The Landscape of Agentic Reinforcement Learning for LLMs: A Survey

Guibin Zhang[ϵ†] Hejia Geng[α†] Xiaohang Yu[η†] Zhenfei Yin[α✉]

Zaibin Zhang[ν,α] Zelin Tan[ζ,β] Heng Zhou[ζ,β] Zhongzhi Li[ι] Xiangyuan Xue[κ,β]

Yijiang Li[ξ] Yifan Zhou[μ] Yang Chen[β] Chen Zhang[ζ] Yutao Fan[β] Zihu Wang[χ]

Songtao Huang[λ,β] Piedrahita-Velez, Francisco[ε] Yue Liao[ϵ] Hongru Wang[κ] Mengyue Yang[θ]

Heng Ji[δ] Jun Wang[γ] Shuicheng Yan[ϵ] Philip Torr[α] Lei Bai[β✉]

[α]University of Oxford [β]Shanghai AI Laboratory [ϵ]National University of Singapore

[γ]University College London [δ]University of Illinois Urbana-Champaign [ε]Brown University

[ζ]University of Science and Technology of China [η]Imperial College London [θ]University of Bristol

[ι]Chinese Academy of Sciences [κ]The Chinese University of Hong Kong [λ]Fudan University

[μ]University of Georgia [ξ]University of California, San Diego [ν]Dalian University of Technology

[χ]University of California, Santa Barbara

[†] *Equal contribution*, [✉] *Corresponding Author*

Abstract: The emergence of agentic reinforcement learning (Agentic RL) marks a paradigm shift from conventional reinforcement learning applied to large language models (LLM RL), reframing LLMs from passive sequence generators into autonomous, decision-making agents embedded in complex, dynamic worlds. This survey formalizes this conceptual shift by contrasting the degenerate single-step Markov Decision Processes (MDPs) of LLM-RL with the partially observable, temporally extended partially observable Markov decision process (POMDP) that define Agentic RL. Building on this foundation, we propose a comprehensive twofold taxonomy: one organized around core agentic capabilities, including planning, tool use, memory, reasoning, self-improvement, and perception, and the other around their applications across diverse task domains. Central to our thesis is that reinforcement learning serves as the critical mechanism for transforming these capabilities from static, heuristic modules into adaptive, robust agentic behavior. To support and accelerate future research, we consolidate the landscape of open-source environments, benchmarks, and frameworks into a practical compendium. By synthesizing over five hundred recent works, this survey charts the contours of this rapidly evolving field and highlights the opportunities and challenges that will shape the development of scalable, general-purpose AI agents.

Keywords: Agentic Reinforcement Learning, Large Language Models, LLM Agent

Date: November 1st, 2025
Code Repository: https://github.com/xhyumiracle/Awesome-AgenticLLM-RL-Papers
Corresponding: jeremyyin@robots.ox.ac.uk, bailei@pjlab.org.cn
Main Contact: guibinz@u.nus.edu, genghejia0530@gmail.com, x.yu21@imperial.ac.uk

Contents

1. Introduction

The rapid convergence of large language models (LLMs) and reinforcement learning (RL) has precipitated a fundamental transformation in how language models are conceived, trained, and deployed. Early LLM-RL paradigms largely treated these models as static conditional generators, optimized to produce single-turn outputs aligned with human preferences or benchmark scores. While successful for alignment and instruction following, such approaches overlook the broader spectrum of sequential decision-making that underpins realistic, interactive settings. These limitations have prompted a shift in perspective: rather than viewing LLMs as passive text emitters, recent developments increasingly frame them as *Agents*, *i.e.*, autonomous decision-makers capable of perceiving, reasoning, planning, invoking tools, maintaining memory, and adapting strategies over extended horizons in partially observable, dynamic environments. We define this emerging paradigm as **Agentic Reinforcement Learning (Agentic RL)**. To more clearly delineate the distinction between the concept of Agentic RL studied in this work and conventional RL approaches, we provide the following definition:

> **Agentic Reinforcement Learning (Agentic RL)** refers to a paradigm in which LLMs, rather than being treated as *static conditional generators* optimized for single-turn output alignment or benchmark performance, are conceptualized as *learnable policies* embedded within sequential decision-making loops, where RL endows them with autonomous agentic capabilities, such as planning, reasoning, tool use, memory maintenance, and self-reflection, enabling the emergence of long-horizon cognitive and interactive behaviors in *partially observable, dynamic environments*.

In Section 2, we present a more formal, symbolically grounded distinction between agentic RL and conventional RL. Prior research relevant to agentic RL can be broadly grouped into two complementary threads: **Synergy between RL and LLMs** and **LLM Agents**, detailed as follows:

Synergy between RL and LLMs. The second line of research investigates how reinforcement learning algorithms are applied to improve or align LLMs. A primary branch, RL for training LLMs, leverages on-policy (*e.g.*, proximal policy optimization (PPO) [1] and Group Relative Policy Optimization (GRPO) [2]) and off-policy (*e.g.*, actor–critic, Q-learning [3]) methods to enhance capabilities such as instruction following, ethical alignment, and code generation [4, 5, 6]. A complementary direction, LLMs for RL, examines the deployment of LLMs as planners, reward designers, goal generators, or information processors to improve sample efficiency, generalization, and multi-task planning in control environments, with systematic taxonomies provided by Cao et al. [7]. RL has also been integrated throughout the LLM lifecycle: from data generation [8, 9] and pretraining [10] to post-training and inference [11], as surveyed by Guo et al. [12]. The most prominent branch here is post-training alignment, notably Reinforcement Learning from Human Feedback (RLHF) [13], along with extensions such as Reinforcement Learning from AI Feedback (RLAIF) and Direct Preference Optimization (DPO) [14, 15, 16, 4].

LLM Agents. LLM-based agents represent an emerging paradigm in which LLMs act as autonomous or semi-autonomous decision-making entities [17, 18], capable of reasoning, planning, and executing actions in pursuit of complex goals. Recent surveys have sought to map this landscape from complementary perspectives. Luo et al. [19] propose a methodology-centered taxonomy that connects architectural foundations, collaboration mechanisms, and evolutionary pathways, while Plaat et al. [20] emphasize the core capabilities of reasoning, acting, and interacting as defining features of *agentic* LLMs. Tool use, encompassing retrieval-augmented generation (RAG) and API utilization, is a central paradigm, extensively discussed in Li et al. [21] and further conceptualized by Wang et al. [22]. Planning and reasoning strategies form another pillar, with

surveys such as Masterman et al. [23] and Kumar et al. [24] highlighting common design patterns like plan-execute-reflect loops, while Tao et al. [25] extend this to self-evolution, where agents iteratively refine knowledge and strategies without substantial human intervention. Other directions explore collaborative, cross-modal, and embodied settings, from multi-agent systems [26] to multimodal integration [27], and brain-inspired architectures with memory and perception [28].

Research Gap and Our Contributions. The recent surge in research on LLM agents and RL-enhanced LLMs reflects two complementary perspectives: one explores what large language models can do as the core of autonomous agents, while the other focuses on how reinforcement learning can optimize their behavior. However, despite the breadth of existing work, a unified treatment of *agentic RL*, which conceptualizes LLMs as policy-optimized agents embedded in sequential decision processes, remains lacking. Current studies often examine isolated capabilities, domains, or custom environments, with inconsistent terminology and evaluation protocols, making systematic comparison and cross-domain generalization difficult. To bridge this gap, we present a coherent synthesis that connects theoretical foundations with algorithmic approaches and practical systems. We formalize agentic RL through Markov decision process (MDP) and partially observable Markov decision process (POMDP) abstractions to distinguish it from classical LLM-RL paradigms, and introduce a capability-centered taxonomy that includes planning, tool use, memory, reasoning, reflection (self-improvement), and interaction as RL-optimizable components. Furthermore, we consolidate representative tasks, environments, frameworks, and benchmarks that support agentic LLM training and evaluation, and conclude by discussing open challenges and outlining promising future directions for scalable, general-purpose agentic intelligence. Overall, we aim to further clarify the research scope of this survey:

Primary focus:

✔ how RL empowers LLM-based agents (or, LLMs with agentic characteristics) in *dynamic environments*

Out of scope (though occasionally mentioned):

✗ RL for human value alignment (*e.g.*, RL for harmful query refusal);
✗ traditional RL algorithms that are not LLM-based (*e.g.*, MARL [29]);
✗ RL for boosting pure LLM performance on static benchmarks.

Structure of the Survey. This survey is organized to progressively build a unified understanding of Agentic RL from conceptual foundations to practical implementations. Section 2 formalizes the paradigm shift to Agentic RL through an MDP/POMDP lens. Section 3 examines agentic RL from the capability perspective, categorizing key modules such as planning, reasoning, tool using, memory, self-improvement, perception, and others. Section 4 explores applications across domains, including search, GUI navigation, code generation, mathematical reasoning, and multi-agent systems. Section 5 consolidates open-source environments and RL frameworks that underpin experimentation and benchmarking. Section 6 discusses open challenges and future directions towards scalable, adaptive, and reliable agentic intelligence, and Section 7 concludes the survey. The overall structure is also illustrated in Figure 1.

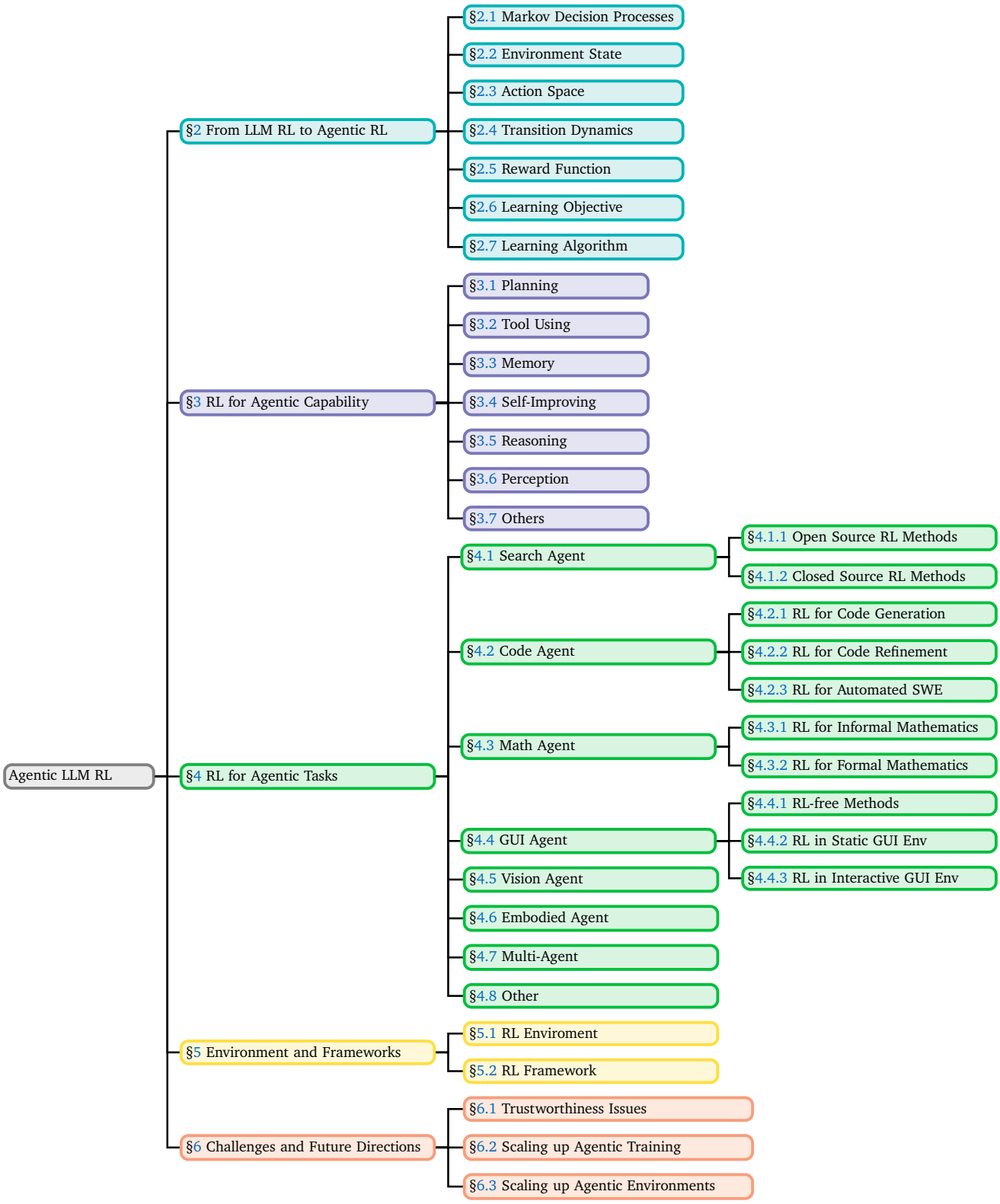

Figure 1: The primary organizational structure of the survey.

2. Preliminary: From LLM RL to Agentic RL

LLMs are initially pre-trained using behavior cloning, which applies maximum likelihood estimation (MLE) to static datasets such as web-scraped text corpora. Subsequent post-training methods enhance capabilities and align outputs with human preferences—transforming them beyond generic web-data replicators. A common technique is supervised fine-tuning (SFT), where models are refined on human-generated (prompt, response) demonstrations. However, procuring sufficient high-quality SFT data remains challenging. Reinforcement fine-tuning (RFT) offers an alternative by optimizing models through reward functions, circumventing dependence on behavioral demonstrations.

In early RFT research, the core objective is to optimize LLMs through human feedback [13] or data preferences [30], aligning them with human preferences or directly with data preferences (as in DPO). This **preference-based RFT (PBRFT)** primarily involves learning reward model optimization for LLMs on a fixed preference dataset, or directly implementing it using data preferences. With the release of LLMs such as OpenAI o1 [31] and DeepSeek-R1 [32] that possess reasoning capabilities, their improved performance and cross-domain generalization have garnered widespread attention. With the release of models like OpenAI o3 [33], which possess both self-evolving reasoning capabilities and support for tool use, researchers are beginning to contemplate how to deeply integrate LLMs with downstream tasks through reinforcement learning methods. Subsequently, researchers have shifted their focus from PBRFT, aimed at optimizing fixed preference datasets, to agentic reinforcement learning tailored for specific tasks and dynamic environments.

In this section, we provide a formalization of the paradigm shift from PBRFT to the emerging framework of **agentic reinforcement learning (Agentic RL)**. While both approaches leverage RL techniques to improve LLMs' performance, they fundamentally differ in their underlying assumptions, task structure, and decision-making granularity. Figure 2 illustrates the paradigm shift from LLM RL to agentic RL.

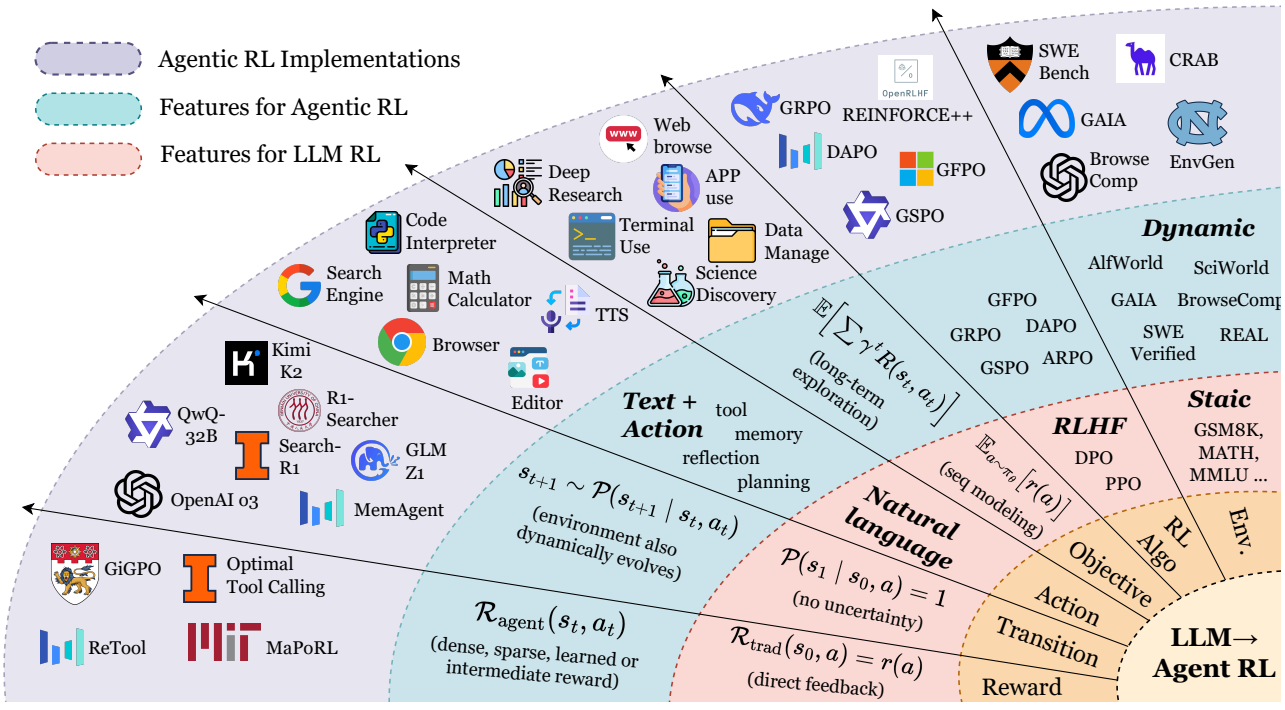

Figure 2: Paradigm shift from LLM-RL to agentic RL. We draw inspiration from the layout of Figure 1 in [24].

