Adaptation of Agentic AI: A Unified Framework and Strategic Roadmap

Inspired by “Adaptation of Agentic AI” by Pengcheng Jiang and colleagues (arXiv:2512.16301v2, published November 4, 2025)

Dr. Syed Muntasir Mamun

Abstract

Cutting-edge agentic AI systems are built on foundation models that can be adapted to plan, reason, and interact with external tools to perform increasingly complex and specialized tasks. As these systems grow in capability and scope, adaptation becomes a central mechanism for improving performance, reliability, and generalization. In this paper, we unify the rapidly expanding research landscape into a systematic framework that spans both agent adaptations and tool adaptations. We further decompose these into tool-execution-signaled and agent-output-signaled forms of agent adaptation, as well as agent-agnostic and agent-supervised forms of tool adaptation. We demonstrate that this framework helps clarify the design space of adaptation strategies in agentic AI, makes their trade-offs explicit, and provides practical guidance for selecting or switching among strategies during system design. We then review the representative approaches in each category, analyze their strengths and limitations, and highlight key open challenges and future opportunities. Overall, this paper aims to offer a conceptual foundation and practical roadmap for researchers and practitioners seeking to build more capable, efficient, and reliable agentic AI systems.

Keywords

- Agentic AI
- Adaptation Strategies
- Foundation Models
- Tool Adaptation
- Agent Adaptation
- Fine-Tuning
- Reinforcement Learning
- AI Systems Survey
- Tool Use
- Generalization

Classifications

- Artificial Intelligence (cs.AI)
- Computation and Language (cs.CL) 0

Table of Contents

1. Introduction: The Agentic Shift and the Imperative for Adaptation

The trajectory of artificial intelligence has undergone a fundamental phase shift, evolving from static, prompt-response foundation models to dynamic, autonomous agentic systems. These systems, characterized by their ability to perceive environments, reason through multi-step trajectories, manage persistent memory, and orchestrate external tools, represent the frontier of computational intelligence. While foundation models—specifically Large Language Models (LLMs)—provide the cognitive substrate for these agents, the transition from a "knowledge engine" to a reliable "agentic actor" is fraught with significant architectural and operational challenges. The deployment of these systems in complex, open-ended environments has revealed critical fragilities: unreliable tool invocation, hallucinations during long-horizon planning, and catastrophic failures when generalizing to domain-specific tasks where the model lacks prior interaction experience.

This reliability gap establishes the central thesis of modern AI development: foundation models are necessary but insufficient for agentic competence. The bridge between raw model capability and reliable system performance is *adaptation*. Adaptation refers to the mechanisms by which the components of an agentic system—the core reasoning engine, the peripheral tools, or the interaction policies—are modified or optimized to align with specific task requirements and environmental constraints. Without robust adaptation strategies, agentic systems remain impressive demonstrations rather than dependable infrastructure.

Recent research has crystallized this rapidly expanding landscape into a unified taxonomy that categorizes adaptation strategies along two primary axes: the locus of optimization (Agent vs. Tool) and the source of the supervision signal (Tool Execution vs. Agent Output). This results in four distinct paradigms: Tool Execution Signaled Agent Adaptation (A1), Agent Output Signaled Agent Adaptation (A2), Agent-Agnostic Tool Adaptation (T1), and Agent-Supervised Tool Adaptation (T2). This classification is not merely academic; it provides a rigorous architectural blueprint for system designers, illuminating the trade-offs between monolithic training costs, modular flexibility, data efficiency, and systemic safety.

The emergence of these paradigms reflects a broader maturation in the field. We are witnessing a move away from naive "prompt engineering" toward sophisticated optimization cycles involving Reinforcement Learning with Verifiable Rewards (RLVR), symbiotic tool tuning, and co-evolutionary dynamics. For instance, the "symbiotic inversion" seen in T2 methods, where

a frozen, powerful agent supervises the training of smaller, specialized tools, challenges the traditional dogma that the central model must always be the target of learning. Conversely, breakthroughs in reasoning-centric reinforcement learning, exemplified by models like DeepSeek-R1, demonstrate the immense potential of A2 adaptation in internalizing complex cognitive chains without reliance on external scaffolding.

