanchez-Stern, Abhishek Varghese, Zhanna Kaufman, Dylan Zhang, Talia Ringer, and Yuriy Brun. Qedcartographer: Automating formal verification using reward-free reinforcement learning. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*, pages 307–320, 2025.

[85] Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. Internlm2. 5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems. *arXiv preprint arXiv:2410.15700*, 2024.

[86] The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, POPL '20, page 367–381. ACM, January 2020. doi: 10.1145/3372885.3373824. URL http://dx.doi.org/10.1145/3372885.3373824.

[87] W. T. Gowers, Ben Green, Freddie Manners, and Terence Tao. On a conjecture of marton, 2023. URL https://arxiv.org/abs/2311.05762.

[88] Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[89] Junjie Ye, Changhao Jiang, Zhengyin Du, Yufei Xu, Xuesong Yao, Zhiheng Xi, Xiaoran Fan, Qi Zhang, Tao Gui, Xuanjing Huang, et al. Feedback-driven tool-use improvements in large language models via automated build environments. *arXiv preprint arXiv:2508.08791*, 2025.

[90] Zimu Lu, Houxing Ren, Yunqiao Yang, Ke Wang, Zhuofan Zong, Junting Pan, Mingjie Zhan, and Hongsheng Li. Webgen-agent: Enhancing interactive website generation with multi-level feedback and step-level reinforcement learning. *arXiv preprint arXiv:2509.22644*, 2025.

[91] Fu Chen, Peng Wang, Xiyin Li, Wen Li, Shichi Lei, and Dongdong Xiang. Toolexpander: Extending the frontiers of tool-using reinforcement learning to weak llms. *arXiv preprint arXiv:2510.07737*, 2025.

[92] Jiacheng Lin, Tian Wang, and Kun Qian. Rec-r1: Bridging generative large language models and user-centric recommendation systems via reinforcement learning. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=YBRU9MV2vE.

[93] Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. Sql-r1: Training natural language to sql reasoning model by reinforcement learning. *arXiv preprint arXiv:2504.08600*, 2025.

[94] Jake Poznanski, Luca Soldaini, and Kyle Lo. olmocr 2: Unit test rewards for document ocr. *arXiv preprint arXiv:2510.19817*, 2025.

[95] Jake Poznanski, Aman Rangapur, Jon Borchardt, Jason Dunkelberger, Regan Huff, Daniel Lin, Christopher Wilhelm, Kyle Lo, and Luca Soldaini. olmocr: Unlocking trillions of tokens in pdfs with vision language models. *arXiv preprint arXiv:2502.18443*, 2025.

[96] Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

[97] Jiacheng Lin, Zhenbang Wu, and Jimeng Sun. Training llms for ehr-based reasoning tasks via reinforcement learning. *arXiv preprint arXiv:2505.24105*, 2025.

[98] Sahil Kale and Devendra Singh Dhami. Knowrl: Teaching language models to know what they know. *arXiv preprint arXiv:2510.11407*, 2025.

[99] Evan Ellis, Vivek Myers, Jens Tuyls, Sergey Levine, Anca Dragan, and Benjamin Eysenbach. Training llm agents to empower humans. *arXiv preprint arXiv:2510.13709*, 2025.

[100] Jiashuo Sun, Shixuan Liu, Zhaochen Su, Xianrui Zhong, Pengcheng Jiang, Bowen Jin, Peiran Li, Weijia Shi, and Jiawei Han. Grace: Generative representation learning via contrastive policy optimization. *arXiv preprint arXiv:2510.04506*, 2025.

[101] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=CjwERcAU7w.

[102] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025.

[103] Guowei Xu, Mert Yuksekgonul, Carlos Guestrin, and James Zou. metatextgrad: Automatically optimizing language model optimizers. *arXiv preprint arXiv:2505.18524*, 2025.

[104] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2024.

[105] Yuanjie Lyu, Zihan Niu, Zheyong Xie, Chao Zhang, Tong Xu, Yang Wang, and Enhong Chen. Retrieve-plan-generation: An iterative planning and answering framework for knowledge-intensive llm generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4683–4702, 2024.

[106] Pengcheng Jiang, Lang Cao, Ruike Zhu, Minhao Jiang, Yunyi Zhang, Jimeng Sun, and Jiawei Han. Ras: Retrieval-and-structuring for knowledge-intensive llm generation. *arXiv preprint arXiv: 2502.10996*, 2025.

[107] Xinyan Guan, Jiali Zeng, Fandong Meng, Chunlei Xin, Yaojie Lu, Hongyu Lin, Xianpei Han, Le Sun, and Jie Zhou. Deeprag: Thinking to retrieve step by step for large language models. *arXiv preprint arXiv:2502.01142*, 2025.

[108] Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

[109] Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z Pan, Wen Zhang, Huajun Chen, et al. Learning to reason with search for llms via reinforcement learning. *arXiv preprint arXiv:2503.19470*, 2025.

[110] Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*, 2025.

[111] Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. Zerosearch: Incentivize the search capability of llms without searching. *arXiv preprint arXiv:2505.04588*, 2025.

[112] Ziliang Wang, Xuhui Zheng, Kang An, Cijun Ouyang, Jialu Cai, Yuhang Wang, and Yichao Wu. Stepsearch: Igniting llms search ability via step-wise proximal policy optimization. *arXiv preprint arXiv:2505.15107*, 2025.

[113] Chuzhan Hao, Wenfeng Feng, Yuewei Zhang, and Hao Wang. Dynasearcher: Dynamic knowledge graph augmented search agent via multi-reward reinforcement learning. *arXiv preprint arXiv:2507.17365*, 2025.

[114] Ailing Yu, Lan Yao, Jingnan Liu, Zhe Chen, Jiajun Yin, Yuan Wang, Xinhao Liao, Zhiling Ye, Ji Li, Yun Yue, et al. Medresearcher-r1: Expert-level medical deep researcher via a knowledge-informed trajectory synthesis framework. *arXiv preprint arXiv:2508.14880*, 2025.

[115] Yaorui Shi, Shihan Li, Chang Wu, Zhiyuan Liu, Junfeng Fang, Hengxing Cai, An Zhang, and Xiang Wang. Search and refine during think: Autonomous retrieval-augmented reasoning of llms. *arXiv e-prints*, pages arXiv–2505, 2025.