2.1. Markov Decision Processes

The Markov decision process (MDP) for the RL fine-tuning process can be formalized as a seven-element tuple $\langle \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{P}, \mathcal{R}, T, \gamma \rangle$, where $\mathcal{S}$ represents the state space and $\mathcal{O}$ is the observation space of the agent. $\mathcal{A}$ denotes the action space. $\mathcal{R}$ is defined as the reward function, $\mathcal{P}$ encapsulates the state transition probabilities, T signifies the task horizon, and γ is the discount factor. By casting both preference-based RFT and agentic RL as MDP or POMDP, we clarify the theoretical implications of treating LLMs either as static sequence generators or as interactive, decision-capable agents embedded within dynamic environments.

PBRFT. The RL training process of PBRFT is formalized as a degenerate MDP defined by the tuple:

$$\langle \mathcal{S}_{\text{trad}}, \mathcal{A}_{\text{trad}}, \mathcal{P}_{\text{trad}}, \mathcal{R}_{\text{trad}}, \gamma = 1, T = 1 \rangle. \tag{1}$$

Agentic RL. The RL training process of agentic RL is modeled as a POMDP:

$$\langle \mathcal{S}_{\text{agent}}, \mathcal{A}_{\text{agent}}, \mathcal{P}_{\text{agent}}, \mathcal{R}_{\text{agent}}, \gamma, \mathcal{O} \rangle, \tag{2}$$

where the agent receives observations $o_t = O(s_t)$ based on the state $s_t \in \mathcal{S}_{\text{agent}}$. The primary distinctions between PBRFT and agentic RL are delineated in Table 1. In summary, PBRFT optimizes sequences of output sentences within a fixed dataset under full observations, whereas agentic RL optimizes semantic-level behaviors in variable environments characterized by partial observations.

Table 1: Formal comparison between traditional PBRFT and Agentic RL.

Concept	**Traditional PBRFT**	**Agentic RL**
$\mathcal{S}$: State space	$\{s_0\}$ (single prompt); episode ends immediately.	$s_t \in \mathcal{S}_{\text{agent}}$; $o_t = O(s_t)$; horizon $T > 1$.
$\mathcal{A}$: Action space	Pure text sequence.	$\mathcal{A}_{\text{text}} \cup \mathcal{A}_{\text{action}}$.
$\mathcal{P}$: Transition	Deterministic to the terminal state.	Dynamic transition function $P(s_{t+1} \mid s_t, a_t)$.
$\mathcal{R}$: Reward	Single scalar $r(a)$.	Step-wise $R(s_t, a_t)$; combines sparse task and dense sub-rewards.
$J(\theta)$: Objective	$\mathbb{E}_{a \sim \pi_\theta}[r(a)]$.	$\mathbb{E}_{\tau \sim \pi_\theta}[\sum_t \gamma^t R(s_t, a_t)]$.

2.2. Environment State

PBRFT. In the training process, each episode starts from a single prompt state s_0; the episode terminates immediately after the model emits one response. Formally, the underlying MDP degenerates to a *single-step* decision problem with horizon $T = 1$. The state space reduces to a single static prompt input:

$$\mathcal{S}_{\text{trad}} = \{\text{prompt}\}. \tag{3}$$

Agentic RL. The LLM agent acts over multiple time-steps in a POMDP. Let $s_t \in \mathcal{S}_{\text{agent}}$ denote the full world-state and the LLM agent gets observation O_t based on current state $o_t = \mathcal{O}(s_t)$. The LLM agent chooses an action a_t based on the current observation o_t, and the state evolves over time:

$$s_{t+1} \sim P(s_{t+1} \mid s_t, a_t), \tag{4}$$

as the agent accumulates intermediate signals such as retrieved tool results, user messages, or environment feedback. The interaction is thus inherently dynamic and temporally extended.

2.3. Action Space

In the agentic RL setting, the LLM's action space comprises two distinct subspaces:

$$\mathcal{A}_{\text{agent}} = \mathcal{A}_{\text{text}} \cup \mathcal{A}_{\text{action}} \tag{5}$$

Here, $\mathcal{A}_{\text{text}}$ denotes the space of free-form natural language tokens emitted via autoregressive decoding, while $\mathcal{A}_{\text{action}}$ denotes the space of abstract, non-linguistic actions (e.g., delimited in the output stream by special tokens `<action_start>` and `<action_end>`). These actions may invoke external tools (e.g., `call("search", "Einstein")`) or interact with an environment (e.g., `move("north")`), depending on task requirements.

Notably, $\mathcal{A}_{\text{action}}$ is recursively constructed, such that an element $a \in \mathcal{A}_{\text{action}}$ may itself represent a sequence $(a_1, \ldots, a_k)$ of primitive actions, thus unifying primitive and composite actions within the same space.

Formally, the two subspaces differ in semantics and functional role: $\mathcal{A}_{\text{text}}$ defines the space of outputs intended for human or machine interpretation without directly altering the external state, whereas $\mathcal{A}_{\text{action}}$ defines the space of environment-interactive behaviors that either (i) acquire new information through tool invocations, or (ii) modify the state of a physical or simulated environment. This distinction enables a unified policy jointly model language generation and environment interaction within the same RL formulation.

2.4. Transition Dynamics

PBRFT. In conventional PBRFT, the transition dynamics are deterministic: the next state is determined once an action is made, as follows:

$$\mathcal{P}(s_1 \mid s_0, a) = 1, \quad \text{where there is no uncertainty.} \tag{6}$$

Agentic RL. In agentic RL, the environment evolves under uncertainty according to

$$s_{t+1} \sim \mathcal{P}(s_{t+1} \mid s_t, a_t), \quad a_t \in \mathcal{A}_{\text{text}} \cup \mathcal{A}_{\text{action}}. \tag{7}$$

Text actions $(\mathcal{A}_{\text{text}})$ generate natural language outputs without altering the environment state. Structured actions $(\mathcal{A}_{\text{action}})$, delimited by `<action_start>` and `<action_end>`, can either query external tools or directly modify the environment. This sequential formulation contrasts with the one-shot mapping of PBRFT, enabling policies that iteratively combine communication, information acquisition, and environment manipulation.

2.5. Reward Function

PBRFT. PBRFT commonly features a reward function with verifiable response correctness, which may be implemented using either a rule-based verifier [32] or a neural network-parameterized reward model [34]. Regardless of the implementation approach, its core follows the equation:

$$\mathcal{R}_{\text{trad}}(s_0, a) = r(a), \tag{8}$$

where $r : \mathcal{A} \to \mathbb{R}$ is a scalar score supplied by a human- or AI-preference model; with no intermediate feedback.

Agentic RL. The reward function of the LLM agent is based on the downstream task.

$$\mathcal{R}_{\text{agent}}(s_t, a_t) = \begin{cases} r_{\text{task}} & \text{on task completion,} \\ r_{\text{sub}}(s_t, a_t) & \text{for step-level progress,} \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

allowing dense, sparse, or learned rewards (*e.g.*, unit-test pass, symbolic verifier success).

2.6. Learning Objective

PBRFT. The optimization objective of PBRFT is to maximize the response reward based on the policy π_θ:

$$J_{\text{trad}}(\theta) = \mathbb{E}_{a\sim\pi_\theta}\big[r(a)\big]. \tag{10}$$

No discount factor is required; optimization resembles maximum-expected-reward sequence modeling.

Agentic RL. The optimization objective of Agentic RL is to maximize the discounted reward:

$$J_{\text{agent}}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\gamma^t R_{\text{agent}}(s_t, a_t)\right], \qquad 0 < \gamma < 1, \tag{11}$$

optimized via policy-gradient or value-based methods with exploration and long-term credit assignment.

PBRFT focuses on single-turn text quality alignment without explicit planning, tool use, or environment feedback, while agentic RL involves multi-turn planning, adaptive tool invocation, stateful memory, and long-horizon credit assignment, enabling the LLM to function as an autonomous decision-making agent.

2.7. RL Algorithms

In contemporary research, RL algorithms constitute a pivotal component in both PBRFT and agentic RL frameworks. Different RL algorithms demonstrate distinct sample efficiency and performance characteristics, each offering a unique approach to the central challenge of aligning model outputs with complex, often subjective, human goals. The canonical methods, such as REINFORCE, PPO [1], GRPO [32], and DPO [30], form a spectrum from general policy gradients to specialized preference learning. We next introduce each of these three classic algorithms and provide a comparison of popular variants from each family in Table 2.

REINFORCE: The Foundational Policy Gradient As one of the earliest policy gradient algorithms, REINFORCE provides the foundational theory for training stochastic policies. It operates by increasing the probability of actions that lead to high cumulative reward and decreasing the probability of those that lead to low reward. Its objective function is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_0}\left[\frac{1}{N}\sum_{i=1}^{N}\Big(\mathcal{R}(s_0, a^{(i)}) - b(s_0)\Big)\nabla_\theta \log \pi_\theta(a^{(i)}|s_0)\right], \tag{12}$$

where $a^{(i)} \sim \pi_\theta(a|s_0)$ is the i-th sampled response, $\mathcal{R}(s_0, a)$ denotes the final rewards received on task completion, and $b(s)$ is a baseline function to reduce the variance of the policy gradient estimate. In general, $b(s)$ can be any function, including random variables.

Proximal Policy Optimization (PPO) PPO [1] became the dominant RL algorithm for LLM alignment due to its stability and reliability. It improves upon vanilla policy gradients by limiting the update step to prevent destructively large policy changes. Its primary clipped objective function is:

$$L_{PPO}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \min\left(\frac{\pi_\theta(a_t^{(i)}|s_t)}{\pi_{\theta_{old}}(a_t^{(i)}|s_t)} A(s_t, a_t^{(i)}),\ \text{clip}\left(\frac{\pi_\theta(a_t^{(i)}|s_t)}{\pi_{\theta_{old}}(a_t^{(i)}|s_t)}, 1-\epsilon, 1+\epsilon\right) A(s_t, a_t^{(i)})\right), \tag{13}$$

where $a_t^{(i)} \sim \pi_{\theta_{old}}(a|s_t)$ is the i-th sampled response from the old policy $\pi_{\theta_{old}}$, whose update is delayed. A_t is the estimated advantage given by

$$A(s_t, a_t) = \mathcal{R}(s_t, a_t) - V(s_t), \tag{14}$$

where $V_\theta(s)$ is the learned value function, i.e., the expectation $\mathbb{E}_{a\sim\pi_\theta(a|s)}[\mathcal{R}(s,a)]$, which is derived from a critic network that is of the same size as the policy network. The clip term prevents the probability ratio from moving too far from 1, ensuring stable updates. A key drawback is its reliance on a separate critic network for advantage estimation, which substantially increases the parameter count during training.

Direct Preference Optimization (DPO) DPO represents a groundbreaking shift by entirely bypassing the need for a separate reward model. It reframes the problem of maximizing a reward under a KL-constraint as a likelihood-based objective on human preference data. Given a dataset of preferences $D = \{(y_w, y_l)\}$, where y_w is the preferred response and y_l is the dispreferred one, the DPO loss is:

$$L_{DPO}(\pi_\theta; \pi_{ref}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right)\right], \tag{15}$$

where π_{ref} is a reference policy (usually the initial SFT model), and β is a hyperparameter. While DPO eliminates the critic, its performance is intrinsically tied to the quality and coverage of its static preference dataset. Variants have emerged to address its limitations, including IPO (Identity Preference Optimization) [35] which adds a regularization term to prevent overfitting, and KTO (Kahneman-Tversky Optimization) [36], which learns from per-response binary signals (desirable/undesirable) rather than strict pairwise comparisons. See Table. 2 for more variants.

Group Relative Policy Optimization (GRPO) The remarkable success achieved by DeepSeek has catalyzed significant research interest in GRPO. Proposed to address the inefficiency of PPO's large critic, GRPO introduces a novel, lightweight evaluation paradigm. It operates on groups of responses, using their relative rewards within a group to compute advantages, thus eliminating the need for an absolute value critic. The core GRPO objective can be conceptualized as:

$$L_{GRPO} = \frac{1}{G}\sum_{g=1}^{G} \min\left(\frac{\pi_\theta(a_t^{(g)}|s_t^{(g)})}{\pi_{\theta_{old}}(a_t^{(g)}|s_t^{(g)})} \hat{A}(s_t^{(g)}, a_t^{(g)}),\ \text{clip}\left(\frac{\pi_\theta(a_t^{(g)}|s_t^{(g)})}{\pi_{\theta_{old}}(a_t^{(g)}|s_t^{(g)})}, 1-\epsilon, 1+\epsilon\right) \hat{A}(s_t^{(g)}, a_t^{(g)})\right), \tag{16}$$

where a group of outputs $\{(s_0^{(g)}, a_0^{(g)}, \ldots, s_{T-1}^{(g)}, a_{T-1}^{(g)})\}_{g=1}^{G}$ is sampled from the old policy $\pi_{\theta_{old}}$. The advantage function is estimated by

$$\hat{A}(s_t, a_t) = \frac{\mathcal{R}(s_t, a_t) - \text{mean}(\mathcal{R}(s_t^{(1)}, a_t^{(1)}), \ldots, \mathcal{R}(s_t^{(G)}, a_t^{(G)}))}{\text{std}(\mathcal{R}(s_t^{(1)}, a_t^{(1)}), \ldots, \mathcal{R}(s_t^{(G)}, a_t^{(G)}))}. \tag{17}$$

Table 2: Comparison of the popular variants of the PPO, DPO, and GRPO families. Clip corresponds to preventing the policy ratio from moving too far from 1 for ensuring stable updates. KL penalty corresponds to penalizing the KL divergence between the learned policy and the reference policy for ensuring alignment.