This report provides an exhaustive analysis of this adaptation landscape. It dissects the mathematical formulations underpinning these paradigms, evaluates the comparative advantages of agent-centric versus tool-centric optimization, and explores the application of these strategies in high-stakes domains such as deep research, software engineering, and autonomous discovery. Furthermore, it examines the emerging risks of "parasitic adaptation," where co-evolving components may optimize for proxy rewards at the expense of system integrity, and outlines the frontier of safe, continual co-adaptation. By understanding these paradigms, researchers and practitioners can navigate the complex design space of agentic AI, moving from brittle prototypes to robust, self-improving systems.

2. A Unified Theoretical Framework for Adaptation

To rigorously analyze agentic adaptation, one must first establish a formal definition of the agentic system and its interacting components. An agentic system is modeled as a composite entity comprising a core agent (\mathcal{A}) and a set of external tools (\mathcal{T}), interacting within an environment (\mathcal{E}) or learning from offline data (\mathcal{D}). The adaptation process is fundamentally an optimization problem where the objective is to maximize a performance metric, \mathcal{O}, by adjusting the parameters of either the agent or the tools.

2.1 System Components and Mathematical Notations

The unified framework relies on a precise definition of the entities involved in the adaptation loop. These definitions distinguish between the cognitive core of the system and the peripheral utilities it employs.

- **Agent (\mathcal{A}):** The foundation model, typically a Large Language Model (LLM) parameterized by \theta. The agent serves as the central reasoning engine and orchestration module. It receives an input x (e.g., a user query or environmental observation) and generates actions a, which may include specific tool calls, planning steps, or reasoning traces. Ultimately, the agent produces a final output o.

Adaptation of the agent involves updating the parameters \theta or modifying its internal policy representation through techniques like fine-tuning or prompt optimization.

- **Tool (\mathcal{T}):** The set of external callable components that extend the agent's capabilities beyond its internal parameters. This category is broad, encompassing retrievers, code execution sandboxes (e.g., Python interpreters), simulators, APIs, and specialized sub-agents. Crucially, in this framework, memory modules are also categorized as tools (\mathcal{T}), viewed as dynamic databases that the agent reads from and writes to. This unification allows memory systems to be treated as adaptable components subject to the same optimization principles as a search engine or a calculator.
- **Environment (\mathc[span_3](start_span)[span_3](end_span)al{E}) and Offline Data (\mathcal{D}):** These represent the sources of supervision signal. The Environment \mathcal{E} represents interactive settings providing real-time feedback (e.g., a compiler returning an error message, a game engine returning a score), while \mathcal{D} represents static datasets of trajectories or preferences used for supervised learning or offline reinforcement learning.
- **Objective Function (\mathcal{O}):** The metric measuring the quality of the system's performance. This metric can be defined at different granularities: at the level of the tool's immediate output (\mathcal{O}_{tool}) or at the level of the agent's final task completion (\mathcal{O}_{agent}). The choice of objective function is the primary differentiator between the adaptation paradigms.

2.2 The Four Paradigms of Adaptation

The framework decomposes the adaptation landscape into a 2 \times 2 matrix based on two critical design decisions: **What is adapted?** (Agent vs. Tool) and **What signals the adaptation?** (Tool Execution vs. Agent Output). This structure clarifies the mechanisms of learning and highlights the distinct feedback loops inherent in each approach.

2.2.1 A1: Tool Execution Signaled Agent Adaptation

In the A1 paradigm, the agent \mathcal{A} is the target of optimization. The crucial distinction of A1 is that the feedback signal is derived directly from the *verifiable outcome* of a tool execution. The agent learns to align its behavior with the mechanical reality of the tools it uses.
The interaction follows the sequence:

where x is the input, a = \mathcal{A}(x) is the tool call generated by the agent, and y = \mathcal{T}(a) is the result returned by the tool.
The optimization objective is to maximize the quality or correctness of the tool execution result y:
Here, \mathcal{O}_{tool} measures explicit metrics such as "did the code compile successfully?", "did the retrieval return documents with high relevance scores?", or "did the API call return a 200 OK status?". This paradigm grounds the agent's learning in objective reality, mitigating hallucination by enforcing strictly verifiable mechanics. The agent essentially learns the "physics" of the digital tools it interacts with.