[116] Jinming Wu, Zihao Deng, Wei Li, Yiding Liu, Bo You, Bo Li, Zejun Ma, and Ziwei Liu. Mmsearch-r1: Incentivizing lmms to search. *arXiv preprint arXiv:2506.20670*, 2025.

[117] Qingyao Li, Xinyi Dai, Xiangyang Li, Weinan Zhang, Yasheng Wang, Ruiming Tang, and Yong Yu. Codeprm: Execution feedback-enhanced process reward model for code generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8169–8182, 2025.

[118] Yifei Zhou, Sergey Levine, Jason Weston, Xian Li, and Sainbayar Sukhbaatar. Self-challenging language model agents. *arXiv preprint arXiv:2506.01716*, 2025.

[119] Zi-Yi Dou, Cheng-Fu Yang, Xueqing Wu, Kai-Wei Chang, and Nanyun Peng. Re-rest: Reflection-reinforced self-training for language agents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15394–15411, 2024.

[120] Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*, 2025.

[121] Emre Can Acikgoz, Cheng Qian, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. Self-improving llm agents at test-time. *arXiv preprint arXiv:2510.07841*, 2025.

[122] Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K Qiu, and Yuqing Yang. Agent lightning: Train any ai agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025.

[123] Qianben Chen, Jingyi Cao, Jiayu Zhang, Tianrui Qin, Xiaowan Li, King Zhu, Dingfeng Shi, He Zhu, Minghao Liu, Xiaobo Liang, et al. A2fm: An adaptive agent foundation model for tool-aware hybrid reasoning. *arXiv e-prints*, pages arXiv–2510, 2025.

[124] Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, et al. Verltool: Towards holistic agentic reinforcement learning with tool use. *arXiv preprint arXiv:2509.01055*, 2025.

[125] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[126] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

[127] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11888–11898, 2023.

[128] Keyan Ding, Jing Yu, Junjie Huang, Yuchen Yang, Qiang Zhang, and Huajun Chen. Scitoolagent: a knowledge-graph-driven scientific agent for multitool integration. *Nature Computational Science*, pages 1–11, 2025.

[129] Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023.

[130] Anthropic. Code execution with mcp: Building more efficient ai agents, November 2025. URL https://www.anthropic.com/engineering/code-execution-with-mcp. Published Nov 04 2025.

[131] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

[132] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4015–4026, 2023.

[133] Haoxiang Wang, Pavan Kumar Anasosalu Vasu, Fartash Faghri, Raviteja Vemulapalli, Mehrdad Farajtabar, Sachin Mehta, Mohammad Rastegari, Oncel Tuzel, and Hadi Pouransari. Sam-clip: Merging vision foundation models towards semantic and spatial understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3635–3647, 2024.

[134] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning*, pages 28492–28518. PMLR, 2023.

[135] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781, 2020.

[136] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48, 2020.

[137] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

[138] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

[139] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

[140] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023.

[141] Tian Xie and Jeffrey C Grossman. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters*, 120(14):145301, 2018.

[142] Bohao Xu, Yingzhou Lu, Chenhao Li, Ling Yue, Xiao Wang, Nan Hao, Tianfan Fu, and Jim Chen. Smiles-mamba: Chemical mamba foundation models for drug admet prediction. *arXiv preprint arXiv:2408.05696*, 2024.

[143] Wengong Jin, Connor W Coley, Regina Barzilay, and Tommi Jaakkola. Predicting organic reaction outcomes with weisfeiler-lehman network. *arXiv preprint arXiv:1709.04555*, 2017.

[144] Wengong Jin, Kevin Yang, Regina Barzilay, and Tommi Jaakkola. Learning multimodal graph-to-graph translation for molecular optimization. *arXiv preprint arXiv:1812.01070*, 2018.

[145] Shuangjia Zheng, Jiahua Rao, Zhongyue Zhang, Jun Xu, and Yuedong Yang. Predicting retrosynthetic reactions using self-corrected transformer neural networks. *Journal of Chemical Information and Modeling*, 60(1):47–55, 2019.

[146] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data, 2020.

[147] Yin Fang, Qiang Zhang, Ningyu Zhang, Zhuo Chen, Xiang Zhuang, Xin Shao, Xiaohui Fan, and Huajun Chen. Knowledge graph-enhanced molecular contrastive learning with functional prompt. *Nature Machine Intelligence*, pages 1–12, 2023.

[148] Pengcheng Jiang, Cao Xiao, Tianfan Fu, Jimeng Sun, and Jiawei Han. Bi-level contrastive learning for knowledge-enhanced molecule representations. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence*, 2025.

[149] Xiaoning Qi, Lianhe Zhao, Chenyu Tian, Yueyue Li, Zhen-Lin Chen, Peipei Huo, Runsheng Chen, Xiaodong Liu, Baoping Wan, Shengyong Yang, et al. Predicting transcriptional responses to novel chemical perturbations using deep generative model for drug discovery. *Nature Communications*, 15(1):9256, 2024.

[150] Xiaochu Tong, Ning Qu, Xiangtai Kong, Shengkun Ni, Jingyi Zhou, Kun Wang, Lehan Zhang, Yiming Wen, Jiangshan Shi, Sulin Zhang, et al. Deep representation learning of chemical-induced transcriptional profile for phenotype-based drug discovery. *Nature Communications*, 15(1):5378, 2024.

[151] Haitao Li, Qingyao Ai, Jia Chen, Qian Dong, Zhijing Wu, and Yiqun Liu. Blade: Enhancing black-box large language models with small domain-specific models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24422–24430, 2025.

[152] Haotian Sun, Yuchen Zhuang, Wei Wei, Chao Zhang, and Bo Dai. Bbox-adapter: Lightweight adapting for black-box large language models. *arXiv preprint arXiv:2402.08219*, 2024.

[153] Alisa Liu, Xiaochuang Han, Yizhong Wang, Yulia Tsvetkov, Yejin Choi, and Noah A Smith. Tuning language models by proxy. In *First Conference on Language Modeling*, 2024.