Method	Year	Objective Type	Clip	KL Penalty	Key Mechanism	Signal
			PPO family			
PPO [1]	2017	Policy gradient	Yes	No	Policy ratio clipping	Reward
VAPO [37]	2025	Policy gradient	Yes	Adaptive	Adaptive KL penalty + variance control	Reward + variance signal
LitePPO [38]	2025	Policy gradient	Yes	Yes	Stable advantage updates	Reward
PF-PPO [39]	2024	Policy gradient	Yes	Yes	Policy filtration	Noisy reward
VinePPO [40]	2024	Policy gradient	Yes	Yes	Unbiased value estimates	Reward
PSGPO [41]	2024	Policy gradient	Yes	Yes	Process supervision	Process Reward
			DPO family			
DPO [30]	2024	Preference optimization	No	Yes	Implicit reward related to the policy	Human preference
β-DPO [42]	2024	Preference optimization	No	Adaptive	Dynamic KL coefficient	Human preference
SimPO [43]	2024	Preference optimization	No	Scaled	Use the average log probability of a sequence as the implicit reward	Human preference
IPO [35]	2024	Implicit preference	No	No	Leverage generative LLMs as preference classifiers for reducing the dependence on external human feedback or reward models	Preference rank
KTO [36]	2024	Knowledge transfer optimization	No	Yes	Teacher stabilization	Teacher-student logit
ORPO [44]	2024	Online regularized preference optimization	No	Yes	Online stabilization	Online feedback reward
Step-DPO [45]	2024	Preference optimization	No	Yes	Step-wise supervision	Step-wise preference
LCPO [46]	2025	Preference optimization	No	Yes	Length preference with limited data and training	Reward
			GRPO family			
GRPO [32]	2025	Policy gradient under group-based reward	Yes	Yes	Group-based relative reward to eliminate value estimates	Group-based reward
DAPO [47]	2025	Surrogate of GRPO's	Yes	Yes	Decoupled clip and dynamic sampling	Dynamic group-based reward
GSPO [48]	2025	Surrogate of GRPO's	Yes	Yes	Define the importance ratio based on sequence likelihood and performs sequence-level clipping, rewarding, and optimization	Smooth group-based reward
GMPO [49]	2025	Surrogate of GRPO's	Yes	Yes	Geometric mean of token-level rewards	Margin-based reward
ProRL [50]	2025	Same as GRPO's	Yes	Yes	Reference policy reset	Group-based reward
Posterior-GRPO [51]	2025	Same as GRPO's	Yes	Yes	Reward only successful processes	Process-based reward
Dr.GRPO [52]	2025	Unbiased GRPO's objective	Yes	Yes	Eliminate the bias in optimization of GRPO	Group-based reward
Step-GRPO [53]	2025	Same as GRPO's	Yes	Yes	Rule-based reasoning rewards	Step-wise reward
SRPO [54]	2025	Same as GRPO's	Yes	Yes	Two-staged history-resampling	Reward
GRESO [55]	2025	Same as GRPO's	Yes	Yes	Pre-rollout filtering	Reward
StarPO [56]	2025	Same as GRPO's	Yes	Yes	Reasoning-guided actions for multi-turn interactions	Group-based reward
GHPO [57]	2025	Policy gradient	Yes	Yes	Adaptive prompt refinement	Reward
Skywork R1V2 [58]	2025	GRPO's with hybrid reward signal	Yes	Yes	Selective sample buffer	Multimodal reward
ASPO [59]	2025	GRPO's with shaped advantage function	Yes	Yes	Apply a clipped bias directly to advantage function	Group-based reward
TreePo [60]	2025	Same as GRPO's	Yes	Yes	Self-guided policy rollout for reducing the compute burden	Group-based reward
EDGE-GRPO [61]	2025	Same as GRPO's	Yes	Yes	Entropy-driven advantage and duided error correction to mitigate advantage collapse	Group-based reward
DARS [62]	2025	Same as GRPO's	Yes	No	Reallocate compute from medium-difficulty to the hardest problems via multi-stage rollout sampling	Group-based reward
CHORD [63]	2025	Weighted sum of GRPO's and Supervised Fine-Tuning losses	Yes	Yes	Reframe Supervised Fine-Tuning as a dynamically weighted auxiliary objective within the on-policy RL process	Group-based reward
PAPO [64]	2025	Surrogate of GRPO's	Yes	Yes	Encourage learning to perceive while learning to reason through the Implicit Perception Loss	Group-based reward
Pass@k Training [65]	2025	Same as GRPO's	Yes	Yes	Pass@k metric as the reward to continually train a model	Group-based reward

This group-relative approach is highly sample-efficient and reduces computational overhead. Consequently, a series of novel algorithms derived from the GRPO framework have been subsequently proposed (see Table. 2), aiming to substantially enhance both the sample efficiency and asymptotic performance of reinforcement learning methodologies.

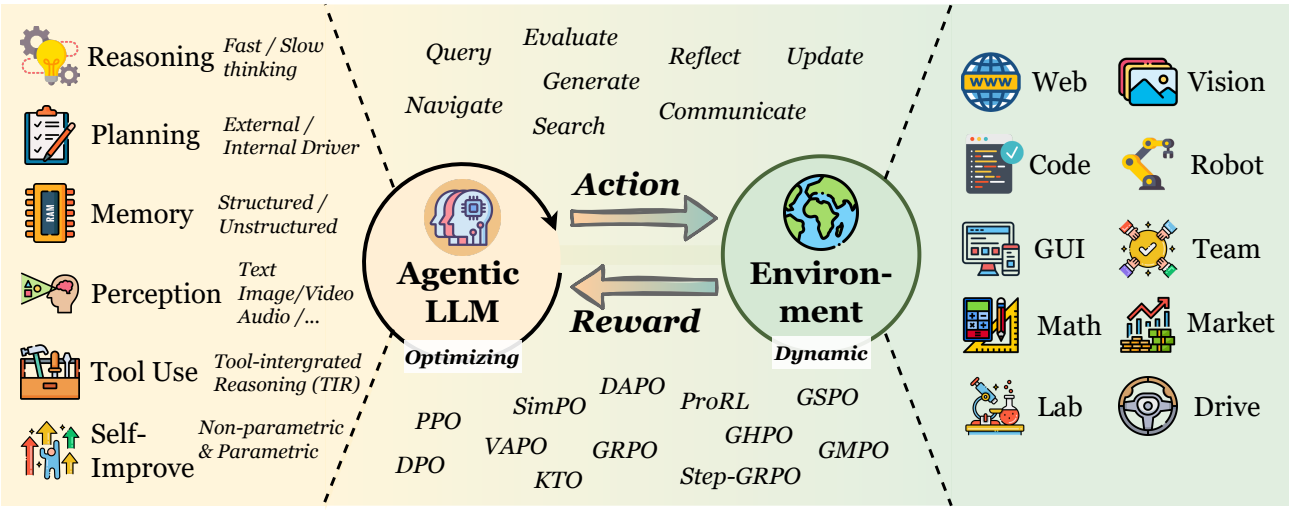

Figure 3: The dynamic interaction process between agentic LLMs and the environment.

3. Agentic RL: The model capability perspective

In this section, we conceptually characterize **Agentic RL** as the principled training of an autonomous agent composed of a set of key abilities/modules, *i.e.*, planning (Section 3.1), tool use (Section 3.2), memory (Section 3.3), self-improvement (Section 3.4), reasoning (Section 3.5), perception (Section 3.6), and others (Section 3.7), following the classic LLM agent definition [66, 67], as demonstrated in Figure 4. Traditionally, an agent pairs an LLM with mechanisms for planning (*e.g.*, task decomposition and plan selection) [68], reasoning (chain-of-thought or multi-turn inference) [69], external tool invocation [70], long- and short-term memory, and iterative reflection to self-correct and refine behavior. Agentic RL thus treats these components not as static pipelines but as interdependent policies that can be jointly optimized: RL for planning learns multi-step decision trajectories; RL for memory shapes retrieval and encoding dynamics; RL for tool use optimizes invocation timing and fidelity; and RL for reflection drives internal self-supervision and self-improvement. Consequently, our survey systematically examines how RL empowers planning, tool use, memory, reflection, and reasoning in subsequent subsections. We aim to provide a high-level conceptual delineation of RL's applications for agent capabilities, rather than an exhaustive enumeration of all related work, which we provide in Section 4.

3.1. Planning

Planning, the deliberation over a sequence of actions to achieve a goal, constitutes a cornerstone of artificial intelligence, demanding complex reasoning, world knowledge, and adaptability [71]. While initial efforts leveraged the innate capabilities of LLMs through prompting-based methods [72] (*e.g.*, ReAct [73]), these approaches lacked a mechanism for adaptation through experience. RL has emerged as a powerful paradigm to address this gap, enabling agents to refine their planning strategies by learning from environmental feedback. The integration of RL into agent planning manifests in two distinct paradigms, distinguished by whether RL functions as an **external guide** to a structured planning process or as an **internal driver** that directly evolves the LLM's intrinsic planning policy, which we will detail below.

RL as an External Guide for Planning. One major paradigm frames RL as an external guide to the planning process, where the LLM's primary role is to generate potential actions within a structured search framework.

Figure 4: A summary of the overall six aspects where RL empowers agentic LLMs. Note that the representative methods listed here are not exhaustive; please refer to our main text.

Here, RL is not employed to fine-tune the LLM's generative capabilities directly, but rather to train an auxiliary value or heuristic function [68]. This learned function then guides a classical search algorithm, such as Monte Carlo Tree Search (MCTS), by evaluating the quality of different planning trajectories. Representative works like RAP [74] and LATS [75] exemplify this approach. Planning without Search [76] extends this idea by leveraging offline goal-conditioned RL to learn a language-based value critic that guides LLM reasoning and planning without updating the LLM parameters. In this configuration, the LLM acts as a knowledge-rich action proposer, while RL provides adaptive, evaluative feedback for efficient exploration. Beyond static guidance, Learning When to Plan [77] formulates dynamic planning as an RL-driven test-time compute allocation problem, training agents to decide when to invoke explicit planning to balance reasoning performance against computational cost. Conversely, MAPF-DT [78] explores the reverse direction, employing Decision Transformer–based offline RL for decentralized multi-agent path planning, with LLM guidance enhancing adaptability and long-horizon efficiency in dynamic environments.

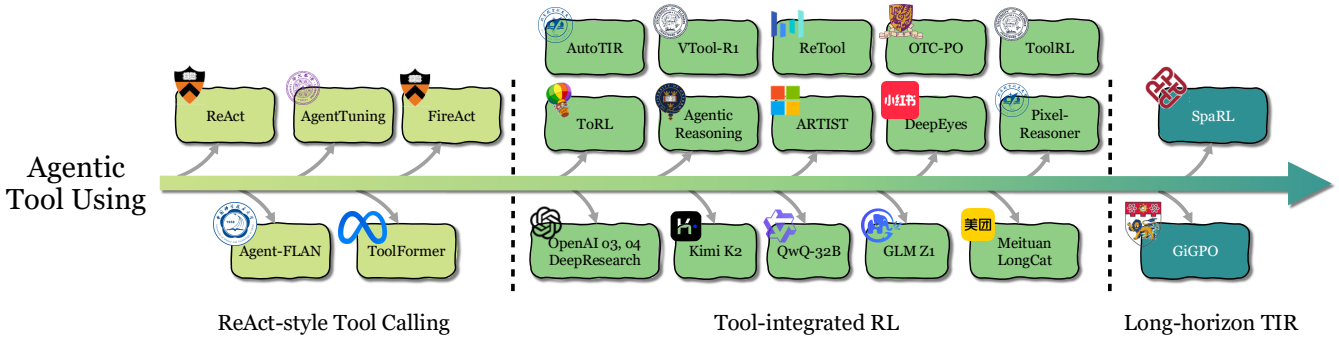

Figure 5: The development of agentic tool use. Note that we only select a small bunch of representative works here to reflect the progress.

RL as an Internal Driver of Planning. A second, more integrated paradigm positions RL as an internal driver of the agent's core planning capabilities. This approach casts the LLM directly as a policy model and optimizes its planning behavior through direct environmental interaction. Instead of guiding an external search algorithm, RL-based feedback from trial and error is used to directly refine the LLM's internal policy for generating plans. This is achieved through methods derived from RLHF, such as leveraging DPO on successful versus failed trajectories as seen in ETO [79], or through lifelong learning frameworks. For instance, VOYAGER [80] iteratively builds and refines a skill library from environmental interaction. This paradigm transforms the LLM from a static generator into an adaptive policy that continuously evolves, enhancing its robustness and autonomy in dynamic environments. In a complementary direction, Dynamic Speculative Planning (DSP) [81] embodies an online reinforcement mechanism that adapts the agent's policy to jointly optimize latency and operational cost, demonstrating that internal policy refinement can govern not only task success but also system efficiency. RLTR [82] decouples planning from answer generation and introduces tool-use rewards that directly evaluate action sequence quality, enabling focused optimization of the agent's planning capability without relying on verifiable final answers. AdaPlan and its PilotRL framework [83] leverage global plan-based guidance with progressive RL to enhance LLM agents' long-horizon planning and execution coordination in text game environments like AFLWorld and TextCraft. Planner-R1 [84] examines reward-density effects in agentic RL, showing that shaped, process-level rewards markedly improve learning efficiency and enable smaller models to attain competitive planning capability. Complementing these RL-driven approaches, the Modular Agentic Planner (MAP) [85] introduces a brain-inspired, modular architecture that decomposes planning into specialized LLM modules for conflict monitoring, state evaluation, and coordination. While not RL-based, this architecture provides a promising substrate for integrating reinforcement signals into future agentic planners.

Prospective: The Synthesis of Deliberation and Intuition. The prospective horizon for agentic planning lies in the synthesis of these two paradigms: moving beyond the distinction between external search and internal policy optimization. The ultimate goal is to develop an agent that **internalizes the structured search process itself**, seamlessly blending intuitive, fast plan generation with deliberate, slow, deliberative reasoning. In such a model, RL would not only refine the final plan but also optimize a meta-policy governing the deliberation process: learning when to explore alternative paths, how to prune unpromising branches, and how deeply to reason before committing to an action. This would transform the LLM agent from a component that either proposes actions or acts as a raw policy into an integrated reasoning engine.

3.2. Tool Using

RL has emerged as a pivotal methodology for evolving tool-enabled language agents from post-hoc, ReAct-style pipelines to deeply interleaved, multi-turn Tool-Integrated Reasoning (TIR) systems. While early paradigms successfully demonstrated the feasibility of tool invocation, their reliance on SFT or prompt engineering limited agents to mimicking static patterns, lacking the strategic flexibility to adapt to novel scenarios or recover from errors. Agentic RL addresses this by shifting the learning paradigm from imitation to outcome-driven optimization, enabling agents to autonomously discover when, how, and which tools to deploy. This evolution charts a clear trajectory, which we explore in three stages. We begin with (1) early ReAct-style tool calling, then examine (2) modern tool-integrated reasoning (TIR) that deeply embeds tool use within cognitive loops, and finally, discuss the prospective challenge of (3) multi-turn TIR, focusing on temporal credit assignment for robust, long-horizon performance.

ReAct-style Tool Calling. Early paradigms for tool invocation predominantly relied on either prompt engineering or SFT to elicit tool-use behaviors. The **(I) prompt engineering** approach, exemplified by ReAct [73], leveraged few-shot exemplars to guide an LLM to interleave reasoning traces and actions within a "Thought-Action-Observation" cycle, capitalizing on the model's in-context learning abilities. Going beyond, **(II) SFT-based methods** were introduced to internalize models' tool-use capabilities. Frameworks like Toolformer [86] employed a self-supervised objective to teach models where to insert API calls, while others like FireAct [87], AgentTuning [88], Agent-FLAN [89] fine-tuned models on expert-generated or curated datasets of tool-interaction trajectories (e.g., AgentBank [90], APIBank [91]). Although SFT improved the reliability of tool invocation, both of these early approaches are fundamentally constrained by their imitative nature. They train agents to replicate static, pre-defined patterns of tool use, thereby lacking the strategic flexibility to adapt to novel scenarios or recover from unforeseen errors, a limitation that RL-centric approaches directly address by shifting the learning objective from imitation to outcome-driven optimization.

Tool-integrated RL. Building on the limitations of purely imitative paradigms, RL-based approaches for tool use shift the objective from replicating fixed patterns to optimizing end-task performance. This transition enables agents to *strategically* decide *when*, *how*, and *in what combination* to invoke tools, adapting dynamically to novel contexts and unforeseen failures. At the foundation, frameworks such as ToolRL [92] demonstrate that, even when initialized from base models without any imitation traces, RL training can elicit emergent capabilities, e.g., self-correction of faulty code, adaptive adjustment of invocation frequency, and the composition of multiple tools for complex sub-tasks. Subsequently, a recent surge in research has produced works such as OTC-PO [93], ReTool [94], AutoTIR [95], VTool-R1 [96], DeepEyes [97], Pixel-Reasoner [98], Agentic Reasoning [99], ARTIST [100], ToRL [101] and numerous other works [102, 103, 104, 105, 106, 107, 108, 109, 110], which employ RL policies that interleave symbolic computation (e.g., code execution, image editing) with natural-language reasoning within a single rollout. This integrated control loop allows the agent to balance precise, tool-mediated operations with flexible verbal inference, tailoring the reasoning process to the evolving task state. Recent work [59] theoretically proves that TIR fundamentally expands LLM capabilities beyond the "invisible leash" of pure-text RL by introducing deterministic tool-driven state transitions, establishes token-efficiency arguments for feasibility under finite budgets, and proposes Advantage Shaping Policy Optimization (ASPO) to stably guide agentic tool use.

Today, such tool-integrated reasoning is no longer a niche capability but a baseline feature of advanced agentic models. Mature commercial and open-source systems, such as OpenAI's DeepResearch and o3 [111], Kimi K2 [112], Qwen QwQ-32B [113], Zhipu GLM Z1 [114], Microsoft rStar2-Agent [115] and Meituan LongCat [116], routinely incorporate these RL-honed strategies, underscoring the centrality of outcome-driven optimization in tool-augmented intelligence.