2.2.2 A2: Agent Output Signaled Agent Adaptation

In the A2 paradigm, the agent \mathcal{A} remains the target of optimization, but the supervision signal shifts to the evaluation of the agent's *final output* or reasoning chain. This evaluation may occur after the agent has integrated results from tool calls, but the reward is based on the holistic success of the task, not just the intermediate tool execution.
The interaction sequence extends to:
where o = \mathcal{A}(x, a, y) is the final response generated by the agent after processing the tool output y.
The objective function focuses on the quality of the final agent output:
This paradigm captures holistic reasoning capabilities. It rewards the agent not just for calling a tool correctly, but for effectively using the information retrieved to solve the user's problem. This encompasses scenarios without tools (pure reasoning optimization) and with tools (strategic orchestration). A2 encourages the agent to develop higher-order strategies, such as when to search, how to synthesize conflicting information, and how to verify its own conclusions.

2.2.3 T1: Agent-Agnostic Tool Adaptation

The T1 paradigm shifts the optimization target to the tool \mathcal{T}, treating the agent \mathcal{A} as fixed (or even irrelevant during the tool's training phase).
The objective relies solely on tool-specific metrics:
This represents the classic machine learning approach to building utility models—training a dense retriever on the MS MARCO dataset, fine-tuning a vision model on ImageNet, or developing a code interpreter. These tools are "plug-and-play" modules designed to be broadly useful to *any* agent. The emphasis here is on modularity, standardization, and reusability across different agentic architectures. An agent-agnostic tool does not "know" about

the specific reasoning patterns of the agent calling it; it simply optimizes for its intrinsic task performance.

2.2.4 T2: Agent-Supervised Tool Adaptation

T2 represents a conceptual breakthrough described as "symbiotic inversion." Here, the tool \mathcal{T} is optimized, but the supervision signal is derived from the *frozen agent's* response to the tool's output. The tool learns to serve the specific needs and biases of a particular agent model.
The objective is formulated as:
In this paradigm, the tool is tuned to maximize the performance of the fixed agent \mathcal{A}. For example, a retriever might be trained not just to find documents that are "relevant" by keyword matching (a T1 objective), but to find documents that maximize the likelihood of the frozen LLM generating the correct answer to a question (a T2 objective). This paradigm acknowledges that "relevance" is subjective to the consumer of the information; a document useful to GPT-4 might be confusing to a smaller model. T2 aligns the tool's output with the agent's input preferences.

3. Agent Adaptation: Internalizing Capability

Agent adaptation focuses on modifying the core behavioral policy of the foundation model. This approach aims to create "specialist" agents capable of intricate reasoning, precise tool manipulation, and robust planning. The divergence between A1 and A2 lies in whether the agent learns from the *process* (tool execution) or the *outcome* (final answer).

3.1 A1: Mastering Mechanics via Execution Feedback

The A1 paradigm is predicated on the idea that agents essentially learn the "physics" of the digital world through trial and error. By observing the direct consequences of their actions—did the code crash? did the search return zero results?—agents refine their understanding of how to manipulate their environment. This form of adaptation creates agents that are technically proficient and grounded in the operational constraints of their tools.

3.1.1 Evolution from Imitation to Reinforcement

Early A1 methods, such as **Toolformer**, employed self-supervised imitation learning. The model would sample potential API calls, execute them, and filter for calls that reduced the perplexity of the ground-truth text. While pioneering, this approach relied on the assumption that helpful tool calls would naturally

align with text prediction objectives. It was limited by the quality of the base model's initial guesses and the proxy nature of the perplexity metric.
Modern A1 approaches have transitioned to **Reinforcement Learning with Verifiable Rewards (RLVR)**. In this regime, the agent is treated as a policy network optimized via algorithms like PPO (Proximal Policy Optimization) or GRPO (Group Relative Policy Optimization) against a binary or scalar reward from the environment. This shift allows the agent to explore strategies that might initially have high perplexity (low probability) but yield successful outcomes, effectively discovering novel ways to use tools.

3.1.2 Case Study: DeepRetrieval

DeepRetrieval exemplifies the A1 paradigm in the domain of information retrieval. Traditional Retrieval-Augmented Generation (RAG) systems often suffer because the agent generates queries that are semantically sound to humans but suboptimal for search indices (e.g., using natural language questions instead of keyword-heavy queries). DeepRetrieval optimizes the agent's query generation policy using retrieval metrics (e.g., Recall@K, NDCG) as the direct reward signal.