[154] Hongjin Su, Shuyang Jiang, Yuhang Lai, Haoyuan Wu, Boao Shi, Che Liu, Qian Liu, and Tao Yu. Evor: Evolving retrieval for code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2538–2554, 2024.

[155] Xi Victoria Lin, Xilun Chen, Mingda Chen, Weijia Shi, Maria Lomeli, Richard James, Pedro Rodriguez, Jacob Kahn, Gergely Szilvasy, Mike Lewis, et al. Ra-dit: Retrieval-augmented dual instruction tuning. In *The Twelfth International Conference on Learning Representations*, 2023.

[156] Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Haotian Sun, Hang Wu, Carl Yang, and May D Wang. Medadapter: Efficient test-time adaptation of large language models towards medical reasoning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing. Conference on Empirical Methods in Natural Language Processing*, volume 2024, page 22294, 2024.

[157] Jaehyung Kim, Dongyoung Kim, and Yiming Yang. Learning to correct for qa reasoning with black-box llms. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8916–8937, 2024.

[158] Lingxi Zhang, Yue Yu, Kuan Wang, and Chao Zhang. Arl2: Aligning retrievers with black-box large language models via self-guided adaptive relevance labeling. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3708–3719, 2024.

[159] Daixuan Cheng, Shaohan Huang, Junyu Bi, Yuefeng Zhan, Jianfeng Liu, Yujing Wang, Hao Sun, Furu Wei, Weiwei Deng, and Qi Zhang. Uprise: Universal prompt retrieval for improving zero-shot evaluation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12318–12337, 2023.

[160] Zixuan Ke, Weize Kong, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. Bridging the preference gap between retrievers and llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10438–10451, 2024.

[161] Jiashuo Sun, Xianrui Zhong, Sizhe Zhou, and Jiawei Han. Dynamicrag: Leveraging outputs of large language model as feedback for dynamic reranking in retrieval-augmented generation. *arXiv preprint arXiv:2505.07233*, 2025.

[162] Yi Jiang, Lei Shen, Lujie Niu, Sendong Zhao, Wenbo Su, and Bo Zheng. Qagent: A modular search agent with interactive query understanding. *arXiv preprint arXiv:2510.08383*, 2025.

[163] Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-{\alpha}: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025.

[164] Hong Ting Tsang, Jiaxin Bai, Haoyu Huang, Qiao Xiao, Tianshi Zheng, Baixuan Xu, Shujie Liu, and Yangqiu Song. Autograph-r1: End-to-end reinforcement learning for knowledge graph construction. *arXiv preprint arXiv:2510.15339*, 2025.

[165] Yilin Xiao, Junnan Dong, Chuang Zhou, Su Dong, Qian-wen Zhang, Di Yin, Xing Sun, and Xiao Huang. Graphrag-bench: Challenging domain-specific reasoning for evaluating graph retrieval-augmented generation. *arXiv preprint arXiv:2506.02404*, 2025.

[166] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

[167] Parth Asawa, Alan Zhu, Matei Zaharia, Alexandros G Dimakis, and Joseph E Gonzalez. How to train your advisor: Steering black-box llms with advisor models. *arXiv preprint arXiv:2510.02453*, 2025.

[168] ChangHao Li, Yuchen Zhuang, Rushi Qiang, Haotian Sun, Hanjun Dai, Chao Zhang, and Bo Dai. Matryoshka pilot: Learning to drive black-box llms with llms. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[169] Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

[170] Yixing Chen, Yiding Wang, Siqi Zhu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. Multi-agent evolve: Llm self-improve through co-evolution. *arXiv preprint arXiv:2510.23595*, 2025.

[171] Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025.

[172] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.

[173] Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, et al. Rethinking memory mechanisms of foundation agents in the second half. *arXiv preprint arXiv:2602.06052*, 2026.

[174] Dongming Jiang, Yi Li, Songtao Wei, Jinxin Yang, Ayushi Kishore, Alysa Zhao, Dingyi Kang, Xu Hu, Feng Chen, Qiannan Li, et al. Anatomy of agentic memory: Taxonomy and empirical analysis of evaluation and system limitations. *arXiv preprint arXiv:2602.19320*, 2026.

[175] Xun Jiang, Feng Li, Han Zhao, Jiahao Qiu, Jiaying Wang, Jun Shao, Shihao Xu, Shu Zhang, Weiling Chen, Xavier Tang, et al. Long term memory: The foundation of ai self-evolution. *arXiv preprint arXiv:2410.15665*, 2024.

[176] Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2023.

[177] Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *arXiv preprint arXiv:2602.12430*, 2026.

[178] Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*, 2025.

[179] Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. *arXiv preprint arXiv:2512.10696*, 2025.

[180] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

[181] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[182] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731, 2024.

[183] Junru Lu, Siyu An, Mingbao Lin, Gabriele Pergola, Yulan He, Di Yin, Xing Sun, and Yunsheng Wu. Memochat: Tuning llms to use memos for consistent long-range open-domain conversation. *arXiv preprint arXiv:2308.08239*, 2023.

[184] Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Ret-llm: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322*, 2023.

[185] Xinnian Liang, Bing Wang, Hui Huang, Shuangzhi Wu, Peihao Wu, Lu Lu, Zejun Ma, and Zhoujun Li. Unleashing infinite-length input capacity for large-scale language models with self-controlled memory system. *arXiv preprint arXiv:2304.13343*, 10, 2023.

[186] Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*, 2025.

[187] Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. pages 25972–25981, 2025.

[188] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[189] Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569, 2024.

[190] Tooraj Helmi. Decentralizing ai memory: Shimi, a semantic hierarchical memory index for scalable agent reasoning. *arXiv preprint arXiv:2504.06135*, 2025.

[191] Na Liu, Liangyu Chen, Xiaoyu Tian, Wei Zou, Kaijiang Chen, and Ming Cui. From llm to conversational agent: A memory enhanced architecture with fine-tuning of large language models. *arXiv preprint arXiv:2401.02777*, 2024.

[192] Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh RN, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, et al. Retroformer: Retrospective large language agents with policy gradient optimization. In *The Twelfth International Conference on Learning Representations*, 2023.