Prospective: Long-horizon TIR. While tool-integrated RL has proven effective for optimizing actions within a single reasoning loop, the primary frontier lies in extending this capability to robust, long-horizon tasks that require multi-turn reasoning [117]. This leap is fundamentally bottlenecked by the challenge of temporal credit assignment [118]. Current RL approaches often depend on sparse, trajectory-level/outcome-based rewards, making it difficult to pinpoint which specific tool invocation in a long, interdependent sequence contributed to success or failure. While nascent research has begun to explore more granular reward schemes, such as turn-level advantage estimation in GiGPO [119] and SpaRL [120], these are still early steps. Consequently, developing more granular credit assignment mechanisms that can accurately guide the agent through complex decision chains without inadvertently punishing useful exploration or promoting reward hacking remains a critical and largely unsolved problem for advancing agentic systems.

3.3. Memory

Agentic RL transforms memory modules from passive data stores into dynamic, RL-controlled subsystems, deciding what to store, when to retrieve, and how to forget similar to human [121]. This section traces this evolution through four representative phases.

RL in RAG-style Memory. Early systems (*e.g.*, retrieval-augmented generation) treated memory as an external datastore; when RL was employed at all, it solely regulated when to perform queries. Several classic memory systems without RL involvement, such as MemoryBank [122], MemGPT [123], and HippoRAG [124], adopt predefined memory management strategies that specify how to *store*, *integrate*, and *retrieve* information (*e.g.*, storage via vector databases or knowledge graphs; retrieval based on semantic similarity or topological connectivity). Subsequently, RL was incorporated into the memory management pipeline as a functional component. A notable example is the framework proposed in [125], where the RL policy adjusts retrieval behavior through *prospective reflection* (multi-level summarization) and *retrospective reflection* (reinforcing retrieval outcomes). Nevertheless, the memory medium itself remained static (*e.g.*, simple vector store or summary buffer), and the agent exerted no control over the write processes. Recently, Memory-R1 [126] introduces a RL-based memory-augmented Agent framework where a Memory Manager learns to perform structured operations (ADD/UPDATE/DELETE/NOOP) via PPO or GRPO based on downstream QA performance, while an Answer Agent employs a Memory Distillation policy over RAG-retrieved entries to reason and answer. Follow-up works like Mem-α [127] and Memory-as-action [128] have also explored RL for training agents into automatic memory manager.

RL for Token-level Memory. Subsequent advancements introduced models equipped with explicit, trainable memory controllers, enabling agents to regulate their own memory states (often stored in token form) without relying on fixed, external memory systems. Notably, such memory is commonly instantiated in two forms. The first is **(I) explicit tokens**, corresponding to human-readable natural language. For example, in MemAgent [129], the agent maintains a natural-language memory pool alongside the LLM, with an RL policy determining, at each segment, which tokens to retain or overwrite, effectively compressing long-context inputs into concise, informative summaries. Similar approaches include MEM1 [130] and Memory Token [131], both of which explicitly preserve a pool of natural-language memory representations. More frequently, works like ReSum [132], context folding [133] has also explored RL for context memory management. The second form is **(II) implicit tokens**, where memory is maintained in the form of latent embeddings. A representative line of work includes MemoryLLM [134] and M+ [135], in which a fixed set of latent tokens serves as "memory tokens." As the context evolves, these tokens are repeatedly retrieved, integrated into the LLM's forward computation, and updated, thereby preserving contextual information and exhibiting strong resistance to forgetting. Unlike explicit tokens, these memory tokens are not tied

Table 3: An overview of three classic categories of agent memory; works marked with † directly employ RL. The list here is not exhaustive, and we refer readers interested in broader agent memory to [121].

Method	Type	Key Characteristics
	RAG-style Memory	
MemoryBank [122]	External Store	Static memory with predefined storage/retrieval rules
MemGPT [123]	External Store	OS-like agent with static memory components
HippoRAG [124]	External Store	Neuro-inspired memory with heuristic access
Prospect† [125]	RL-guided Retrieval	Uses RL for reflection-driven retrieval adjustment
Memory-R1† [126]	RL-guided Retrieval	RL-driven memory management: ADD/UPDATE/DELETE/NOOP
Mem-α† [127]	RL-guided Retrieval	RL-guided agents for memory retrieval
Memory-as-action [128]	RL-guided Manage	Ene-to-end training agents for memory management
	Token-level Memory	
MemAgent† [129]	Explicit Token	RL controls which NL tokens to retain or overwrite
MEM1† [130]	Explicit Token	Memory pool managed by RL to enhance context handling
Memory Token [131]	Explicit Token	Structured memory for reasoning disentanglement
ReSum† [132]	Explicit Token	Turn-wise Interaction summary for ReAct agents
Context Folding† [133]	Explicit Token	Context folding for ReAct agents
MemoryLLM [134]	Latent Token	Latent tokens repeatedly integrated and updated
M+ [135]	Latent Token	Scalable memory tokens for long-context tracking
IMM [136]	Latent Token	Decouples word representations and latent memory
Memory [137]	Latent Token	Forget-resistant memory tokens for evolving context
MemGen† [138]	Latent Token	Context-sensitive latent token as memory carriers
	Structured Memory	
Zep [139]	Temporal Graph	Temporal knowledge graph enabling structured retrieval
A-MEM [140]	Atomic Memory Notes	Symbolic atomic memory units; structured storage
G-Memory [141]	Hierarchical Graph	Multi-level memory graph with topological structure
Mem0 [142]	Structured Graph	Agent memory with full-stack graph-based design

to human-readable text but rather constitute a machine-native form of memory. Related efforts include IMM [136] and Memory [137]. Across both paradigms, these approaches empower agents to autonomously manage their memory banks, delivering significant improvements in long-context understanding, continual adaptation, and self-improvement. MemGen [138] for the first time proposes the paradigm of leveraging latent memory tokens for carrying and generating experiential knowledge, posing promising directions for RL-based latent memory.

Prospective: RL for Structured Memory. Building on token-level approaches, recent trends are moving toward *structured* memory representations, which organize and encode information beyond flat token sequences. Representative examples include the temporal knowledge graph in Zep [139], the atomic memory notes in A-MEM [140], and the hierarchical graph-based memory designs in G-Memory [141] and Mem0 [142]. These systems capture richer relational, temporal, or hierarchical dependencies, enabling more precise retrieval and reasoning. However, their management, spanning insertion, deletion, abstraction, and linkage updates, has thus far been governed by handcrafted rules or heuristic strategies. To date, little work has explored the use of RL to dynamically control the construction, refinement, or evolution of such structured memory, making this an open and promising direction for advancing agentic memory capabilities.

3.4. Self-Improvement

As LLM agents evolve, recent research increasingly emphasizes RL as a mechanism for ongoing reflection, enabling agents to learn from their own mistakes across planning, reasoning, tool use, and memory [143]. Rather than relying exclusively on data-driven training phases or static reward models, these systems incorporate *iterative, self-generated feedback loops*, ranging from prompt-level heuristics to fully fledged RL controllers, to guide agents toward continual self-improvement.

RL for Verbal Self-correction. Initial methods in this vein leveraged prompt-based heuristics, sometimes referred to as *verbal reinforcement learning*, where agents generate an answer, linguistically reflect on its potential errors, and subsequently produce a refined solution, all within a single inferential pass without gradient updates. Prominent examples include Reflexion [144], Self-refine [145], CRITIC [146], and Chain-of-Verification [147]. For instance, the Self-Refine [145] protocol directs an LLM to iteratively polish its output using three distinct prompts for generation, feedback, and refinement, proving effective across domains like reasoning and programming. To enhance the efficacy and robustness of such self-reflection, several distinct strategies have been developed: **(I) multiple sampling**, which involves generating multiple output rollouts by sampling from the model's distribution. By aggregating critiques or solutions from multiple attempts, the agent can improve the consistency and quality of its self-reflection. This method has been widely studied in works like If-or-Else [148], UALA [149] and Multi-agent Verification [150]. This approach is conceptually analogous to test-time scaling techniques, so we refer the reader to [118] for more details; **(II) structured reflection workflows**, rather than prompting for a monolithic reflection on a final answer, prescribe a more dedicated and granular workflow. For example, Chain-of-Verification [147] manually decomposes the process into distinct "Retrieving, Rethinking, and Revising" stages; **(III) external guidance**, which grounds the reflection process in verifiable, objective feedback by incorporating external tools. These tools include code interpreter as seen in Self-Debugging [151], CAD modeling programs in Luban [152], mathematical calculators in T1 [153], step-wise reward models [154], and others [146].

RL for Internalizing Self-correction. While verbal self-correction offers a potent inference-time technique, its improvements are ephemeral and confined to a single session. To instill a more durable and generalized capability for self-improvement, subsequent research has employed RL with gradient-based updates to internalize these reflective feedback loops directly into the model's parameters and to fundamentally enhance the model's inherent ability to identify and correct its own errors. This paradigm has been applied across multiple domains. For instance, KnowSelf [155] leverages DPO and RPO [156] to enhance agents' self-reflection capabilities in text-based game environments, while Reflection-DPO [157] focuses on user–agent interaction scenarios, enabling agents to better infer user intent through reflective reasoning. DuPo [158] employs RL with dual-task feedback to enable annotation-free optimization, enhancing LLM agents' self-correction across translation, reasoning, and reranking tasks. SWEET-RL [159] and ACC-Collab [160] adopt a slightly different setting from the above works: they train an external critic model to provide higher-quality revision suggestions for the actor agent's actions. Nonetheless, the underlying principle remains closely aligned.

RL for Iterative Self-training. Moving toward full agentic autonomy, the third and most advanced class of models combines reflection, reasoning, and task generation into a self-sustaining loop, enabling unbounded self-improvement without human-labeled data. These methods can be distinguished by the architecture of their learning loops: **(I) Self-play and search-guided refinement**, which emulates classic RL paradigms like AlphaZero. R-Zero [161], for instance, employs a Monte Carlo Tree Search (MCTS) to explore a reasoning tree, using the search results to iteratively train both a policy LLM (the actor) and a value LLM (the critic)

entirely from scratch. Similarly, the ISC framework [162] operationalizes a cycle of "Imagination, Searching, and Criticizing," where the agent generates potential solution paths, uses a search algorithm to explore them, and applies a critic to refine its reasoning strategy before producing a final answer. **(II) Execution-guided curriculum generation**, where the agent creates its own problems and learns from verifiable outcomes. Absolute Zero [163] exemplifies this by proposing its own tasks, attempting solutions, verifying them via execution, and using the outcome-based reward to refine its policy. Similarly, Self-Evolving Curriculum [164] enhances this process by framing problem selection itself as a non-stationary bandit task, allowing the agent to strategically generate a curriculum that maximizes its learning gains over time. TTRL [165] applies this principle for on-the-fly adaptation to a single problem. At test time, it uses execution-based rewards to rapidly fine-tune a temporary copy of the agent's policy for the specific task at hand; this specialized policy is then used to generate the final answer before being discarded. Though differing in whether the learning is permanent or ephemeral, all these methods underscore a powerful, unified strategy: harnessing execution-based feedback to autonomously guide the agent's reasoning process. ALAS [166] constructs an autonomous pipeline that crawls web data, distills it into training signals, and continuously fine-tunes LLMs, thereby enabling self-training and self-evolution without manual dataset curation. **(III) Collective bootstrapping**, where learning is accelerated by aggregating shared experience. SiriuS [167], for example, constructs and augments a live repository of successful reasoning trajectories from multi-agent interactions, using this growing knowledge base to bootstrap its own training curriculum. MALT [168] shares a similar motivation, yet its implementation is limited to a three-agent setup. Nevertheless, all these methods define feedback loops that are internally generated and continuously evolving, representing a significant step toward truly autonomous agents.

Prospective: Meta Evolution of Reflection Ability. While current research successfully uses RL to refine an agent's behavior through reflection, the reflection process itself remains largely handcrafted and static. The next frontier lies in applying RL at a higher level of abstraction to enable **meta-learning for adaptive reflection**, focusing not just on correcting an error, but on learning how to self-correct more effectively over time. In this paradigm, the agent may learn a meta-policy that governs its own reflective strategies. For instance, it could learn to dynamically choose the most appropriate form of reflection for a given task, deciding whether a quick verbal check is sufficient or if a more costly, execution-guided search is necessary. Furthermore, an agent could use long-term outcomes to evaluate and refine the very heuristics it uses for self-critique, effectively learning to become a better internal critic. By optimizing the reflective mechanism itself, this approach moves beyond simple self-correction and toward a state of continuous self-improvement in the learning process, representing a crucial step toward agents that can not only solve problems but also autonomously enhance their fundamental capacity to learn from experience.

3.5. Reasoning

Reasoning in large language models can be broadly categorized into *fast reasoning* and *slow reasoning*, following the dual-process cognitive theory [169, 24]. Fast reasoning corresponds to rapid, heuristic-driven inference with minimal intermediate steps, while slow reasoning emphasizes deliberate, structured, and multi-step reasoning. Understanding the trade-offs between these two paradigms is crucial for designing models that balance efficiency and accuracy in complex problem-solving.

Fast Reasoning: Intuitive and Efficient Inference Fast reasoning models operate in a manner analogous to System 1 [18] cognition: quick, intuitive, and pattern-driven. They generate immediate responses without explicit step-by-step deliberation, excelling in tasks that prioritize fluency, efficiency, and low latency. Most conventional LLMs fall under this category, where reasoning is implicitly encoded in next-token

prediction [2, 170]. However, this efficiency comes at the cost of systematic reasoning, making these models more vulnerable to factual errors, biases, and shallow generalization.

To address the severe hallucination problems in fast reasoning, current research has largely focused on various direct approaches. Some studies attempt to mitigate errors and hallucinations in the next-token prediction paradigm by leveraging internal mechanisms [171, 172, 173] or by simulating human-like cognitive reasoning. Other works propose introducing both external and internal confidence estimation methods [174, 175] to identify more reliable reasoning paths. However, constructing such external reasoning frameworks often risks algorithmic adaptivity issues and can easily fall into the complexity trap.

Slow Reasoning: Deliberate and Structured Problem Solving In contrast, slow reasoning models are designed to emulate System 2 cognition [18] by explicitly producing intermediate reasoning traces. Techniques such as chain-of-thought prompting, multi-step verification [176], and reasoning-augmented reinforcement learning allow these models to engage in deeper reflection and achieve greater logical consistency. While slower in inference due to extended reasoning trajectories, they achieve higher accuracy and robustness in knowledge-intensive tasks such as mathematics, scientific reasoning, and multi-hop question answering [177]. Representative examples include OpenAI's o1 [31] and o3 series [33], DeepSeek-R1 [32], as well as methods that incorporate dynamic test-time scaling [178, 179, 180, 172] or reinforcement learning [181, 47, 182, 183, 184, 185] for reasoning.

Modern slow reasoning exhibits output structures that differ substantially from fast reasoning. These include a clear exploration and planning structure, frequent verification and checking behaviors, and generally longer inference lengths and times. Past work has explored diverse patterns for constructing long-chain reasoning outputs. Some methods—Macro-o1, HuatuoGPT-o1, and AlphaZero—have attempted to synthesize long chains-of-thought via structured, agentic search [186, 187, 188]. Other approaches focus on generating long-CoT datasets that embody specific deliberative or reflective thinking patterns; examples include HiICL-MCTS, LLaVA-CoT, rStar-Math, and ReasonFlux [189, 190, 191, 192]. Recent approaches that perform reasoning in the latent space leverage latent representations to conduct parallel reasoning and explore diverse reasoning trajectories [193, 194]. With the progress of pretrained foundation models, more recent work has shifted toward self-improvement paradigms—frequently instantiated with reinforcement learning—to further enhance models' reasoning capabilities [181, 47].

Prospective: Integrating Slow Reasoning Mechanisms into Agentic Reasoning The dichotomy between fast and slow reasoning highlights an open challenge in agentic reasoning: how to employ reinforcement learning for reliably training slow-thinking reasoning capabilities in agentic scenarios. Reinforcement learning in agentic scenarios faces greater challenges in training stability, such as ensuring compatibility with diverse environments. Agentic reasoning itself is also susceptible to overthinking problems. Purely fast models may overlook critical reasoning steps, while slow models often suffer from excessive latency or **overthinking behaviors**, such as unnecessarily long chains of thought. Emerging approaches seek hybrid strategies [195] that combine the efficiency of fast reasoning with the rigor of slow reasoning [196, 197, 198, 199]. For instance, adaptive test-time scaling allows a model to decide whether to respond quickly or to engage in extended deliberation depending on task complexity. Developing such cognitively-aligned mechanisms is a key step toward building reasoning agents that are both efficient and reliable.