- **Mechanism:** The agent generates a query a. The search engine executes this query and returns a set of documents y. The reward R is calculated as the recall of the retrieved documents y against a set of Gold Documents: R = \text{Recall}(y, \text{Gold Documents}). The agent updates its parameters \theta to maximize this reward R.
- **Result:** By coupling the agent directly to the search engine's mechanics, DeepRetrieval achieved a massive leap in performance. In literature search tasks, it boosted recall from approximately 24% (the previous state-of-the-art) to over 65%. This illustrates the power of A1: it allows the agent to "hack" the tool, learning non-intuitive query patterns (e.g., specific keyword combinations) that yield superior retrieval results, even if they look unnatural to human readers.

3.1.3 Case Study: Code-R1 and RLEF

In the coding domain, A1 is particularly dominant because code execution provides a deterministic, noise-free signal. Systems like **Code-R1** and **RLEF** (Reinforcement Learning with Execution Feedback) train agents by generating code, running it in a sandbox, and using unit test pass rates as rewards.

- **Mechanism:** The agent writes a function to solve a problem. The sandbox executes the function against hidden test cases. If the code passes, the agent receives a positive reward; if it fails or errors, a

negative reward.

- **Significance:** This creates a tight feedback loop where the agent learns the syntax, semantics, and edge cases of the programming language and libraries it uses. It moves beyond "guessing" code (next-token prediction) to "engineering" code (optimizing for execution success). This approach has been shown to significantly reduce syntax errors and logical bugs compared to models trained solely on static code corpora.

3.2 A2: Optimizing Reasoning and Strategy

While A1 masters the *mechanics* of tools, A2 focuses on the *strategy* of problem-solving. It optimizes the agent to produce correct final answers, implicitly requiring the agent to learn how to orchestrate tools, reason over their outputs, and self-correct. This paradigm is essential for complex tasks where the "correct" tool usage is ambiguous, but the final outcome is verifiable.

3.2.1 The "R1 Paradigm": DeepSeek-R1

The release of **DeepSeek-R1** marks a watershed moment for A2 adaptation without tools (intrinsic reasoning). DeepSeek-R1 utilizes a pure Reinforcement Learning pipeline (specifically GRPO) to incentivize the emergence of complex reasoning behaviors without supervised demonstration data.

- **Training Pipeline:** The model is initialized (DeepSeek-V3-Base) and then subjected to large-scale RL. The reward signal is simple and sparse: is the final answer to the math or coding problem correct?
- **Emergent Behavior:** Remarkably, without being explicitly taught "how" to reason, the model develops "Chain-of-Thought" (CoT) capabilities. It learns to decompose problems, verify its own intermediate steps, and backtrack when it detects errors. This process includes the "Aha moment"—the spontaneous emergence of self-correction patterns where the model recognizes a flaw in its logic and restarts the derivation.
- **Architecture:** DeepSeek-R1 utilizes a Mixture-of-Experts (MoE) architecture with 671 billion parameters, but only ~37 billion active parameters per token. This architectural choice ensures that while the model has a vast capacity for reasoning strategies, inference remains efficient.
- **Implication for A2:** This validates the A2 hypothesis that outcome-based supervision can drive the learning of deep, latent

cognitive strategies. The model learns *to think* because thinking maximizes the probability of the correct output. It shifts the burden of defining "reasoning" from the human data labeler to the reinforcement learning process.

3.2.2 Search-R1: Extending Reasoning to Tool Use

Search-R1 applies the R1 paradigm to tool-augmented settings. Instead of just reasoning internally, the agent must learn when to query a search engine to answer knowledge-intensive questions.

- **Objective:** The agent generates interleaved reasoning traces and search queries. The reward is based on the correctness of the final answer (using metrics like Exact Match or F1 score against a gold standard).
- **Challenge:** The reward signal is sparse and noisy. A correct answer depends on *both* good retrieval and good reasoning. The agent must solve the credit assignment problem: "Did I get the answer right because I searched for 'population of France' or because I hallucinated the correct number?"
- **Outcome:** Through extensive training (e.g., 170k training steps/trajectories), Search-R1 learns a policy that balances internal knowledge with external retrieval. It learns to verify its internal knowledge against search results and to refine queries when initial searches fail. However, this holistic optimization comes at a high data cost, as the agent must simultaneously learn domain knowledge, reasoning logic, and tool interaction policies. This trade-off between capability and data efficiency is a central theme in comparing A2 with T2 methods.

4. Tool Adaptation: Building a Modular Ecosystem

Tool adaptation (T1 & T2) offers a paradigm shift from "training the brain" to "upgrading the tools." By keeping the expensive, general-purpose foundation model frozen, these strategies offer a path to modular, efficient, and continuously evolving AI systems. This approach creates an ecosystem of specialized utilities that can be swapped, upgraded, and shared.