[193] Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang. Think-in-memory: Recalling and post-thinking enable llms with long-term memory. *arXiv preprint arXiv:2311.08719*, 2023.

[194] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

[195] Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Xufang Luo, Hao Cheng, Dongsheng Li, Yuqing Yang, Chin-Yew Lin, H Vicky Zhao, Lili Qiu, et al. On memory construction and retrieval for personalized conversational agents. *arXiv preprint arXiv:2502.05589*, 2025.

[196] Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. *arXiv preprint arXiv:2407.04363*, 2024.

[197] Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. Chatdb: Augmenting llms with databases as their symbolic memory. *arXiv preprint arXiv:2306.03901*, 2023.

[198] Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. *arXiv preprint arXiv:2410.14052*, 2024.

[199] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

[200] Xiaoxia Cheng, Zeqi Tan, Wei Xue, and Weiming Lu. Information re-organization improves reasoning in large language models. *Advances in Neural Information Processing Systems*, 37:130214–130236, 2024.

[201] Hongkang Yang, Zehao Lin, Wenjin Wang, Hao Wu, Zhiyu Li, Bo Tang, Wenqiang Wei, Jinbo Wang, Zeyun Tang, Shichao Song, et al. Memory^3: Language modeling with explicit memory. *arXiv preprint arXiv:2407.01178*, 2024.

[202] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

[203] Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026.

[204] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

[205] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

[206] Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

[207] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

[208] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

[209] Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. *arXiv preprint arXiv:2504.06821*, 2025.

[210] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[211] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023.

[212] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023.

[213] Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, et al. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(3):1894–1907, 2024.

[214] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642, 2024.

[215] Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*, 2024.

[216] Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024.

[217] Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, et al. Cradle: Empowering foundation agents towards general computer control. *arXiv preprint arXiv:2403.03186*, 2024.

[218] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. pages 1–20, 2025.

[219] Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*, 2024.

[220] Chengyou Jia, Minnan Luo, Zhuohang Dang, Qiushi Sun, Fangzhi Xu, Junlin Hu, Tianbao Xie, and Zhiyong Wu. Agentstore: Scalable integration of heterogeneous agents as specialized generalist computer assistant. pages 8908–8934, 2025.

[221] Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. Exact: Teaching ai agents to explore with reflective-mcts and exploratory learning. *arXiv preprint arXiv:2410.02052*, 2024.

[222] Xufeng Zhao, Cornelius Weber, and Stefan Wermter. Agentic skill discovery. *Robotics and Autonomous Systems*, page 105248, 2025.

[223] Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

[224] Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Li Erran Li. Proposer-agent-evaluator (pae): Autonomous skill discovery for foundation model internet agents. In *Forty-second International Conference on Machine Learning*, 2025.

[225] Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. *arXiv preprint arXiv:2512.17102*, 2025.

[226] Xu Huang, Junwu Chen, Yuxing Fei, Zhuohan Li, Philippe Schwaller, and Gerbrand Ceder. Cascade: Cumulative agentic skill creation through autonomous development and evolution. *arXiv preprint arXiv:2512.23880*, 2025.

[227] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. 2023.

[228] Cheng Qian, Chi Han, Yi Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6922–6939, 2023.

[229] Dongge Han, Camille Couturier, Daniel Madrigal Diaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. Legomem: Modular procedural memory for multi-agent llm systems for workflow automation. *arXiv preprint arXiv:2510.04851*, 2025.

[230] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

[231] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[232] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

[233] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[234] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023.

[235] Howard Chen, Ramakanth Pasunuru, Jason Weston, and Asli Celikyilmaz. Walking down the memory maze: Beyond context limit through interactive reading. *arXiv preprint arXiv:2310.05029*, 2023.

[236] Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint arXiv:2402.09727*, 2024.

[237] Sangwoo Park, Jinheon Baek, Soyeong Jeong, and Sung Ju Hwang. Chain of retrieval: Multi-aspect iterative search expansion and post-order search aggregation for full paper retrieval. *arXiv preprint arXiv:2507.10057*, 2025.

[238] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. pages 7036–7050, 2024.

[239] Ziyang Wang, Heba Elfardy, Markus Dreyer, Kevin Small, and Mohit Bansal. Unified embeddings for multimodal retrieval via frozen llms. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1537–1547, 2024.

[240] Song Tang, Wenxin Su, Mao Ye, and Xiatian Zhu. Source-free domain adaptation with frozen multimodal foundation model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23711–23720, 2024.

[241] Lei Zhu, Fangyun Wei, and Yanye Lu. Beyond text: Frozen large language models in visual signal comprehension. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27047–27057, 2024.

[242] Kartik Sharma, Yiqiao Jin, Vineeth Rakesh, Yingtong Dou, Menghai Pan, Mahashweta Das, and Srijan Kumar. Sysformer: Safeguarding frozen large language models with adaptive system prompts. *arXiv preprint arXiv:2506.15751*, 2025.

[243] Ben Pan, Carlo Baronio, Albert Tam, Pietro Marsella, Mokshit Jain, Swyx, and Silas Alberti. Introducing swe-grep and swe-grep-mini: Rl for multi-turn, fast context retrieval. https://cognition.ai/blog/swe-grep, October 2025. Accessed: 2025-10-29.

[244] Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[245] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[246] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

[247] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037, 2023.

[248] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[249] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[250] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

[251] Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=KfTf9vFvSn.

[252] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. 2023.

[253] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. 2023.

[254] David Schmotz, Luca Beurer-Kellner, Sahar Abdelnabi, and Maksym Andriushchenko. Skill-inject: Measuring agent vulnerability to skill file attacks. *arXiv preprint arXiv:2602.20156*, 2026.

[255] Yi Liu, Weizhe Wang, Ruitao Feng, Yao Zhang, Guangquan Xu, Gelei Deng, Yuekang Li, and Leo Zhang. Agent skills in the wild: An empirical study of security vulnerabilities at scale. *arXiv preprint arXiv:2601.10338*, 2026.

[256] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

[257] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. {τ}-bench: A benchmark for \underline{T}ool-\underline{A}gent-\underline{U}ser interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

[258] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. Tau-2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

[259] Jize Wang, Zerun Ma, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. Gta: a benchmark for general tool agents. volume 37, pages 75749–75790, 2024.