3.6. Perception

By bridging visual perception with linguistic abstraction, Large Vision–Language Models (LVLMs) have demonstrated unprecedented capabilities for perceiving and understanding multimodal content [200, 201, 202, 203, 204, 205, 206, 207]. Central to this progress is the incorporation of explicit reasoning mechanisms into multimodal learning frameworks [208, 209], moving beyond passive perception toward active visual cognition [210]. RL has emerged as a powerful paradigm for this purpose, enabling the alignment of vision–language–action models with complex, multi-step reasoning objectives that go beyond the constraints of supervised next-token prediction [211, 212].

From Passive Perception to Active Visual Cognition Multimodal content often requires nuanced, context-dependent interpretation. Inspired by the remarkable success of RL in enhancing reasoning within LLMs [32, 213], researchers have increasingly sought to transfer these gains to multimodal learning [214, 215]. Early efforts focused on preference-based RL to strengthen the Chain-of-Thought (CoT) reasoning ability of MLLMs [216, 217, 218]. Visual-RFT [219] and Reason-RFT [220] directly apply GRPO to the vision domain, adaptively incorporating vision-specific metrics such as IoU as verifiable reward signals, while STAR-R1 [221] extended this idea by introducing partial rewards tailored for visual GRPO. Building upon this, a series of approaches—Vision-R1 [222], VLM-R1 [214], LMM-R1 [215], and MM-Eureka [223]—developed specialized policy optimization algorithms designed to incentivize step-wise visual reasoning, demonstrating strong performance even on smaller 3B-parameter models. SVQA-R1 [224] introduced Spatial-GRPO, a novel groupwise RL method that enforces view-consistent and transformation-invariant objectives. Visionary-R1 [225] enforces image captioning as a prerequisite step before reasoning, mitigating shortcut exploitation during reinforcement finetuning. A line of curriculum-learning methods have also been proposed to ease and smooth the RL training process of vision reinforcement finetuning [226, 227, 228, 229, 217]. R1-V [227] introduces VLM-Gym and trains G0/G1 models via scalable, pure RL self-evolution with a perception-enhanced cold start, yielding emergent perception–reasoning synergy across diverse visual tasks. R1-Zero [230] shows that even simple rule-based rewards can induce self-reflection and extended reasoning in non-SFT models, surpassing supervised baselines. PAPO [64] proposes a perception-aware policy optimization framework that augments RLVR methods with an implicit perception KL loss and double-entropy regularization, while [231] proposes a summarize-and-then-reason framework under RL training to mitigate visual hallucinations and improve reasoning without dense human annotations. Collectively, these approaches demonstrate that R1-style RL can be successfully transferred to the vision domain, provided that well-designed, verifiable reward metrics are used—yielding significant improvements in performance, robustness, and out-of-distribution generalization.

More recent work explores another key advantage of RL: moving beyond the formulation of tasks as passive perception, where static, verifiable rewards are computed only on the text-based outputs of LVLMs. Instead, RL can be used to incentivize active cognition over multimodal content—treating visual representations as manipulable and verifiable intermediate thoughts. This paradigm empowers models not merely to "look and answer," but to actively see, manipulate, and reason with visual information as part of a multi-step cognitive process [210].

Grounding-Driven Active Perception. To advance from passive perception to active visual cognition, a key direction is enabling LVLMs to repeatedly look back and query the image while generating their reasoning process. This is achieved through grounding [232, 233], which anchors each step of the generated chain-of-thought (CoT) to specific regions of the multimodal input—facilitating more valid and verifiable

reasoning by explicitly linking text with corresponding visual regions.

To begin with, GRIT [234] interleaves bounding-box tokens with textual CoT and uses GRPO with both verifiable rewards and bounding-box correctness as supervision. [235] introduces a simple point-and-copy mechanism that allows the model to dynamically retrieve relevant image regions throughout the reasoning process. Ground-R1 [236] and BRPO [237] highlight evidence regions (via IoU-based or reflection rewards) prior to text-only reasoning, while DeepEyes [97] demonstrates that end-to-end RL can naturally induce such grounding behaviors. Chain-of-Focus further refines this approach by grounding CoT steps followed by zooming in operations.

Tool-Driven Active Perception. Another promising direction for enabling active perception is to frame visual cognition as an agentic process, where external tools, code snippets, and runtime environments assist the model's cognitive workflow [238, 239]. For instance, VisTA [240] and VTool-R1 [241] focus on teaching models how to select and use visual tools through RL, while OpenThinkIMG [242] provides standardized infrastructure for training models to "think with images." Finally, Visual-ARFT [219] leverages RL to facilitate tool creation, harnessing the code-generation capabilities of MLLMs to dynamically extend their perceptual toolkit. Pixel Reasoner [98] expands the model's action space with operations such as crop, erase, and paint, and introduces curiosity-driven rewards to discourage premature termination of exploration.

Generation-Driven Active Perception. In addition to grounding and tool use, humans employ one of their most powerful cognitive abilities—imagination—to produce sketches or diagrams that aid problem-solving. Inspired by this, researchers have begun equipping LVLMs with the ability to generate sketches or images interleaved with chain-of-thought (CoT) reasoning, enabling models to externalize intermediate representations and reason more effectively [243, 244, 245]. Visual Planning [243] proposes to use imagined image rollouts only as the CoT images thinking, using downstream task success as the reward signal. GoT-R1 [246] applies RL within the Generation-CoT framework, allowing models to autonomously discover semantic–spatial reasoning plans before producing the image. Similarly, T2I-R1 [247] explicitly decouples the process into a semantic-level CoT for high-level planning and a token-level CoT for patch-wise pixel generation, jointly optimizing both stages with RL.

Audio. RL has also been extended beyond vision–language models to a diverse range of modalities, including audio. Within the audio–language domain, we categorize RL applications into two broad classes. (1) Reasoning enhancement for large audio–language models: RL is leveraged to guide models in producing structured, step-by-step reasoning chains for tasks such as audio question answering and logical inference [248, 249, 250, 250, 248]. (2) Fine-grained component optimization in speech synthesis (TTS): RL is employed to directly refine system components—for example, improving duration predictors—using perceptual quality metrics such as speaker similarity and word error rate as reward signals, thereby yielding more natural and intelligible speech [251]. Some other works such as EchoInk-R1 [252] further enrich visual reasoning by integrating audio–visual synchrony under GRPO optimization.

3.7. Others

Beyond optimizing the above core cognitive modules, agentic RL also strengthens the ability to maintain strategic coherence over extended, **multi-turn interactions**. Here, RL is applied to support long-horizon reasoning and effective credit assignment.

For long-horizon interactions, the central challenge is temporal credit assignment [118], where sparse and delayed feedback obscures the link between an agent's actions and a distant outcome. Agentic RL directly confronts this by evolving both the learning signal and the optimization framework. One major approach is the **(I) integration of process-based supervision with final outcome rewards.** Rather than relying on a single reward at a trajectory's conclusion, this paradigm uses *auxiliary models* or *programmatic rules* to evaluate the quality of intermediate steps, providing a denser and more immediate learning signal that guides the agent's multi-turn strategy. For example, EPO [253], ThinkRM [254], SPO [255], and AgentPRM [256] introduce external reward models to provide step-wise signals for agents; in contrast, RLVMR [257] designs manually defined, programmatic rules to guide the intermediate supervision. A second, complementary strategy is to **(II) extend preference optimization from single turns to multi-step segments.** Techniques like Segment-level DPO (SDPO) [258] move beyond comparing isolated responses and instead construct preference data over entire conversational snippets or action sequences. This allows the model to directly learn how early decisions influence long-term success, thereby refining its ability to maintain strategic coherence in extended dialogues and complex tasks.

4. Agentic RL: The Task Perspective

Agentic RL manifests through a wide spectrum of concrete tasks that test and shape its evolving capabilities. This section surveys representative application domains where Agentic RL has demonstrated remarkable potential and unique challenges. We begin with *search and information retrieval* (Section 4.1), followed by *code generation and software engineering* (Section 4.2), and *mathematical reasoning* (Section 4.3). We then discuss its role in *GUI navigation* (Section 4.4), *vision understanding tasks* (Section 4.5) as well as *VLM embodied interaction* (Section 4.6). Beyond single-agent scenarios, we extend the perspective to *multi-agent systems* (Section 4.7) and conclude with other emerging domains (Section 4.8). Together, these applications highlight how agentic RL transitions from abstract paradigms into actionable, real-world problem solving, as illustrated in Figure 6.

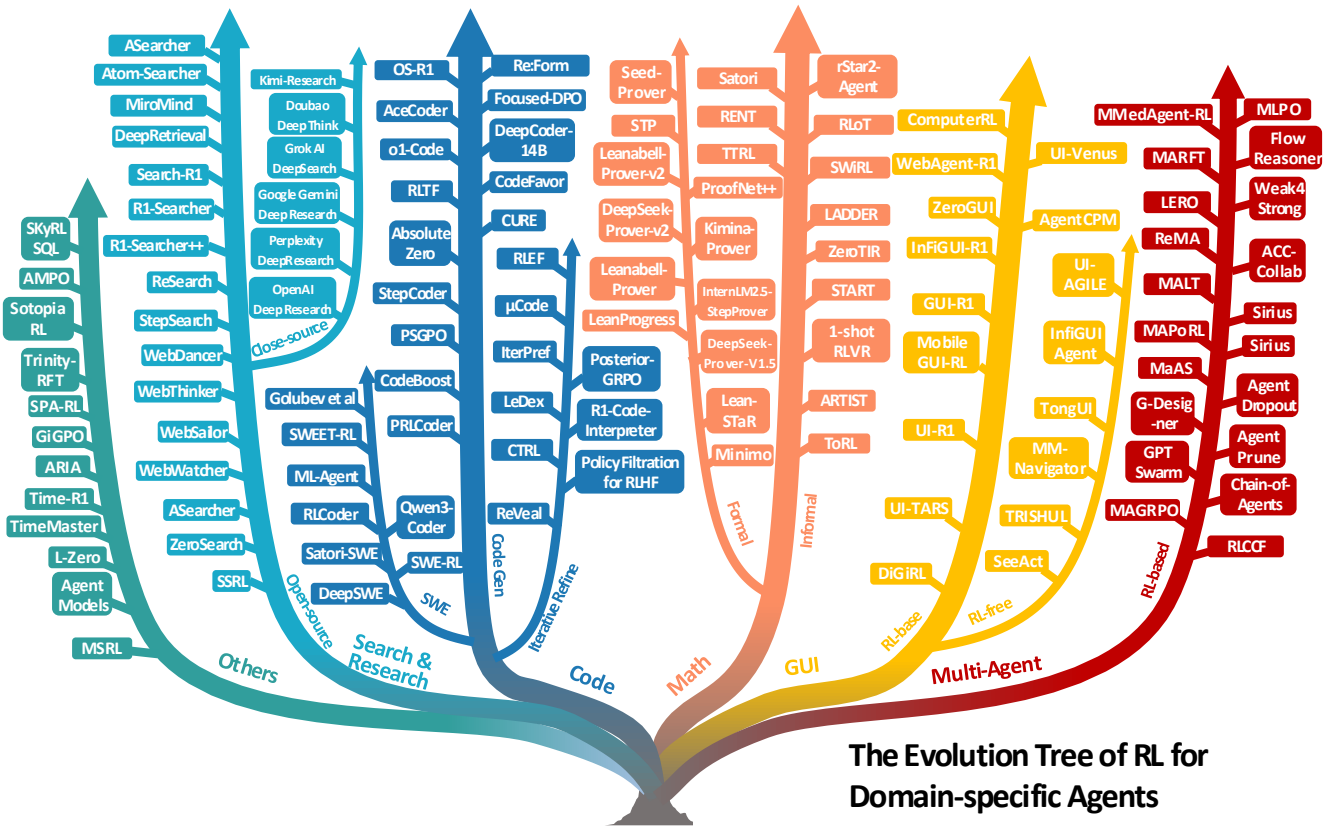

Figure 6: The evolution tree of RL for domain-specific agents.

4.1. Search & Research Agent

Search has been central to extending LLMs with external knowledge, with Retrieval-Augmented Generation (RAG) as a widely used approach [259, 260]. The paradigm is now evolving beyond simple information retrieval towards creating autonomous agents capable of *deep research*: complex, multi-step processes that involve not just finding information but also performing in-depth analysis, synthesizing insights from numerous sources, and drafting comprehensive reports [112, 261]. This shift elevates the objective from answering queries to tackling complex research tasks. Early prompt-driven methods relied on brittle query strategies and manual engineering. While more recent works like Search-o1 [107] leverage large reasoning models for agentic, inference-time retrieval, and multi-agent systems such as DeepResearch [262] coordinate querying and summarization sub-agents, they still lack learning signals. These prompt-based methods lack any fine-tuning signal, leading to limited generalization and poor effectiveness in multi-turn settings that demand a tight loop of search, reasoning, and synthesis. These limitations have led to the adoption of reinforcement learning to directly optimize the end-to-end process of query generation and search–reasoning coordination for advanced research objectives. Table 4 presents the majority of works studied in this section. In the following, we will detail how RL empowers these agents.

Table 4: A summary of RL-based methods for search and research agents.

Method	Category	Base LLM	Resource Link
Open Source Methods			
DeepRetrieval [263]	External	Qwen2.5-3B-Instruct, Llama-3.2-3B-Instruct	GitHub
Search-R1 [264]	External	Qwen2.5-3B/7B-Base/Instruct	GitHub
R1-Searcher [265]	External	Qwen2.5-7B, Llama3.1-8B-Instruct	GitHub
R1-Searcher++ [266]	External	Qwen2.5-7B-Instruct	GitHub
ReSearch [108]	External	Qwen2.5-7B/32B-Instruct	GitHub
StepSearch [267]	External	Qwen2.5-3B/7B-Base/Instruct	GitHub
DeepResearcher [268]	External	Qwen2.5-7B-Instruct	GitHub
WebDancer [106]	External	Qwen2.5-7B/32B, QWQ-32B	GitHub
WebThinker [269]	External	QwQ-32B, DeepSeek-R1-Distilled-Qwen, Qwen2.5-32B	GitHub
WebSailor [105]	External	Qwen2.5-3B/7B/32B/72B	GitHub
WebWatcher [270]	External	Qwen2.5-VL-7B/32B	GitHub
WebShaper [271]	External	Qwen-2.5-32B/72B, QwQ-32B	GitHub
ASearcher [117]	External	Qwen2.5-7B/14B, QwQ-32B	GitHub
Atom-Searcher [272]	External	Qwen2.5-7B-Instruct	GitHub
MiroMind Open Deep Research [273]	External	-	Website
SimpleDeepResearcher [274]	External	QwQ-32B	GitHub
AWorld [275]	External	Qwen3-32B	GitHub
SFR-DeepResearch [276]	External	QwQ-32B, Qwen3-8B, GPT-oss-20b	-
ZeroSearch [277]	Internal	Qwen2.5-3B/7B-Base/Instruct	GitHub
SSRL [278]	Internal	Qwen2.5,Llama-3.2/Llama-3.1, Qwen3	GitHub
Closed Source Methods			
OpenAI Deep Research [111]	External	OpenAI Models	Website
Perplexity's DeepResearch [261]	External	-	Website
Google Gemini's DeepResearch [279]	External	Gemini	Website
Kimi-Researcher [112]	External	Kimi K2	Website
Grok AI DeepSearch [280]	External	Grok3	Website
Doubao with Deep Think [281]	External	Doubao	Website
Manus WideResearch	External	-	Website

4.1.1. Open Source RL Methods

Search from the external Internet A major line of work builds on the RAG foundation but relies on *real-time web search APIs* as the external environment, using reinforcement learning to optimize query generation and multi-step reasoning. Early progress was spearheaded by DeepRetrieval [263], which framed one-shot query generation as a GRPO-trained policy and directly rewarded recall and relevance against live search results. Motivated by its gains, subsequent methods extended the paradigm into multi-turn, reasoning-integrated, and multi-modal search. Search-R1 [264] and DeepResearcher [268] integrates retrieved-token masking with outcome-based rewards to interleave query formulation and answer generation. AutoRefine [282] further advances this trajectory by inserting refinement phases between successive search calls, using GRPO to reward not only answer correctness but also retrieval quality, enabling agents to iteratively filter and structure noisy evidence during long-horizon reasoning. R1-Searcher [265] employs a two-stage, cold-start PPO strategy—first learning when to invoke web search, then how to exploit it—while its successor R1-Searcher++ [266] adds supervised fine-tuning, internal-knowledge rewards to avoid redundancy, and dynamic memory for continual assimilation. ReSearch [108] pursues fully end-to-end PPO without supervised tool-use trajectories, while StepSearch [267] accelerates convergence on multi-hop QA by assigning intermediate step-level rewards. Atom-Searcher [272] is an agentic deep research framework

that significantly improves LLM problem-solving by refining the reasoning process itself, not just the final outcome. WebDancer [106] leverages human browsing trajectory supervision plus RL fine-tuning to produce autonomous ReAct-style agents, excelling on GAIA [283] and WebWalkerQA [284]. WebThinker [269] embeds a Deep Web Explorer into a think-search-draft loop, aligning via DPO with human feedback to tackle complex report-generation. WebSailor [105] is a complete post-training methodology designed to teach LLM agents sophisticated reasoning for complex web navigation and information-seeking tasks. WebWatcher [270] further extends to multimodal search, combining visual-language reasoning, tool use, and RL to outperform text-only and multimodal baselines on BrowseComp-VL and VQA benchmarks. ASearcher [117] uses large-scale asynchronous reinforcement learning with synthesized QA data, enabling long-horizon search (40+ tool calls) and outperforming prior open-source methods. MiroMind Open Deep Research (MiroMind ODR) [273] is to build a high-performance, fully open-sourced, open-collaborative deep research ecosystem — with an agent framework, model, data, and training infra all fully accessible and open.