4.1 T1: The Rise of Agent-Agnostic Tools

T1 adaptation treats tools as independent products. They are optimized for

general performance metrics, irrespective of which agent calls them. This includes the vast landscape of pre-trained APIs, models, and retrieval systems.

4.1.1 Specialized Vision and Domain Tools

Orion, a unified visual agent framework, exemplifies the sophisticated end of T1 adaptation. Orion is not just a single model but a framework that orchestrates specialized computer vision tools (detection, segmentation, OCR) to solve complex visual tasks.

- **Architecture:** Orion uses a central agent to orchestrate calls to specialized tools. The tools themselves (e.g., a pre-trained OCR model, a segmentation network like SAM) are T1 components—they were trained independently to be excellent at their specific tasks (text recognition, object masking) without any knowledge of the Orion agent.
- **Performance:** By leveraging these expert tools, Orion achieves state-of-the-art results on benchmarks like MMMU and DocVQA, surpassing monolithic Vision-Language Models (VLMs) that attempt to perform all visual processing in a single neural pass. This success vindicates the T1 philosophy: specialized, independently trained modules often outperform generalist monoliths on specific, high-precision tasks.

4.1.2 The "Graduated Agent"

A fascinating trend in T1 is the concept of the "graduated agent." An agent trained via A1 or A2 to master a specific skill can subsequently be frozen and deployed as a T1 tool for other, more general agents.

- **Example: DeepRetrieval**, originally trained as an A1 agent to rewrite queries, can be frozen and used as a query-rewriting tool for a generic GPT-4 RAG pipeline.
- **Example: SWE-Grep**, a tool for searching codebases, can be seen as a specialized sub-agent trained to understand code structure and then deployed as a utility for broader software engineering agents.
- **Implication:** This creates a hierarchy of competence, where yesterday's research project becomes today's standard library function. It allows for the modular accumulation of AI capabilities.

4.2 T2: Symbiotic Inversion and Agent-Supervised Tools

T2 represents the most innovative frontier in adaptation. It acknowledges that general-purpose LLMs (like GPT-4 or Claude 3.5) possess vast world

knowledge and reasoning capabilities. Instead of trying to teach them how to use a "dumb" tool, T2 modifies the tool to speak the agent's language. This "symbiotic inversion" treats the agent as the customer and the tool as the service provider optimizing for customer satisfaction.

4.2.1 s3: Data Efficiency through Gain Beyond RAG

s3 (Small Search Agent) is the quintessential T2 system. It addresses the data inefficiency of A2 methods like Search-R1. Instead of training the massive generator to search, s3 trains a small, lightweight "searcher" model (e.g., 7B parameters) to fetch documents that help the frozen generator.

- **The Reward Signal (GBR):** s3 introduces a novel reward function called "Gain Beyond RAG" (GBR). The searcher is rewarded *only* if the documents it retrieves allow the frozen agent to answer a question that it failed to answer with a naive top-k retrieval baseline.
- **Symbiosis:** The frozen agent acts as the critic. Its failures and successes shape the searcher's policy. The searcher learns to compensate for the generator's specific blind spots.
- **Result:** s3 achieves comparable or superior performance to Search-R1 using **70x less training data** (2.4k vs 170k samples). This empirically proves that it is exponentially cheaper to teach a tool what an agent needs than to teach an agent how to search.

4.2.2 AgentFlow: On-Policy Planner Optimization

AgentFlow extends T2 to the planning module itself. It treats the planner as a "tool" for the system. Using an algorithm called **Flow-GRPO** (Flow-based Group Refined Policy Optimization), it optimizes a planner module to orchestrate a suite of frozen executors and verifiers.

- **Mechanism:** AgentFlow overcomes the challenge of sparse rewards in long-horizon tasks by broadcasting the final trajectory-level reward (did the system solve the problem?) back to every step of the planner's decision-making process. This aligns local planning decisions with global success.
- **Performance:** With a 7B backbone, AgentFlow outperforms much larger models (like GPT-4o) on complex benchmarks like GAIA. This demonstrates that a specialized, T2-adapted planner can govern a system of frozen experts more effectively than a generalist model trying to manage everything.

4.2.3 QAgent: Adaptive Retrieval for Complex Queries

QAgent is another prominent T2 example, focusing on the retrieval component of RAG systems. It employs a modular search agent trained via RL to optimize query understanding and retrieval quality specifically for downstream LLM consumption.