[260] Keyu Li, Junhao Shi, Yang Xiao, Mohan Jiang, Jie Sun, Yunze Wu, Shijie Xia, Xiaojie Cai, Tianze Xu, Weiye Si, et al. Agencybench: Benchmarking the frontiers of autonomous agents in 1m-token real-world contexts. *arXiv preprint arXiv:2601.11044*, 2026.

[261] Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, et al. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *arXiv preprint arXiv:2510.25726*, 2025.

[262] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

[263] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

[264] Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, et al. Are we done with mmlu? In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5069–5096, 2025.

[265] Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2299–2314, 2024.

[266] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, 2023.

[267] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[268] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 3214–3252, 2022.

[269] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[270] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, 2019.

[271] Minh-Thang Luong, Dawsen Hwang, Hoang H Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, et al. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 35406–35430, 2025.

[272] Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. TheoremQA: A theorem-driven question answering dataset. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=Wom397PB55.

[273] Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SMa9EAovKMC.

[274] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

[275] Minyang Tian, Luyu Gao, Shizhuo Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, et al. Scicode: A research coding benchmark curated by scientists. *Advances in Neural Information Processing Systems*, 37:30624–30650, 2024.

[276] Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. *arXiv preprint arXiv:2507.02833*, 2025.

[277] Kaustubh Deshpande, Ved Sirdeshmukh, Johannes Baptist Mols, Lifeng Jin, Ed-Yeremai Hernandez-Cardona, Dean Lee, Jeremy Kritz, Willow E Primack, Summer Yue, and Chen Xing. Multichallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier llms. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 18632–18702, 2025.

[278] Artificial Analysis Team. Artificial analysis long context reasoning benchmark(lcr), 2025.

[279] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

[280] OpenAI. Mrcr: Multi-round co-reference resolution dataset. https://huggingface.co/datasets/openai/mrcr, 2024. Accessed: 2025-12-17.

[281] Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

[282] Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, et al. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025.

[283] The Terminal-Bench Team. Terminal-bench: A benchmark for ai agents in terminal environments, Apr 2025. URL https://github.com/laude-institute/terminal-bench.

[284] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk.

[285] AgentBeats Authors. Agentified agent assessment (aaa) & agentbeats. https://docs.agentbeats.org/, 2025. Accessed: 2025-12-17.

[286] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. 2021.

[287] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[288] Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, et al. Holistic agent leaderboard: The missing infrastructure for ai agent evaluation. *arXiv preprint arXiv:2510.11977*, 2025.

[289] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. volume 36, pages 46595–46623, 2023.

[290] Yichuan Ma, Linyang Li, Peiji Li, Xiaozhe Li, Qipeng Guo, Dahua Lin, Kai Chen, et al. Timely machine: Awareness of time makes test-time scaling agentic. *arXiv preprint arXiv:2601.16486*, 2026.

[291] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905, 2024.

[292] Howard Chen, Noam Razin, Karthik Narasimhan, and Danqi Chen. Retaining by doing: The role of on-policy data in mitigating forgetting. *arXiv preprint arXiv:2510.18874*, 2025.

[293] Zepeng Ning and Lihua Xie. A survey on multi-agent reinforcement learning and its application. *Journal of Automation and Intelligence*, 3(2):73–91, 2024.

[294] Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

[295] Zhezheng Hao, Hong Wang, Haoyang Liu, Jian Luo, Jiarui Yu, Hande Dong, Qiang Lin, Can Wang, and Jiawei Chen. Rethinking entropy interventions in rlvr: An entropy change perspective. *arXiv preprint arXiv:2510.10150*, 2025.

[296] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox. 2023.

[297] Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, et al. R-judge: Benchmarking safety risk awareness for llm agents. pages 1467–1490, 2024.

[298] Jiayi Fu, Xuandong Zhao, Chengyuan Yao, Heng Wang, Qi Han, and Yanghua Xiao. Reward shaping to mitigate reward hacking in rlhf. *arXiv preprint arXiv:2502.18770*, 2025.

[299] Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. *arXiv preprint arXiv:2404.06474*, 2024.

[300] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[301] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

[302] OpenAI. Introducing deep research. https://openai.com/index/introducing-deep-research/, 2025.

[303] Anthropic. Claude takes research to new places. https://www.anthropic.com/news/research, 2025.

[304] Dave Citron. Deep research is now available on gemini 2.5 pro experimental. https://blog.google/products/gemini/deep-research-gemini-2-5-pro-experimental/, 2025.

[305] Cursor. Cursor - the ai code editor. https://www.cursor.com/, 2025. Accessed: 2025-10-29.

[306] Anthropic. Claude code: Deep coding at terminal velocity. https://www.anthropic.com/claude-code, 2025. Accessed: 2025-10-29.

[307] OpenAI. Codex. https://openai.com/codex/, 2025. Accessed: 2025-10-29.

[308] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[309] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

[310] Jacob Jackson, Phillip Kravtsov, and Shomil Jain. Improving cursor tab with online reinforcement learning. https://cursor.com/blog/tab-rl, September 2025. Accessed: 2025-10-29.

[311] OpenAI. Computer-using agent. https://openai.com/index/computer-using-agent/, January 2025. Accessed: 2025-10-28.

[312] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16022–16076, 2024.

[313] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890, 2024.

[314] Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Zheng Boyuan, LI PEIHANG, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Hu Jiarui, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Y.Charles, Zhilin Yang, and Tao Yu. OpenCUA: Open foundations for computer-use agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=6iRZvJiC9Q.

[315] Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=EEgYUccwsV.

[316] Tianyi Chen, Yinheng Li, Michael Solodko, Sen Wang, Nan Jiang, Tingyuan Cui, Junheng Hao, Jongwoo Ko, Sara Abdali, Suzhen Zheng, et al. Cua-skill: Develop skills for computer using agent. *arXiv preprint arXiv:2601.21123*, 2026.

[317] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

[318] Zhizheng Wang, Qiao Jin, Chih-Hsuan Wei, Shubo Tian, Po-Ting Lai, Qingqing Zhu, Chi-Ping Day, Christina Ross, Robert Leaman, and Zhiyong Lu. Geneagent: self-verification language agent for gene-set analysis using domain databases. *Nature Methods*, pages 1–9, 2025.