Search from LLM internal knowledge However, these training methods that rely on external APIs face two major challenges: (1) the document quality of real-time internet document search is uncontrolled, and noisy information brings instability to the training process. (2) The API cost is too high and severely limits scalability. To enhance the efficiency, controllability and stability of training, some recent studies have used controllable simulated search engines such as LLM internal knowledge. For example, ZeroSearch [277] replaces live web retrieval with a pseudo search engine distilled from LLMs themselves, combining curriculum RL to gradually approach live-engine performance without issuing real queries. SSRL [278] takes this idea further: the agent performs entirely offline "self-search" during training, without explicit search engines, yet transfers seamlessly to online inference, where real APIs can still boost performance. Though still at an early stage, offline self-search enhances stability and scalability beyond API limits, pointing toward more self-reliant research agents.

4.1.2. Closed Source RL Methods

Despite progress in combining RAG and RL, most open source models still fail on OpenAI's BrowseComp [285], a challenging benchmark that measures the ability of AI agents to locate hard-to-find information, revealing gaps in long-horizon planning, page-grounded tool use, and cross-source verification. In contrast, recent closed source systems are markedly stronger, having shifted from mere query optimization to fully autonomous research agents that navigate the open web, synthesize information from multiple sources, and draft comprehensive reports. This is likely due to the industry's more powerful foundation models and the availability of more high-quality data. OpenAI Deep Research [111] achieves 51.5% pass@1 on BrowseComp. Other prototypes, Perplexity's DeepResearch [261], Google Gemini's DeepResearch [279], Kimi-Researcher [112], Grok AI DeepSearch [280], Doubao with Deep Think [281], combine RL-style fine-tuning with advanced tool integration and memory modules, ushering in a new era of interactive, iterative research assistants.

4.2. Code Agent

Code generation, or more broadly, software engineering, provides an ideal testbed for LLM-based agentic RL: execution semantics are explicit and verifiable, and automated signals (compilation, unit tests, and runtime traces) are readily available [286]. Early multi-agent frameworks (*e.g.*, MetaGPT, AutoGPT, AgentVerse) coordinated roles through prompting without parameter updates, showcasing the promise of modular role allocation [287, 288, 289]. Initial RL for code, such as CodeRL, incorporated execution-based reward modeling and actor–critic training [290], catalyzing a wave of studies that exploit execution feedback to guide

policy updates. Table 5 presents the majority of works studied in this section. We structure the literature along increasing *task complexity*, progressing from *code generation* (Section 4.2.1) to *code refinement* (Section 4.2.2) and *software engineering* (Section 4.2.3).

4.2.1. RL for Code Generation

Early research focused on relatively simple, single-round code generation (*e.g.*, completing a function or or solving a coding challenge in one go), which lays the foundation for subsequent large-scale software engineering.

Outcome reward RL. Methods in this class optimize directly for final correctness, typically measured by pass@k or unit-test success. AceCoder [291] introduces a data-efficient RLHF pipeline for code generation, constructing large-scale preference pairs from existing code fragments to train a reward model via Bradley–Terry loss, which then guides RFT on the synthesized dataset. Beyond early actor–critic formulations, recent open-source efforts scale outcome-based RL on large pre-trained code models. DeepCoder-14B [292] stabilizes GRPO training via iterative context lengthening and DAPO-inspired filtering, and employs a sparse Outcome Reward Model (ORM) to prevent reward hacking on curated coding data. RLTF employs an online RL loop that uses unit test results as multi-granularity reward signals, from coarse pass/fail outcomes to fine-grained fault localization, directly guiding code refinement [293]. CURE formalizes coder–tester co-evolution: a tester generates or evolves unit tests while a coder iteratively patches code; a reward-precision objective mitigates low-quality test effects during joint training [294]. Absolute Zero applies self-play RL without human data. It generates coding tasks for itself and uses execution outcomes as verifiable rewards to bootstrap reasoning ability [163]. Re:Form [295] leverages formal language-based reasoning with RL and automated verification to reduce human priors, enabling reliable program synthesis and surpassing strong baselines on formal verification tasks. In [296], authors propose a two-stage training pipeline: first fine-tuning for a high-correctness baseline, then perform efficiency-driven online RL optimization.

Process reward RL. To mitigate sparsity and credit assignment, several works design process-level supervision by integrating compilation and execution feedback. StepCoder [297] decomposes compilation and execution into step-level signals for shaping; Process Supervision-Guided Policy Optimization (PSGPO) [41] leverages intermediate error traces and process annotations for dense rewards; and CodeBoost [298] mines raw repositories to unify heterogeneous execution-derived signals, ranging from output correctness to error-message quality, under a single PPO framework. Further, PRLCoder [299] introduces process-supervised RL by constructing reward models that score each partial snippet: a teacher model mutates lines of reference solutions and assigns positive/negative signals based on compiler and test feedback. This fine-grained supervision yields faster convergence and +10.5% pass-rate improvements over the base model, illustrating how dense shaping at the line-level can guide code synthesis more effectively than outcome-only signals. o1-Coder [300] combines RL with Monte Carlo Tree Search, where the policy learns from exploration guided by test case rewards and gradually improves from pseudocode to executable code. Posterior-GRPO [301] rewards intermediate reasoning but gates credit by final test success to prevent speculative reward exploitation; Policy Filtration for RLHF [39] improves reward-correctness alignment by filtering low-confidence pairs before policy updates. Scaling preference supervision beyond costly human annotation has proven effective as well. CodeFavor [302] constructs CodePrefBench from code evolution histories, covering correctness, efficiency, security, and style to improve preference modeling and alignment. Focused-DPO [303] adapts preference-based RL by weighting preference optimization on error-prone regions of the code, making feedback more targeted and improving robustness across benchmarks. [304] studies how RL-trained small-scale agents surpass large-scale prompt-based models in MLE environments via duration-aware gradient updates

in a distributed asynchronous RL.

4.2.2. RL for Iterative Code Refinement

A second line of research targets more complex coding tasks that require debugging and iterative refinement. In these scenarios, an agent may need multiple attempts to improve solutions, using feedback from human requirements or failed test results, which is closer to real-world tasks.

Outcome reward RL. A representative line treats the entire refinement loop as a trajectory while optimizing for final task success. RLEF [305] (Reinforcement Learning from Execution Feedback) grounds correction loops in real error messages as context while optimizing for ultimate pass rates; this reduces the number of attempts needed and improves competitive-programming performance relative to single-shot baselines. μCode [306] jointly trains a generator and a learned verifier under single-step reward feedback, showing that verifier-guided outcome rewards can outperform purely execution-feedback baselines. R1-Code-Interpreter [307] harnesses multi-turn supervised fine-tuning and reinforcement learning to train LLMs to decide when and how to invoke a code interpreter during step-by-step reasoning.

Process reward RL. Process-supervised approaches explicitly guide *how* the model debugs. IterPref [308] constructs localized preference pairs from iterative debugging traces and applies targeted preference optimization to penalize faulty regions, improving correction accuracy with minimal collateral updates. LeDex [309] couples explanation-driven diagnosis with self-repair: it automatically curates explanation–refinement trajectories and applies dense, continuous rewards to jointly optimize explanation quality and code correctness via PPO, yielding consistent pass@1 gains over SFT-only coders. Beyond explanation-driven shaping, some works like CTRL [310] explicitly train separate critic models to evaluate each attempted refinement and provide gradient signals to the policy, though at the cost of added inference overhead. ReVeal [311] extends process-level refinement into a self-evolving agent that autonomously generates tests and learns from per-turn rewards to enhance reasoning and recovery from errors.

4.2.3. RL for Automated Software Engineering

Outcome reward RL. End-to-end training in realistic environments demonstrates that sparse—but validated—success signals can scale. DeepSWE performs large-scale RL on software engineering missions using verified task completion as the sole reward, achieving leading open-source results on SWE-bench–style evaluations [312]. SWE-RL extracts rule-based, outcome-oriented signals from GitHub commit histories, enabling training on authentic improvement patterns and generalization to unseen bug-fixing tasks [313]. Satori-SWE introduces an evolutionary RL-enabled test-time scaling method (EvoScale) that trains models to self-improve generations across iterations for sample-efficient software engineering tasks [314]. OS-R1 [317] presents a rule-based reinforcement learning framework for Linux kernel tuning, enabling efficient exploration, accurate configuration, and superior performance over heuristic methods. RLCoder frames retrieval-augmented repository-level code completion as an RL problem, using perplexity-based feedback to train a retriever to fetch helpful context without labeled data [315]. Qwen3-Coder performs large-scale execution-driven reinforcement learning on long-horizon, multi-turn interactions across 20,000 parallel environments, yielding state-of-the-art performance on benchmarks like SWE-Bench Verified [195]. In machine learning domains, ML-Agent executes multi-step pipelines (*e.g.*, automated ML), optimizing performance-based terminal rewards [316].

Process reward RL. Dense supervision over agentic trajectories improves credit assignment across many steps. From the optimization perspective, long-context, multi-turn software agents benefit from stabilized

Table 5: A summary of RL methods for code and software engineering agents.

Method	Reward	Base LLM	Resource
RL for Code Generation			
AceCoder [291]	Outcome	Qwen2.5-Coder-7B-Base/Instruct, Qwen2.5-7B-Instruct	GitHub
DeepCoder-14B [292]	Outcome	Deepseek-R1-Distilled-Qwen-14B	GitHub
RLTF [293]	Outcome	CodeGen-NL 2.7B, CodeT5	GitHub
CURE [294]	Outcome	Qwen2.5-7B/14B-Instruct, Qwen3-4B	GitHub
Absolute Zero [163]	Outcome	Qwen2.5-7B/14B, Qwen2.5-Coder-3B/7B/14B, Llama-3.1-8B	GitHub
StepCoder [297]	Process	DeepSeek-Coder-Instruct-6.7B	GitHub
PSGPO [41]	Process	-	-
CodeBoost [298]	Process	Qwen2.5-Coder-7B-Instruct, Llama-3.1-8B-Instruct, Seed-Coder-8B-Instruct, Yi-Coder-9B-Chat	GitHub
PRLCoder [299]	Process	CodeT5+, Unixcoder, T5-base	-
o1-Coder [300]	Process	DeepSeek-1.3B-Instruct	GitHub
Posterior-GRPO [301]	Process	Qwen2.5-Coder-3B/7B-Base, Qwen2.5-Math-7B	-
Policy Filtration for RLHF [39]	Process	DeepSeek-Coder-6.7B, Qwen1.5-7B	GitHub
CodeFavor [302]	Process	Mistral-NeMo-12B-Instruct, Gemma-2-9B-Instruct, Llama-3-8B-Instruct, Mistral-7B-Instruct-v0.3,	GitHub
Focused-DPO [303]	Process	DeepSeek-Coder-6.7B-Base/Instruct, Magicoder-S-DS-6.7B, Qwen2.5-Coder-7B-Instruct	-
Re:Form [295]	Outcome	Qwen-2.5, 0.5B-14B	GitHub
Qwen Team [296]	Outcome	Qwen-2.5-Coder-Instruct-7B/32B	-
RL for Iterative Code Refinement			
RLEF [305]	Outcome	Llama-3.0-8B-Instruct, Llama-3.1-8B/70B-Instruct	-
μCode [306]	Outcome	Llama-3.2-1B/8B-Instruct	GitHub
R1-Code-Interpreter [307]	Outcome	Qwen2.5-7B/14B-Instruct-1M, Qwen2.5-3B-Instruct	GitHub
IterPref [308]	Process	Deepseek-Coder-7B-Instruct, Qwen2.5-Coder-7B, StarCoder2-15B	-
LeDex [309]	Process	StarCoder-15B, CodeLlama-7B/13B	-
CTRL [310]	Process	Qwen2.5-Coder-7B/14B/32B-Instruct	GitHub
ReVeal [311]	Process	DAPO-Qwen-32B	-
RL for Automated Software Engineering (SWE)			
DeepSWE [312]	Outcome	Qwen3-32B	GitHub
SWE-RL [313]	Outcome	Llama-3.3-70B-Instruct	GitHub
Satori-SWE [314]	Outcome	Qwen-2.5-Math-7B	GitHub
RLCoder [315]	Outcome	CodeLlama7B, StartCoder-7B, StarCoder2-7B, DeepSeekCoder-1B/7B	GitHub
Qwen3-Coder [195]	Outcome	-	GitHub
ML-Agent [316]	Outcome	Qwen2.5-7B-Base/Instruct, DeepSeek-R1-Distill-Qwen-7B	GitHub
OS-R1 [317]	Outcome	Qwen2.5-3B/7B-Instruct	GitHub
Golubev et al. [318]	Process	Qwen2.5-72B-Instruct	-
SWEET-RL [159]	Process	Llama-3.1-8B/70B-Instruct	GitHub

policy-gradient variants; e.g., Decoupled Clip and Dynamic sAmpling Policy Optimization (DAPO) improves training stability and performance on SWE-bench Verified through multi-turn code generation and debugging interactions, leveraging long-context feedback [318]. SWEET-RL trains multi-turn agents on ColBench (backend and frontend tasks), leveraging privileged information during RL to reduce exploration noise and improve long-horizon generalization [159].

Remark on closed-source systems. Commercial systems such as OpenAI's Codex and Anthropic's Claude Code have emphasized preference-aligned fine-tuning and reinforcement learning to improve usefulness and safety in code generation and editing workflows [319, 320]. While concrete training details are limited publicly, these systems underscore the growing role of RL in aligning agentic behavior with developer-centric objectives in practical IDE and terminal environments.

4.3. Mathematical Agent

Mathematical reasoning is widely regarded as a gold standard for assessing LLM agents' reasoning ability, owing to its symbolic abstraction, logical consistency, and long-horizon deductive demands. We structure the research efforts around two complementary paradigms: *informal reasoning* (Section 4.3.1), which operates without formal verification support and includes natural-language reasoning and programming-language tool use; and *formal reasoning* (Section 4.3.2), which relies on rigorously specified formal languages and proof assistants.

We note that RLVR methods such as DAPO [47], GRPO [321], and GRESO [55] have consistently played a substantial role in recent enhancements of mathematical reasoning in LLMs. However, given their broader relevance across reasoning tasks, we discuss them in Section 2.7, instead of elaborating here.

4.3.1. RL for Informal Mathematical Reasoning

Informal mathematics essentially refers to reasoning and expression in natural language. Such reasoning may incorporate symbols or function names, but no finite and explicit set of logical rules defines their syntactic validity, and no formal semantics precisely determines their interpretation and meaning [322, 323].

While informal mathematical reasoning relaxes strict rigor at the detail level, it affords greater expressive flexibility and better captures the high-level structure of arguments. This makes it particularly suited for a variety of math tasks such as mathematical word problem solving, equation manipulation, and symbolic computation [100, 324]. Although general-purpose programming languages are symbolic, they lack the rigor and formal semantics of proof-assistant languages, and are therefore regarded as informal when applied to mathematical reasoning [322], typically through tool invocation of executors such as Python with numerical or symbolic libraries.