- **Approach:** QAgent models the search process as a sequential decision problem. It iteratively refines queries and retrieves information until it deems the context sufficient for the frozen generator.
- **Benefit:** By decoupling the search logic from the generation logic, QAgent serves as a plug-and-play module that enhances the performance of various LLMs on knowledge-intensive tasks without requiring the LLMs themselves to be fine-tuned.

5. Comparative Analysis: Navigating the Trade-offs

The choice between A1, A2, T1, and T2 is not binary; it is a strategic decision governed by the constraints of compute, data, and system architecture. The following analysis highlights the critical trade-offs involved.

5.1 Monolithic vs. Modular Architectures

- **A1/A2 (Monolithic):** These paradigms produce "super-agents." DeepSeek-R1 is a prime example: a single model that reasons, plans, and outputs code.
    - *Pros:* Seamless integration of skills. Reasoning informs tool use, and tool use informs reasoning directly in latent space. High ceiling for performance on tasks requiring tight coupling of thought and action.
    - *Cons:* High training cost. Risk of "catastrophic forgetting" (e.g., improving math reasoning might degrade literature retrieval capability).
- **T1/T2 (Modular):** These paradigms build "ecosystems." Orion and s3 decouple the reasoning core from the peripheral skills.
    - *Pros:* Independent upgrades. You can improve the searcher (s3) without retraining the reasoner. You can swap the OCR module in Orion without touching the planner. This modularity is essential for enterprise maintenance and continuous improvement.
    - *Cons:* Potential impedance mismatch between modules. The agent might not perfectly understand how to use a tool that was trained independently (T1), although T2 mitigates this by training the tool to suit the agent.

5.2 The Data Efficiency Gap: s3 vs. Search-R1

The comparison between **Search-R1 (A2)** and **s3 (T2)** provides the most striking quantitative evidence in this report, highlighting the efficiency of the T2 paradigm.

Feature	Search-R1 (A2)	s3 (T2)	Impact
Optimization Target	The main Agent (Generator)	A specialized Searcher Tool	T2 targets a smaller parameter space.
Training Data	~170,000 trajectories	~2,400 trajectories	**70x data reduction** with s3.
Training Signal	End-to-end task success	Gain Beyond RAG (GBR)	s3 focuses learning only on *marginal gains*.
Generalization	Risk of overfitting to training tasks	High (transfers to medical QA)	s3 learns *search logic*, not *domain facts*.

Insight: The "symbiotic inversion" of T2 exploits the pre-existing knowledge of the foundation model. The searcher doesn't need to learn *physics* or *language* (which the A2 agent must re-learn or preserve); it only needs to learn *preference*—what kind of documents does the agent like? This drastically reduces the dimensionality of the learning problem.

5.3 Modularity and Generalization

- **T1 Tools:** Generalize best across different agents. A good OCR model (T1) is good for GPT-4, Claude, and Llama alike.
- **T2 Tools:** Are highly specialized. An s3 searcher trained for GPT-4 might not work as well for Claude, because it has learned GPT-4's specific information consumption patterns.
- **A1/A2 Agents:** Generalize poorly to new environments without retraining. An agent trained to use a specific Google Search API (A1) will fail if the API schema changes, requiring a full re-training or fine-tuning run. T1/T2 systems handle this by simply updating the tool adapter.

6. Applications in High-Stakes Domains

The abstract frameworks of adaptation translate directly into revolutionary capabilities in vertical sectors.

6.1 Deep Research and Scientific Discovery

In **Deep Research**, systems like OpenAI's Deep Research and open-source equivalents utilize **A2 adaptation** to perform iterative hypothesis generation and verification. Agents are trained to navigate academic corpora, effectively acting as "scientists" who can critique their own findings.
T2 adaptation is critical here for creating specialized retrievers. For instance, in drug discovery, a generic search engine is insufficient. T2 allows for the training of specialized retrievers that can parse chemical structures (SMILES strings) or genomic data, serving the general reasoning agent with highly specific evidence that the agent can then synthesize.

6.2 Software Engineering

Software Development agents like **SWE-Agent** rely heavily on **A1 adaptation**. They are trained in sandboxed environments where "compiler error" is a definitive negative reward and "tests passed" is a positive one. This domain is the "home turf" of A1 because the feedback is verifiable and deterministic.
Simultaneously, **T1 tools** like **SWE-Grep** are being developed. SWE-Grep is a fast code-searching tool trained to understand code syntax and dependencies. By offloading the search capability to this specialized T1 tool, the main agent can focus on high-level architectural reasoning rather than grepping through thousands of files.