[319] Zifeng Wang, Benjamin Danek, Ziwei Yang, Zheng Chen, and Jimeng Sun. Making large language models reliable data science programming copilot for biomedical research. *Nature Biomedical Engineering*, 2025.

[320] Kyle Swanson, Wesley Wu, Nash L Bulaong, John E Pak, and James Zou. The virtual lab of ai agents designs new sars-cov-2 nanobodies. *Nature*, pages 1–3, 2025.

[321] Zifeng Wang, Lang Cao, Qiao Jin, Joey Chan, Nicholas Wan, Behdad Afzali, Hyun-Jin Cho, Chang-In Choi, Mehdi Emamverdi, Manjot K Gill, et al. A foundation model for human-ai collaboration in medical literature mining. *Nature Communications*, 16(1):8361, 2025.

[322] Haoyang Li, Weishen Pan, Suraj Rajendran, Chengxi Zang, and Fei Wang. Trialgenie: Empowering clinical trial design with agentic intelligence and real world data. *medRxiv*, pages 2025–04, 2025.

[323] Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf Roohani, Ryan Li, Lin Qiu, Gavin Li, Junze Zhang, et al. Biomni: A general-purpose biomedical ai agent. *biorxiv*, 2025.

[324] Kyle Swanson, Gary Liu, Denise B Catacutan, Autumn Arnold, James Zou, and Jonathan M Stokes. Generative ai for designing and validating easily synthesizable and structurally novel antibiotics. *Nature machine intelligence*, 6(3): 338–353, 2024.

[325] Aarti Krishnan, Jacqueline A Valeri, Wengong Jin, Nina M Donghia, Leif Sieben, Andreas Luttens, Yu Zhang, Seyed Majed Modaresi, Andrew Hennes, Jenna Fromer, et al. A generative deep learning approach to de novo antibiotic design. *Cell*, 2025.

[326] Shanghua Gao, Richard Zhu, Pengwei Sui, Zhenglun Kong, Sufian Aldogom, Yepeng Huang, Ayush Noori, Reza Shamji, Krishna Parvataneni, Theodoros Tsiligkaridis, et al. Democratizing ai scientists using tooluniverse. *arXiv preprint arXiv:2509.23426*, 2025.

[327] Ruofan Jin, Zaixi Zhang, Mengdi Wang, and Le Cong. Stella: Self-evolving llm agent for biomedical research. *arXiv preprint arXiv:2507.02004*, 2025.

[328] W Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena*, 42(1-3):228–234, 1990.

[329] Christopher D. Rosin and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.

[330] Mitchell A. Potter and Kenneth A. De Jong. Cooperative coevolution: An architecture for evolving co-adapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[331] Louis Sushil and collaborating authors. A comprehensive survey of coevolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 2008. Survey of competitive, cooperative, and multi-population CEA frameworks.

[332] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

[333] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

[334] Liang Zhou, Rohan Patel, and Seong Kim. Multi-agent tool-integrated policy optimization. *arXiv preprint arXiv:2510.04678*, 2025. URL https://arxiv.org/pdf/2510.04678. Accessed: 2025-11-17.

[335] Jiajun Wang, Shurui Liu, and Tianyi Zhang. A joint optimization framework for enhancing efficiency of tool utilization in llm agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025. URL https://aclanthology.org/2025.findings-acl.1149.pdf. Accessed: 2025-11-17.

[336] Rui Huang, Ashok Kumar, and Sandip Sen. A design framework for scalable and adaptive multi-agent coordination in dynamic environments. *IEEE Transactions on Systems, Man, and Cybernetics*, 2023. URL https://ieeexplore.ieee.org/iel8/6287639/10820123/10965637. Accessed: 2025-11-17.

[337] Joon Lee, Robert Martens, and Yifan Du. From chaos to symbiosis: Exploring adaptive co-evolution strategies for hybrid intelligent systems. *Complexity*, 2024. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC12465495/. Accessed: 2025-11-17.

[338] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5362–5383, 2024.

[339] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

[340] Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. Continual learning of large language models: A comprehensive survey. *ACM Computing Surveys*, 2024.

[341] Jiacheng Lin, Zhongruo Wang, Kun Qian, Tian Wang, Arvind Srinivasan, Hansi Zeng, Ruochen Jiao, Xie Zhou, Jiri Gesi, Dakuo Wang, et al. Sft doesn't always hurt general capabilities: Revisiting domain-specific fine-tuning in llms. *arXiv preprint arXiv:2509.20758*, 2025.

[342] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[343] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[344] Yichen Wu, Long-Kai Huang, Renzhen Wang, Deyu Meng, and Ying Wei. Meta continual learning revisited: Implicitly enhancing online hessian approximation via variance reduction. In *The Twelfth international conference on learning representations*, volume 2, 2024.

[345] Shipeng Wang, Xiaorong Li, Jian Sun, and Zongben Xu. Training networks in null space of feature covariance for continual learning. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 184–193, 2021.

[346] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International conference on artificial intelligence and statistics*, pages 3762–3773. PMLR, 2020.

[347] Yichen Wu, Hongming Piao, Long-Kai Huang, Renzhen Wang, Wanhua Li, Hanspeter Pfister, Deyu Meng, Kede Ma, and Ying Wei. Sd-lora: Scalable decoupled low-rank adaptation for class incremental learning. In *ICLR*, 2025.

[348] Yan-Shuo Liang and Wu-Jun Li. Inflora: Interference-free low-rank adaptation for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23638–23647, 2024.

[349] Liyuan Wang, Jingyi Xie, Xingxing Zhang, Mingyi Huang, Hang Su, and Jun Zhu. Hierarchical decomposition of prompt-based continual learning: Rethinking obscured sub-optimality. *Advances in Neural Information Processing Systems*, 36:69054–69076, 2023.