Outcome reward RL. Outcome-only methods define rewards solely by final numerical or symbolic correctness (e.g. algebraic equations) during RL. Empirically, such training often leads to emergent agentic behaviors, including adaptive tool use interleaved with natural language reasoning. ARTIST [100] introduces a framework for tool-integrated agentic reasoning, interleaving tool invocations, e.g. code execution, directly within the reasoning chain. Trained with outcome-only rewards, it achieves strong performance and observes emergent agentic behaviors, including self-reflection, and context-aware CoT, which further shows that by integrating dynamic tool use with RL, agentic tool-integrated reasoning could learn optimal strategies for interacting with environments, highlighting the potential of RL to internalize tool-integrated reasoning strategies in LLMs. Similarly, ToRL [101] improves performance by exploiting the scaling of tool-integrated reasoning RL and encouraging code execution behaviour, and experiments show emergent cognitive behaviors, such as adaptive tool-use, self-correction based on tool feedback, and adaptive computational reasoning. ZeroTIR [324] investigates the scaling law of RL from outcome-based rewards for Tool-Integrated Reasoning with Python code execution settings, revealing a strong correlation between training computational effort and the spontaneous code execution frequency, the average response length, and the final task accuracy, which corroborates the empirical emergence of tool-integrated reasoning strategies. TTRL [165] leverages

majority voting to estimate rewards, enabling training on unlabeled data. Fine-tuned on these majority-vote rewards, it not only surpasses the base model's maj@n accuracy but also achieves an empirical performance curve and upper bound that, surprisingly, closely approach those of direct RL training with labeled test answers on MATH-500, underscoring its practical value and potential. However, RENT [325] suggests that majority voting is short on generalization, it applies only to questions with deterministic answers, and will not work on free-response outputs. To address this limitation, it extends the entropy minimization idea [326] to RL, using the token-level average negative entropy as a reward to guide learning, achieving improvements on an extensive suite of benchmarks including math problem solving, suggesting that confidence-based reward shaping can serve as a path toward continual improvement. Alternatively, Satori [314] proposes Chain-of-Action-Thought (COAT), a variant of CoT that explicitly integrates action choices, and modularizes reasoning into 3-fold meta-actions, including continuation, reflection, and exploration of alternatives, and internalizes this behavior via RL with outcome-only rewards. In particular, 1-shot RLVR [327] studies data efficiency of outcome-only RL with verifier signals. Surprisingly, they found that RL with only 1 example performs close to using a 1.2k-example dataset, and with 2 examples comes close to using the 7.5k MATH training dataset. They also highlight an intriguing phenomenon, named post-saturation generalization, that test accuracy continues to improve even after the training accuracy on the single example approaches 100%. In addition to correctness, hallucination remains a major challenge in informal mathematical reasoning, motivating methods that explicitly promote trustworthiness. For instance, Kirchner et al. [328] propose a game-theoretic training algorithm that jointly optimizes for both correctness and legibility. Inspired by Prover-Verifier Games [329], the method alternates between training a small verifier that predicts solution correctness, a "helpful" prover that generates solutions accepted by the verifier, and a "sneaky" prover that aims to fool it. Empirically, this increases the helpful prover accuracy, verifier robustness and legibility (measured by human accuracy in time-constraint verification tasks). This result suggests that verifier-guided legibility optimization can enhance the interpretability and trustworthiness of LLM-generated informal reasoning. Recent rStar2-Agent [115] is a 14B-parameter math reasoning model trained with agentic reinforcement learning using a high-throughput Python execution environment, a novel GRPO-RoC algorithm to resample on correct rollouts amid tool-noise, and a multi-stage training recipe—achieving state-of-the-art results in just 510 RL steps, achieving average pass@1 scores of 80.6% on AIME24 and 69.8% on AIME25.

Process reward RL. Process-aware methods leverage intermediate evaluators (e.g. unit tests, assertions, sub-task checks) to provide denser feedback, shaping credit assignment and improving tool-integrated reasoning (TIR). START [330] guides TIR by injecting handcrafted hint text into Long CoT traces, typically after conjunction words or before the CoT stop token, to encourage code executor calls during inference. This enables test-time scaling that improves reasoning accuracy. The collected trajectories are then used to fine-tune the model, internalizing the tool-invocation behavior. LADDER [331] introduces a training-time framework where an LLM recursively generates and solves progressively simpler variants of a complex problem, using verifiable reward signals to guide a difficulty-based curriculum, and achieves substantial improvements in mathematical reasoning. An additional test-time RL step (TTRL) further enhances performance. The authors suggest that this approach of self-generated curriculum learning with verifiable feedback may generalize beyond informal mathematical tasks to any domain with reliable automatic verification. To improve performance on complex problems, SWiRL [332] synthesizes step-wise tool use reasoning data by iteratively decomposing solutions, and then adopts a preference-based step-wise RL approach to fine-tune the base model on the multi-step trajectories. While many of these approaches exploits inference-time interventions, they often suffer from generalization limitations due to their reliance on manually designed logical structures. To overcome this, RLoT [333] instead trains a lightweight navigator agent model with RL to adaptively enhance reasoning, showing improved generalization across diverse tasks.

While informal approaches excel at word problems and symbolic computations, they struggle to extend effectively to advanced mathematical tasks such as automated theorem proving. This limitation arises from two fundamental challenges: evaluation difficulty, which demands machine-verifiable feedback unavailable to informal methods, and scarcity of high-quality formal proof data. [322, 323]

4.3.2. RL for Formal Mathematical Reasoning

Formal mathematical reasoning refers to reasoning carried out in a formal language with precisely defined syntax and semantics, yielding proof objects that are mechanically checkable by a verifier. This paradigm is particularly suited for advanced tasks such as automated theorem proving (ATP) [334], where an agent, given a statement (theorem, lemma, or proposition), must construct a proof object that the verifier accepts, thereby ensuring machine-verifiable correctness. From a reinforcement learning perspective, formal theorem proving is commonly modeled as a Markov Decision Process (MDP): proof states transition via the application of tactics[1], each of which is treated as a discrete action in RL-based proof search. [335]. Under this formulation, formal theorem proving can be cast as a search problem over a vast, discrete, and parameterized action space.

Formal proofs are verified by proof assistants such as Lean, Isabelle, Coq, and HOL Light. These systems, often referred to as Interactive Theorem Provers (ITPs), deterministically accept or reject proof objects producing binary pass/fail signals as the primary reward for RL training, while some works also explore leveraging error messages as auxiliary signals [336, 337].

Outcome reward RL. The outcome-only paradigm was demonstrated at scale in 2024 with DeepSeek-Prover-v1.5 [334], which releases an end-to-end RL pipeline in Lean based solely on binary verifier feedback, resulting in significant improvements in proof success on benchmarks like miniF2F [338] and ProofNet [339]. The authors propose a variant of MCTS, i.e. RMaxTS, that incorporates intrinsic rewards for discovering novel tactic states to encourage diversity of proof exploration during inference-time search and mitigate the sparse-reward issue. Building on this direction, Leanabell-Prover [340] scales up DeepSeek-Prover-v1.5 by aggregating an expansive hybrid dataset of statement-proof pairs and informal reasoning sketches from multiple sources and pipelines such as Mathlib4 [341], LeanWorkbook [342], NuminaMath [343], STP [344], etc., covering well over 20 mathematical domains. This broad coverage mitigates the scarcity of aligned informal-to-formal (NL to Lean4) training examples, which are crucial for bridging natural-language reasoning and formal proof generation. At the same time, Kimina-Prover [345] Preview further emphasizes the critical challenge of aligning informal and formal reasoning. It implements a structured "formal reasoning pattern," where natural-language reasoning and Lean 4 code snippets are interleaved within thinking blocks. To reinforce this alignment, the output is constrained—to include at least one tactic block and to reuse no less than 60% of the Lean 4 snippets in the final proof, ensuring close correspondence between internal reasoning and formal output. A recent work, Seed-Prover [346], integrates multiple techniques. It first adopts a lemma-centered proof paradigm, which enables systematic problem decomposition, cross-trajectory lemma reuse, and explicit progress tracking. It then enriches RL training with a diverse prompting strategy that randomly incorporates both informal and formal proofs, successful and failed lemmas, and Lean compiler feedback, thereby enhancing adaptability to varied inputs. At inference, it employs a conjecture–prover pipeline that interleaves proving conjectures into lemmas and generating new conjectures from the evolving lemma pool, substantially improving its capacity to tackle difficult problems. Complementarily, the accompanying

[1] In Lean-style Interactive Theorem Provers (ITPs), a tactic is a command or small script that instructs the system to refine the current proof goal, with the resulting proof term checked by the ITP kernel for correctness.

Seed-Geometry system extends formal reasoning to geometry, providing state-of-the-art performance on Olympiad benchmarks. Together, these efforts demonstrate that sparse but explicit reward signals can yield nontrivial gains, particularly when paired with effective exploration strategies.

Process reward RL. To improve credit assignment and reduce wasted exploration, several works extend the outcome-only paradigm with denser, step-level signals. DeepSeek-Prover-v2 [347] designs a dual-model pipeline to unify both informal (natural-language) and formal (Lean4) mathematical reasoning to reinforce the proving reasoning ability. It introduces subgoal decomposition, where a prover model solves recursively decomposed subgoals and receives binary Lean feedback at the subgoal level, effectively providing denser supervision and improving both accuracy and interpretability. Following this dual-role collaborative mindset, ProofNet++[336] implements a neuro-symbolic RL framework featuring a Symbolic Reasoning Interface, which maps LLM-generated reasoning into formal proof trees, and a Formal Verification Engine, which verifies these proofs with Lean or HOL Light and routes error feedback back to the LLM for self-correction. Leanabell-Prover-v2 [337] integrates verifier messages into reinforcement updates within a long CoT framework, enabling explicit verifier-aware self-monitoring that stabilizes tactic generation and reduces repeated failure patterns.

Hybrid reward RL. Although both outcome-only and process-aware reward paradigms have demonstrated encouraging advances, the scarcity of high-quality theorem-proving data further amplifies the challenges of reinforcement learning under sparse rewards as well as the design of step-level preference signals [348, 349, 344]. To mitigate these limitations, a prominent line of work adopts expert iteration (ExIt) [350], a framework that combines search with policy learning. This paradigm provides an alternative to outcome-only or process-aware RL, alleviating data scarcity by producing high-quality supervised trajectories. Instead of directly optimizing against sparse verifier signals, ExIt performs *search-guided data augmentation*: valid proof trajectories discovered by search and checked by a verifier are reused as expert demonstrations in an imitation-learning loop. It usually employs a two-role system: the *expert* collects valid and progressive trajectories via MCTS under outcome-only verifier feedback, while the *apprentice* trains a policy on these process-level trajectories and then shares the improved policy back with the expert, thereby bootstrapping subsequent rounds of search and accelerating convergence. Polu and Sutskever [351] introduces ExIt into formal theorem proving, demonstrating that search-generated expert data can bootstrap models toward tackling complex multi-step proving challenges. Later works adapt this design to Lean and other ITPs.

When applying to formal theorem proving, naive tree search methods often face severe search space explosion when navigating the vast parameterized tactic space. To mitigate this, InternLM2.5-StepProver [352] introduces a preference-based critic model, trained with RLHF-style optimization, to guide expert search, effectively providing a curriculum that directs exploration toward problems of suitable difficulty. Lean-STaR [353] further enhances ExIt by integrating Self-Taught Reasoner (STaR) [354]. It first trains a thought-augmented tactic predictor on synthesized *(proof state, generated thought, ground-truth tactic)* triples. Then, in the expert-iteration loop, the model produces trajectories that interleave thoughts with tactics; trajectories with tactics successfully validated by Lean are retained and reused for imitation learning. Empirically, the inclusion of thoughts increases the diversity of exploration in the sample-based proof search.

A recent work, STP [344], points out that solely relying on expert iteration will quickly plateau due to the sparse positive rewards. To address this, it extends the conjecturer–prover self-play idea from Minimo [355] to practical formal languages (Lean/Isabelle) with an open-ended action space and starts from a pretrained model. STP instantiates a dual-role loop in which a conjecturer proposes statements

that are barely provable by the current prover, and a prover is trained with standard expert iteration; this generates an adaptive curriculum and alleviates sparse training signals. Empirically, STP reports large gains on LeanWorkbook [342] and reports competitive results among whole-proof generation methods on miniF2F [338] and ProofNet [339].

Table 6: A summary of RL methods for mathematical reasoning agents.

Method	Reward	Resources
RL for Informal Mathematical Reasoning		
ARTIST [100]	Outcome	-
ToRL [101]	Outcome	GitHub HuggingFace
ZeroTIR [324]	Outcome	GitHub HuggingFace
TTRL [165]	Outcome	GitHub
RENT [325]	Outcome	GitHub Website
Satori [314]	Outcome	GitHub HuggingFace Website
1-shot RLVR [327]	Outcome	GitHub HuggingFace
Prover-Verifier Games (legibility) [328]	Outcome	-
rStar2-Agent [115]	Outcome	GitHub
START [330]	Process	-
LADDER [331]	Process	-
SWiRL [332]	Process	-
RLoT [333]	Process	GitHub
RL for Formal Mathematical Reasoning		
DeepSeek-Prover-v1.5 [334]	Outcome	GitHub HuggingFace
Leanabell-Prover [340]	Outcome	GitHub HuggingFace
Kimina-Prover (Preview) [345]	Outcome	GitHub HuggingFace
Seed-Prover [346]	Outcome	GitHub
DeepSeek-Prover-v2 [347]	Process	GitHub HuggingFace
ProofNet++ [336]	Process	-
Leanabell-Prover-v2 [337]	Process	GitHub
InternLM2.5-StepProver [352]	Hybrid	GitHub
Lean-STaR [353]	Hybrid	GitHub HuggingFace Website
STP [344]	Hybrid	GitHub HuggingFace

4.4. GUI Agent

GUI agents have progressed through distinct training paradigms. Early systems used pre-trained vision–language models (VLMs) in a pure zero-shot fashion, mapping screenshots and prompts directly to single-step actions. Later, SFT on static (screen, action) trajectories improved grounding and reasoning, but was limited by scarce human operation traces. Reinforcement fine-tuning (RFT) reframes GUI interaction as sequential decision-making, allowing agents to learn via trial-and-error with sparse or shaped rewards, and has advanced from simple single-task settings to complex, real-world, long-horizon scenarios. Table 7 presents the majority of works studied in this section.

4.4.1. RL-free Methods

Vanilla VLM-based GUI Agents Early GUI agents directly leveraged pre-trained Vision–Language Models (VLMs) in a pure zero-shot manner, mapping screenshots and prompts to single-step actions without any task-specific fine-tuning. Representative systems include MM-Navigator [356], SeeAct [357], and TRISHUL [358], which differ in interface domains or parsing strategies but share the same reliance on off-the-shelf VLMs. While showcasing the generality of foundation models, these approaches suffer from limited grounding accuracy and reliability, restricting their applicability to complex tasks [359, 360].

Supervised Fine-Tuning (SFT) with Static Trajectory Data The SFT paradigm adapts pre-trained vision–language models to GUI tasks by minimizing cross-entropy loss on offline (screen, action) pairs, without online interaction. InfiGUIAgent [361] employs a two-stage pipeline that first improves grounding and then incorporates hierarchical and reflective reasoning. UI-AGILE [362] enhances supervised fine-tuning by incorporating continuous rewards, simplified reasoning, and cropping-based resampling, while further proposing a decomposed grounding mechanism for handling high-resolution displays. TongUI [363] instead emphasizes data scale, constructing the 143K-trajectory GUI-Net from multimodal web tutorials to enhance generalization. While differing in focus, these approaches all face the limitation of scarce human operation traces.

4.4.2. RL in Static GUI Environments

In static settings, reinforcement learning is applied on pre-collected datasets with deterministic execution traces, using rule-based criteria for outcome evaluation in the absence of live environment interactions. GUI-R1 [364] adopts an R1-style reinforcement fine-tuning pipeline over a unified action schema, using simple format and correctness rewards to improve step-level action prediction with modest data. UI-R1 [365] applies group-relative policy optimization to stabilize policy updates and improve exact parameter matching through a compact action interface and reward shaping for action-type and argument accuracy. InFiGUI-R1 [366] introduces a two-stage training paradigm that first distills spatial reasoning to enhance grounding, followed by reinforcement learning with sub-goal supervision and recovery mechanisms to improve long-horizon reasoning. AgentCPM-GUI [367] combines grounding-aware pre-training, supervised imitation, and GRPO-based reinforcement fine-tuning with a concise JSON action space, reducing decoding overhead while improving robustness on long-horizon sequences. UI-Venus [368] is a multimodal screenshot-based UI agent fine-tuned via RFT with custom reward functions and a self-evolving trajectory framework, achieving new state-of-the-art in both UI grounding and navigation.