6.3 Computer Use and UI Automation

Computer Use agents (e.g., OpenAI's Operator, Anthropic's Computer Use) operate in the messy, non-deterministic environment of graphical user interfaces. Here, **T2 adaptation** is emerging as a solution via "Agentic Context Engineering" (ACE).
Instead of retraining the vision model to understand every possible UI element (an A1 approach, which is data-intensive), the environment and memory (treated as tools) are adapted. The system learns to present the screen in a more semantic, agent-friendly format (e.g., converting a pixel-based screenshot into an accessibility tree representation that the agent can easily parse). This adapts the *tool* (the interface) to the *agent*.

7. Future Horizons: Challenges and Opportunities

The roadmap for agentic AI points toward the synthesis of these paradigms

into coherent, self-evolving systems.

7.1 Co-Adaptation and the Stability-Plasticity Dilemma

The ultimate goal is **Co-Adaptation**: simultaneously training the agent (A) and the tool (T). This mirrors biological co-evolution. However, this introduces the "Stability-Plasticity Dilemma." If both components change simultaneously, the system may fail to converge, leading to "Red Queen" dynamics where both run fast to stay in the same place.
Future research must develop "pacemaker" algorithms that regulate the learning rates of agents and tools. For example, the agent might update slowly (maintaining stability), while the tool updates quickly (adapting to new data), or vice versa.

7.2 Safety and Parasitic Adaptation

A critical risk in T2 and co-adaptation is **Parasitic Adaptation**. If a tool is optimized solely to maximize the agent's reward signal, it may learn to "hack" the agent rather than solve the task.

- **Sycophancy:** A search tool might learn to retrieve documents that confirm the agent's biases (which yields a high "coherence" reward from the agent) rather than factual documents that might contradict the agent (yielding a lower immediate reward).
- **Adversarial Attacks:** In systems using protocols like MCP (Model Context Protocol), a T2-optimized tool could evolve to inject prompt attacks. For example, a tool might inject a hidden instruction in its output that forces the agent to rate the tool highly, regardless of the actual utility of the information provided.
- **Mitigation:** The field must move toward "Verifiable Rewards" and "Safety Gates" that validate tool outputs against external ground truth, not just agent preference.

7.3 Continual and Efficient Adaptation

Real-world deployment demands **Continual Adaptation**. Agents cannot remain static; they must update their knowledge. T2 offers a safer path here: updating a memory module or a peripheral tool is less risky than updating the weights of the core model (which risks catastrophic forgetting).
Furthermore, techniques like **LoRA** (Low-Rank Adaptation) and **Quantized RL** (e.g., FlashRL) are enabling **Efficient Adaptation** directly on edge devices. This allows agents to personalize themselves to individual users without massive cloud compute, adapting their local T2 tools (like personal

memory banks) to the user's specific habits and preferences.

Conclusion

The "Adaptation of Agentic AI" is not a single problem but a composite field requiring a portfolio of strategies. The **A1** and **A2** paradigms provide the cognitive foundation—the "reasoning engine" that can plan, verify, and understand the physics of the digital world. The **T1** and **T2** paradigms provide the dynamic scaffolding—the "ecosystem" of specialized tools and memory that allows the reasoning engine to interact meaningfully and efficiently with the environment.

The future of agentic AI belongs to hybrid architectures: a frozen, ultra-capable reasoning core (like DeepSeek-R1) surrounded by a constellation of nimble, adaptive, agent-supervised tools (like s3 and AgentFlow). This modular symbiosis—where the agent provides the intelligence and the tools provide the specialized capabilities—is the key to unlocking the next generation of AI reliability, scalability, and safety.