[350] Shujun Xia, Haokun Lin, Yichen Wu, Yinan Zhou, Zixuan Li, Zhongwei Wan, Xingrun Xing, Yefeng Zheng, Xiang Li, Caifeng Shan, et al. Medrek: Retrieval-based editing for medical llms with key-aware prompts. In *Socially Responsible and Trustworthy Foundation Models at NeurIPS 2025*, 2025.

[351] Daniel Marczak, Bartłomiej Twardowski, Tomasz Trzciński, and Sebastian Cygert. Magmax: Leveraging model merging for seamless continual learning. In *European Conference on Computer Vision*, pages 379–395. Springer, 2024.

[352] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019.

[353] Jihwan Bang, Heesu Kim, YoungJoon Yoo, Jung-Woo Ha, and Jonghyun Choi. Rainbow memory: Continual learning with a memory of diverse samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8218–8227, 2021.

[354] Quanziang Wang, Renzhen Wang, Yichen Wu, Xixi Jia, and Deyu Meng. Cba: Improving online continual learning via continual bias adaptor. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 19082–19092, 2023.

[355] Yichen Wu, Hong Wang, Peilin Zhao, Yefeng Zheng, Ying Wei, and Long-Kai Huang. Mitigating catastrophic forgetting in online continual learning by modeling previous task interrelations via pareto optimization. In *Forty-first international conference on machine learning*, 2024.

[356] Liyuan Wang, Xingxing Zhang, Kuo Yang, Longhui Yu, Chongxuan Li, Lanqing HONG, Shifeng Zhang, Zhenguo Li, Yi Zhong, and Jun Zhu. Memory replay with data compression for continual learning. In *International Conference on Learning Representations*, 2022.

[357] Quang Pham, Chenghao Liu, and Steven Hoi. Dualnet: Continual learning, fast and slow. *Advances in Neural Information Processing Systems*, 34:16131–16144, 2021.

[358] Elahe Arani, Fahad Sarfraz, and Bahram Zonooz. Learning fast, learning slow: A general continual learning method based on complementary learning system. In *International Conference on Learning Representations*, 2022.

[359] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 139–149, 2022.

[360] Zhiyi Shi, Binjie Wang, Chongjie Si, Yichen Wu, Junsik Kim, and Hanspeter Pfister. Dualedit: Dual editing for knowledge updating in vision-language models. *arXiv preprint arXiv:2506.13638*, 2025.

[361] Hongming Piao, Yichen Wu, Dapeng Wu, and Ying Wei. Federated continual learning via prompt-based dual knowledge transfer. In *Forty-first International Conference on Machine Learning*, 2024.

[362] Adarsh Kumarappan, Mo Tiwari, Peiyang Song, Robert Joseph George, Chaowei Xiao, and Anima Anandkumar. Leanagent: Lifelong learning for formal theorem proving. *arXiv preprint arXiv:2410.06209*, 2024.

[363] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[364] Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve. Rlef: Grounding code llms in execution feedback with reinforcement learning. *arXiv preprint arXiv:2410.02089*, 2024.

[365] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[366] Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Jan Leike, and Shane Legg. Specification gaming: the flip side of ai ingenuity. *DeepMind Blog*, 2020.

[367] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[368] Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, and Alois Knoll. A review of safe reinforcement learning: Methods, theories and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[369] Kaiwen Zhou, Chengzhi Liu, Xuandong Zhao, Shreedhar Jangam, Jayanth Srinivasa, Gaowen Liu, Dawn Song, and Xin Eric Wang. The hidden risks of large reasoning models: A safety assessment of r1. *arXiv preprint arXiv:2502.12659*, 2025.

[370] Paul Kassianik and Amin Karbasi. Evaluating security risk in deepseek and other frontier reasoning models. *Cisco Blogs, Cisco Systems*, 31, 2025.

[371] David Manheim and Scott Garrabrant. Categorizing variants of goodhart's law. *arXiv preprint arXiv:1803.04585*, 2018.

[372] Alexander Bondarenko, Denis Volk, Dmitrii Volkov, and Jeffrey Ladish. Demonstrating specification gaming in reasoning models. *arXiv preprint arXiv:2502.13295*, 2025.

[373] Shuli Zhao, Qinsheng Hou, Zihan Zhan, Yanhao Wang, Yuchong Xie, Yu Guo, Libo Chen, Shenghong Li, and Zhi Xue. Mind your server: A systematic study of parasitic toolchain attacks on the mcp ecosystem. *arXiv preprint arXiv:2509.06572*, 2025.

[374] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*, pages 79–90, 2023.

[375] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? *Advances in Neural Information Processing Systems*, 36:80079–80110, 2023.

[376] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

[377] Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. Learning to be safe: Deep rl with a safety critic. *arXiv preprint arXiv:2010.14603*, 2020.

[378] Nathan Hunt, Nathan Fulton, Sara Magliacane, Trong Nghia Hoang, Subhro Das, and Armando Solar-Lezama. Verifiably safe exploration for end-to-end reinforcement learning. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2021.

[379] Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Y Wu, and Zhifang Sui. Math-shepherd: A label-free step-by-step verifier for llms in mathematical reasoning. *arXiv preprint arXiv:2312.08935*, 3, 2023.

[380] Víctor Gallego. Specification self-correction: Mitigating in-context reward hacking through test-time refinement. *arXiv preprint arXiv:2507.18742*, 2025.

[381] SHengjie Ma, Chenlong Deng, Jiaxin Mao, Jiadeng Huang, Teng Wang, Junjie Wu, Changwang Zhang, et al. Pou: Proof-of-use to counter tool-call hacking in deepresearch agents. *arXiv preprint arXiv:2510.10931*, 2025.

[382] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

[383] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

[384] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023.

[385] Chongjie Si, Zhiyi Shi, Shifan Zhang, Xiaokang Yang, Hanspeter Pfister, and Wei Shen. Unleashing the power of task-specific directions in parameter efficient fine-tuning. In *The Thirteenth International Conference on Learning Representations*, 2024.

[386] Zhiyi Shi, Junsik Kim, Wanhua Li, Yicong Li, and Hanspeter Pfister. Mora: Lora guided multi-modal disease diagnosis with missing modality. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 273–282. Springer, 2024.

[387] Chongjie Si, Xuehui Wang, Xue Yang, Zhengqin Xu, Qingyun Li, Jifeng Dai, Yu Qiao, Xiaokang Yang, and Wei Shen. Flora: Low-rank core space for n-dimension. *arXiv preprint arXiv:2405.14739*, 10, 2024.

[388] Chongjie Si, Zhiyi Shi, Xuehui Wang, Yichen Xiao, Xiaokang Yang, and Wei Shen. Generalized tensor-based parameter-efficient fine-tuning via lie group transformations. *arXiv preprint arXiv:2504.00851*, 2025.

[389] John Schulman and Thinking Machines Lab. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250929. https://thinkingmachines.ai/blog/lora/.

[390] Liyuan Liu, Feng Yao, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Flashrl: 8bit rollouts, full power rl, August 2025. URL https://fengyao.notion.site/flash-rl.

[391] Dan Peng, Zhihui Fu, and Jun Wang. Pocketllm: Enabling on-device fine-tuning for personalized llms. *arXiv preprint arXiv:2407.01031*, 2024.

[392] Liang Li, Xingke Yang, Wen Wu, Hao Wang, Tomoaki Ohtsuki, Xin Fu, Miao Pan, and Xuemin Shen. Mobillm: Enabling llm fine-tuning on the mobile device via server assisted side tuning. *arXiv preprint arXiv:2502.20421*, 2025.

[393] Xiaopei Chen, Liang Li, Fei Ji, and Wen Wu. Memory-efficient split federated learning for llm fine-tuning on heterogeneous mobile devices. *arXiv preprint arXiv:2506.02940*, 2025.

[394] Mengwei Xu, Dongqi Cai, Yaozong Wu, Xiang Li, and Shangguang Wang. {FwdLLM}: Efficient federated finetuning of large language models with perturbed inferences. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 579–596, 2024.

[395] Vinay Venkatesh, Vamsidhar R Kamanuru, Lav Kumar, and Nikita Kothari. Edge-fit: Federated instruction tuning of quantized llms for privacy-preserving smart home environments. *arXiv preprint arXiv:2510.03284*, 2025.

[396] Guangji Bai, Yijiang Li, Zilinghan Li, Liang Zhao, and Kibaek Kim. Fedspallm: Federated pruning of large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8361–8373, 2025.

[397] Taiyi Wang, Zhihao Wu, Jianheng Liu, Jianye Hao, Jun Wang, and Kun Shao. Distrl: An asynchronous distributed reinforcement learning framework for on-device control agents. *arXiv preprint arXiv:2410.14803*, 2024.

[398] Weizhi Zhang, Xinyang Zhang, Chenwei Zhang, Liangwei Yang, Jingbo Shang, Zhepei Wei, Henry Peng Zou, Zijie Huang, Zhengyang Wang, Yifan Gao, Xiaoman Pan, Lian Xiong, Jingguo Liu, Philip S. Yu, and Xian Li. Personaagent: When large language model agents meet personalization at test time. In *First Workshop on Multi-Turn Interactions in Large Language Models*, 2025. URL https://openreview.net/forum?id=fgCOkyJG3f.

[399] Vinay Samuel, Henry Peng Zou, Yue Zhou, Shreyas Chaudhari, Ashwin Kalyan, Tanmay Rajpurohit, Ameet Deshpande, Karthik Narasimhan, and Vishvak Murahari. Personagym: Evaluating persona agents and llms. *arXiv preprint arXiv:2407.18416*, 2024.

[400] Xueyang Feng, Zhi-Yuan Chen, Yujia Qin, Yankai Lin, Xu Chen, Zhiyuan Liu, and Ji-Rong Wen. Large language model-based human-agent collaboration for complex task solving. *arXiv preprint arXiv:2402.12914*, 2024.

[401] Rafael Mendoza, Isabella Cruz, Richard Liu, Aarav Deshmukh, David Williams, Jesscia Peng, and Rohan Iyer. Adaptive self-supervised learning strategies for dynamic on-device llm personalization. *arXiv preprint arXiv:2409.16973*, 2024.

[402] Logan Cross, Violet Xiang, Agam Bhatia, Daniel LK Yamins, and Nick Haber. Hypothetical minds: Scaffolding theory of mind for multi-agent tasks with large language models. *arXiv preprint arXiv:2407.07086*, 2024.

[403] Run Luo, Lu Wang, Wanwei He, Longze Chen, Jiaming Li, and Xiaobo Xia. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*, 2025.

[404] Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, et al. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*, 2025.

[405] Yuhang Liu, Zeyu Liu, Shuanghe Zhu, Pengxiang Li, Congkai Xie, Jiasheng Wang, Xueyu Hu, Xiaotian Han, Jianbo Yuan, Xinyao Wang, et al. Infigui-g1: Advancing gui grounding with adaptive exploration policy optimization. *arXiv preprint arXiv:2508.05731*, 2025.

[406] Yuqi Zhou, Sunhao Dai, Shuai Wang, Kaiwen Zhou, Qinglin Jia, and Jun Xu. Gui-g1: Understanding r1-zero-like training for visual grounding in gui agents. *arXiv preprint arXiv:2505.15810*, 2025.

[407] Yucheng Shi, Wenhao Yu, Zaitang Li, Yonglin Wang, Hongming Zhang, Ninghao Liu, Haitao Mi, and Dong Yu. Mobilegui-rl: Advancing mobile gui agent through reinforcement learning in online environment. *arXiv preprint arXiv:2507.05720*, 2025.

[408] Liujian Tang, Shaokang Dong, Yijia Huang, Minqi Xiang, Hongtao Ruan, Bin Wang, Shuo Li, Zhiheng Xi, Zhihui Cao, Hailiang Pang, et al. Magicgui: A foundational mobile gui agent with scalable data pipeline and reinforcement fine-tuning. *arXiv preprint arXiv:2508.03700*, 2025.

[409] Xinbin Yuan, Jian Zhang, Kaixin Li, Zhuoxuan Cai, Lujian Yao, Jie Chen, Enguang Wang, Qibin Hou, Jinwei Chen, Peng-Tao Jiang, et al. Enhancing visual grounding for gui agents via self-evolutionary reinforcement learning. *arXiv preprint arXiv:2505.12370*, 2025.