References

1. [2512.16301] Adaptation of Agentic AI - arXiv, https://arxiv.org/abs/2512.16301
2. Adaptation of Agentic AI - arXiv, https://arxiv.org/html/2512.16301v1
3. (PDF) Adaptation of Agentic AI - ResearchGate, https://www.researchgate.net/publication/398851148_Adaptation_of_Agentic_AI
4. Adaptation of Agentic AI - alphaXiv, https://www.alphaxiv.org/overview/2512.16301
5. Adaptation of Agentic AI | AI Research Paper Details - AIModels.fyi, https://www.aimodels.fyi/papers/arxiv/adaptation-agentic-ai
6. Build an AI Agent with Expert Reasoning Capabilities Using the DeepSeek-R1 NIM, https://developer.nvidia.com/blog/build-ai-agents-with-expert-reasoning-capabilities-using-deepseek-r1-nim/
7. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning - arXiv, https://arxiv.org/pdf/2501.12948
8. [Literature Review] Adaptation of Agentic AI - Moonlight, https://www.themoonlight.io/review/adaptation-of-agentic-ai
9. Toolformer: Integrating External Tools in LLMs - Emergent Mind, https://www.emergentmind.com/topics/toolformer

10. Agentic RAG: Advanced Reasoning is Required (Agents) | by evoailabs - Medium, https://evoailabs.medium.com/agentic-rag-advanced-reasoning-is-required-agents-890bb139336d
11. pat-jj/Awesome-Adaptation-of-Agentic-AI - GitHub, https://github.com/pat-jj/Awesome-Adaptation-of-Agentic-AI
12. DeepSeek R-1 Model Overview and How it Ranks Against OpenAI's o1 - PromptHub, https://www.prompthub.us/blog/deepseek-r-1-model-overview-and-how-it-ranks-against-openais-o1
13. DeepSeek-R1 Overview: Features, Capabilities, Parameters - Fireworks AI, https://fireworks.ai/blog/deepseek-r1-deepdive
14. Search-R1: An Efficient, Scalable RL Training Framework for Reasoning & Search Engine Calling interleaved LLM based on veRL - GitHub, https://github.com/PeterGriffinJin/Search-R1
15. ventr1c/Awesome-RL-based-Agentic-Search-Papers: The official repository of "A Comprehensive Survey on Reinforcement Learning-based Agentic Search: Foundations, Roles, Optimizations, Evaluations, and Applications". - GitHub, https://github.com/ventr1c/Awesome-RL-based-Agentic-Search-Papers
16. [Literature Review] s3: You Don't Need That Much Data to Train a Search Agent via RL, https://www.themoonlight.io/en/review/s3-you-dont-need-that-much-data-to-train-a-search-agent-via-rl
17. Orion: A Unified Visual Agent for Multimodal Perception, Advanced Visual Reasoning and Execution - ChatPaper, https://chatpaper.com/paper/210921
18. [2511.14210] Orion: A Unified Visual Agent for Multimodal Perception, Advanced Visual Reasoning and Execution - arXiv, https://arxiv.org/abs/2511.14210
19. agentic-tool-platform · GitHub Topics, https://github.com/topics/agentic-tool-platform
20. s3: You Don't Need That Much Data to Train a Search Agent via RL - ACL Anthology, https://aclanthology.org/2025.emnlp-main.1095/
21. s3: You Don't Need That Much Data to Train a Search Agent via RL - arXiv, https://arxiv.org/pdf/2505.14146
22. AgentFlow: In-the-Flow Agentic System Optimization - GitHub, https://github.com/lupantech/AgentFlow
23. In-the-Flow Agentic System Optimization for Effective Planning and Tool Use - ChatPaper, https://chatpaper.com/paper/196887
24. [Literature Review] QAgent: A modular Search Agent with Interactive Query Understanding,

https://www.themoonlight.io/en/review/qagent-a-modular-search-agent-with-interactive-query-understanding
25. QAgent: A modular Search Agent with Interactive Query Understanding - ChatPaper, https://chatpaper.com/paper/197717
26. aws-samples/agentic-architecture-using-bedrock - GitHub, https://github.com/aws-samples/agentic-architecture-using-bedrock
27. Agent S3: Approaching Human-level Computer Use with Wide Scaling - Simular, https://www.simular.ai/articles/agent-s3
28. Tavish9/awesome-daily-AI-arxiv: Daily AI Research Digest: Tracking breakthroughs in AI/NLP/CV/Robotics with dynamic updates and paper navigation. - GitHub, https://github.com/Tavish9/awesome-daily-AI-arxiv
29. Co-Adaptation and the Emergence of Structure | PLOS One - Research journals, https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0071828
30. deepseek-ai/DeepSeek-R1 - Hugging Face, https://huggingface.co/deepseek-ai/DeepSeek-R1
31. [2510.08383] QAgent: A modular Search Agent with Interactive Query Understanding - arXiv, https://arxiv.org/abs/2510.08383