4.4.3. RL in Interactive GUI Environments

In interactive settings, reinforcement learning agents are optimized through online rollouts in dynamic environments, requiring robustness to stochastic transitions and long-horizon dependencies. WebAgent-R1 [104] conducts end-to-end multi-turn reinforcement learning with asynchronous trajectory generation and group-wise advantages, improving success on diverse web tasks. Vattikonda et al. [369] study reinforcement learning for web agents under realistic page dynamics and large action spaces, highlighting challenges in credit assignment and safe exploration. UI-TARS [370] integrates pre-training for GUI understanding with reinforcement learning for native desktop control, coupling milestone tracking and reflection to enhance long-horizon execution. DiGiRL [371] introduces an offline-to-online reinforcement learning pipeline on real Android devices, combining advantage-weighted updates, doubly robust advantage estimation, and instruction-level curricula to cope with non-stationarity. ZeroGUI [372] automates task generation and reward estimation with a vision-language evaluator, then applies two-stage online reinforcement learning

Table 7: A summary of methods for GUI agents, categorized by training paradigm and environment complexity.

Method	Paradigm	Environment	Resource Link
RL-free GUI Agents			
MM-Navigator [356]	Vanilla VLM	-	GitHub
SeeAct [357]	Vanilla VLM	-	GitHub
TRISHUL [358]	Vanilla VLM	-	-
InfiGUIAgent [361]	SFT	Static	GitHub HuggingFace Website
UI-AGILE [362]	SFT	Interactive	GitHub HuggingFace
TongUI [363]	SFT	Static	GitHub HuggingFace Website
RL-based GUI Agents			
GUI-R1 [364]	RL	Static	GitHub HuggingFace
UI-R1 [365]	RL	Static	GitHub HuggingFace
InFiGUI-R1 [366]	RL	Static	GitHub HuggingFace
AgentCPM [367]	RL	Static	GitHub HuggingFace
UI-Venus [368]	RL	Static	GitHub
WebAgent-R1 [104]	RL	Interactive	-
Vattikonda et al. [369]	RL	Interactive	-
UI-TARS [370]	RL	Interactive	GitHub HuggingFace Website
UI-TARS-2 [375]	RL	Interactive	GitHub Website
DiGiRL [371]	RL	Interactive	GitHub HuggingFace Website
ZeroGUI [372]	RL	Interactive	GitHub
MobileGUI-RL [373]	RL	Interactive	-
ComputerRL [374]	RL	Interactive	-

(training on generated tasks followed by test-time adaptation) to reduce human supervision. MobileGUI-RL [373] scales training on Android virtual devices with trajectory-aware GRPO, a decaying efficiency reward, and curriculum filtering, improving execution efficiency and generalization while keeping the system practical for large rollout volumes. ComputerRL [374] introduces an API-GUI hybrid interaction paradigm paired with a massively parallel, fully asynchronous RL infrastructure and the novel Entropulse training strategy—alternating RL with supervised fine-tuning—to empower GUI-based agents to operate efficiently and scalably in desktop environments.

4.5. RL in Vision Agents

RL has been applied to a wide range of vision tasks (including, but not limited to, image / video / 3D perception and generation). Since the number of related papers is substantial, this section does not aim to provide an exhaustive overview; for a more comprehensive survey on RL for various vision tasks, we refer readers to two dedicated surveys in vision [212, 211].

Image Tasks. The success of DeepSeek-R1 [32] has sparked widespread interest in applying RL to incentivize long-form reasoning behavior, encouraging LVLMs to produce extended CoT sequences that improve visual perception and understanding [208]. This research trajectory has evolved from early work that simply adapted R1-style objectives to the vision domain—aimed primarily at enhancing passive perception [220, 221, 222, 214, 215, 225, 226, 376]—toward the now-popular paradigm of active perception, or "think-

ing with images" [210]. The key transition lies in moving from text-only CoT that references an image once, to interactive, visually grounded reasoning, achieved through (i) grounding [377, 232, 233, 234, 235, 236], (ii) agentic tool use [239, 240, 241, 242, 219, 98], and (iii) visual imagination via sketching or generation [243, 246, 247]. Beyond text-only outputs, many vision tasks—such as scene understanding—require structured predictions like bounding boxes, masks, and segmentation maps. To begin with, Visual-RFT [219] uses IoU with confidence as a verifiable reward for bounding-box outputs, while Vision-R1 [222] incorporates precision and recall as localization rewards. Extending this idea, [378] applies GRPO to segmentation tasks, combining soft and strict rewards with bounding-box IoU and L1 loss, and point-wise L1 distance. VLM-R1 [214] employs mean Average Precision (mAP) as a reward to explicitly incentivize detection and localization capabilities in LVLMs. Finally, R1-SGG [379] introduces three variants of GRPO rewards for scene-graph matching—ranging from hard rewards based on text matching and IoU to softer rewards computed via text-embedding dot products. RL has also been widely applied to image generation, particularly through its integration with diffusion and flow models—for example, RePrompt [380], Diffusion-KTO [381], Flow-GRPO [382], and GoT-R1 [383]. Beyond diffusion-based approaches, RL has been leveraged for autoregressive image generation, where it improves coherence, fidelity, and controllability by directly optimizing task- or user-specific reward signals [384, 247, 385].

Video Tasks. Following the same spirit, numerous works have extended GRPO variants to the video domain [386, 387, 388] to enhance temporal reasoning [389, 390, 391, 392, 393]. TW-GRPO [394] introduces a token-weighted GRPO framework that emphasizes high-information tokens to generate more focused reasoning chains and employs soft, multi-choice rewards for lower-variance optimization. EgoVLM [395] combines keyframe-based rewards with direct GRPO training to produce interpretable reasoning traces tailored for egocentric video. DeepVideo-R1 reformulates the GRPO objective as a regression task [389], while VideoChat-R1 demonstrates that reinforcement finetuning (RFT) can be highly data-efficient for task-specific video reasoning improvements [390]. TinyLLaVA-Video-R1 explores scaling RL to smaller video LLMs [396], and [397] introduces infrastructure and a two-stage pipeline (CoT-SFT + RL) to support large-scale RL for long videos. Additional efforts have also extended RL for embodied video reasoning tasks [398]. A similar trend is observed in video generation, where RL is applied to improve temporal coherence, controllability, and semantic alignment. Key examples include DanceGRPO [399], GAPO [400], GRADEO [401], InfLVG [402], Phys-AR [403], VideoReward [404], TeViR [405], and InstructVideo [406].

3D Vision Tasks. RL has also been widely adopted to advance 3D understanding [407, 408, 409, 410, 411, 412] and generation [413, 414, 415]. MetaSpatial [416] introduces the first RL-based framework for 3D spatial reasoning, leveraging physics-aware constraints and rendered-image evaluations as rewards during training. Scene-R1 [417] learns to reason about 3D scenes without point-wise 3D supervision, while SpatialReasoner [418] introduces shared 3D representations that unify perception, computation, and reasoning stages. In the domain of 3D generation, RL has been applied to improve text-to-3D alignment and controllability. Notable efforts include DreamCS [419], which aligns generation with human preferences; DreamDPO [420] and DreamReward [421], which optimize 3D generation using 2D reward signals; and Nabla-R2D3 [422], which further refines 3D outputs with reinforcement-driven objectives.

4.6. RL in Embodied Agents

While traditional agents are typically developed for general-purpose vision or language tasks, extending these capabilities to embodied agents requires a comprehensive understanding of real-world visual environments and the capacity to reason across modalities. Such competencies are essential for perceiving complex physical

contexts and executing goal-directed actions conditioned on high-level instructions, forming a foundational element of agentic LLMs and MLLMs. In instruction-driven embodied scenarios, RL is often employed as a post-training strategy. A common pipeline begins with a pre-trained vision-language-action (VLA) model [423, 424, 425, 426] obtained through imitation learning under teacher-forcing supervision. This model is then embedded into an interactive agent that engages with the environment to collect reward signals. These rewards guide the iterative refinement of the policy, supporting effective exploration, improving sample efficiency, and enhancing the model's generalization capabilities across diverse real-world conditions. RL in VLA frameworks [427, 428, 429, 430] can be broadly categorized into two classes: navigation agents, which emphasize spatial reasoning and locomotion in complex environments, and manipulation agents, which focus on the precise control of physical objects under diverse and dynamic constraints.

RL in VLA Navigation Agent. For navigation agents, planning is the central capability. Reinforcement learning is employed to enhance the VLA model's ability to predict and optimize future action sequences. A common strategy [431] is to integrate traditional robotics-style RL, using step-wise directional rewards, directly into VLA-based navigation frameworks. Some approaches operate at the trajectory level. VLN-R1 [429] aligns predicted and ground-truth paths to define trajectory-level rewards, and applies GRPO, following DeepSeek-R1, to improve predictive planning. OctoNav-R1 [376] also leverages GRPO but focuses on reinforcing internal deliberation within the VLA model, promoting a thinking-before-acting paradigm that enables more anticipatory and robust navigation. S2E [432] introduces a reinforcement learning framework that augments navigation foundation models with interactivity and safety, combining video pretraining with RL to achieve superior generalization and performance on the NavBench-GS benchmark.

RL in VLA Manipulation Agent. Manipulation agents, typically involving robotic arms, require fine-grained control for executing structured tasks under diverse conditions. In this context, RL is employed to enhance the instruction-following and trajectory prediction capabilities of VLA models, especially to improve generalization across tasks and environments. RLVLA [433] and VLA-RL [428] adopt pre-trained VLMs as evaluators, using their feedback to assign trajectory-level rewards for VLA policy refinement. These methods establish an online RL framework that effectively improves manipulation performance and demonstrates favorable scaling properties. TGRPO further [434] incorporates GRPO into manipulation tasks by defining rule-based reward functions over predicted trajectories. This enables the VLA model to generalize to unseen scenarios and improves its robustness in real-world deployment. VIKI-R [435] complements this with a unified benchmark and two-stage framework for multi-agent embodied cooperation, combining Chain-of-Thought fine-tuning with multi-level RL to enable compositional coordination across diverse embodiments.

A central challenge in RL for VLA embodied agents is scaling training to real-world environments. While simulation platforms enable efficient large-scale experimentation, the sim-to-real gap remains significant, particularly in fine-grained manipulation tasks. Conducting RL directly in real-world settings is currently impractical due to the high cost and complexity of physical robot experiments. Most RL algorithms require millions of interaction steps, which demand substantial time, resources, and maintenance. As a result, developing scalable embodied RL pipelines that can bridge the gap between simulation and real-world deployment remains an open and pressing problem.

4.7. RL in Multi-Agent Systems

Large Language Model (LLM)-based Multi-agent Systems (MAS) comprise multiple autonomous agents collaborating to solve complex tasks through structured interaction, coordination, and memory management.

Table 8: A summary of reinforcement learning and evolution paradigms in LLM-based Multi-Agent Systems. "Dynamic" denotes whether the multi-agent system is task-dynamic, *i.e.*, processes different task queries with different configurations (agent count, topologies, reasoning depth, prompts, *etc*). "Train" denotes whether the method involves training the LLM backbone of agents.

Method	Dynamic	Train	RL Algorithm	Resource Link
RL-Free Multi-Agent Systems (not exhaustive)				
CAMEL [436]	✗	✗	-	GitHub HuggingFace
MetaGPT [287]	✗	✗	-	GitHub
MAD [438]	✗	✗	-	GitHub
MoA [437]	✗	✗	-	GitHub
AFlow [444]	✗	✗	-	GitHub
RL-Based Multi-Agent Training				
GPTSwarm [440]	✗	✗	policy gradient	GitHub Website
MaAS [446]	✔	✗	policy gradient	GitHub
G-Designer [447]	✔	✗	policy gradient	GitHub
Optima [448]	✗	✔	DPO	GitHub
DITS [449]	✗	✔	DPO	-
MALT [168]	✗	✔	DPO	-
MARFT [450]	✗	✔	MARFT	GitHub
ACC-Collab [160]	✗	✔	DPO	-
MAPoRL [451]	✔	✔	PPO	GitHub
MLPO [452]	✔	✔	MLPO	-
ReMA [453]	✔	✔	MAMRP	GitHub
FlowReasoner [454]	✔	✔	GRPO	GitHub
CURE [294]	✗	✔	rule-based RL	GitHub HuggingFace
MMedAgent-RL [455]	✗	✔	GRPO	-
Chain-of-Agents [456]	✔	✔	DAPO	GitHub HuggingFace
RLCCF [457]	✗	✔	GRPO	-
MAGRPO [458]	✗	✔	MAGRPO	-

Early static and hand-designed MAS such as CAMEL and MetaGPT [436, 287] explored role specialization and task decomposition, while debate-based frameworks such as MAD and MoA [437, 438] enhanced reasoning via collaborative refinement. Subsequent multi-agent research has shifted to proposing optimizable cooperative systems, which enable MAS to not only dynamically adjust coordination patterns but also directly enhance agent-level reasoning and decision-making strategies. Table 8 summarizes the main body of works discussed in this section.

RL-Free Multi-Agent Evolution In the RL-free self-evolving setting, foundation models cannot be directly optimized; instead, system evolution is driven by mechanisms such as symbolic learning [439], dynamic graph optimization [440, 441, 442], and workflow rewriting [443, 444, 445]. These methods improve the coordination and adaptability within MAS, but cannot directly update the parameters of foundation models. MALT [168] employs a heterogeneous multi-agent search tree to generate large-scale labeled trajectories, fine-tuning agents via a combination of Supervised Fine-Tuning (SFT) and Direct Preference Optimization (DPO) from both successful and failed reasoning paths.

RL-Based Multi-Agent Training MARFT [450] formalizes a reinforcement fine-tuning framework for MAS with mathematical guarantees and empirical validation. MAGRPO [458] formalizes multi-LLM cooperation as a Dec-POMDP problem and introduces a multi-agent variant of GRPO, which enables joint training of LLM agents in MAS while maintaining decentralized execution. MAPoRL [451] extends MAD by verifying debate responses and using validation outcomes as RL rewards to improve collaborative reasoning. RLCCF [457] is a self-supervised multi-agent RL framework that leverages self-consistency-weighted ensemble voting to generate pseudo-labels and collaboratively optimize individual model policies via GRPO, boosting both individual and collective reasoning accuracy. MLPO [452] introduces a hierarchical paradigm in which a central LLM leader learns, via RL, to synthesize and evaluate peer agent outputs without auxiliary value networks. ReMA [453] separates reasoning into a meta-thinking agent and an execution agent, jointly trained under aligned RL objectives with parameter sharing. FlowReasoner [454] designs query-level meta-agents optimized through RL with multi-dimensional rewards (performance, complexity, efficiency) guided by execution feedback. LERO [459] combines MARL with LLM-generated hybrid rewards and evolutionary search to improve credit assignment and partial observability handling in cooperative tasks. CURE [294] focuses on code generation, jointly training a code generator and unit tester via RL to produce richer reward signals, achieving strong generalization across diverse coding benchmarks. MMedAgent-RL [455] introduces a reinforcement learning-based multi-agent framework for medical VQA, where dynamically coordinated general practitioners and specialists collaboratively reason with curriculum-guided learning, significantly outperforming existing Med-LVLMs and achieving more human-like diagnostic behavior. Chain-of-Agents (COA) [456] is an end-to-end paradigm where a single LLM simulates multi-agent collaboration by dynamically orchestrating role-playing and tool-using agents; this is achieved through multi-agent distillation (converting trajectories from state-of-the-art multi-agent systems into training data) and agentic reinforcement learning with carefully designed reward functions, resulting in Agent Foundation Models (AFMs). SPIRAL [460] presents a fully online, multi-turn, multi-agent self-play reinforcement learning framework for LLMs in zero-sum games, employing a shared policy with role-conditioned advantage estimation (RAE) to stabilize learning, and demonstrates that gameplay fosters transferable reasoning skills that significantly improve mathematical and general reasoning benchmarks.

4.8. Other Tasks

TextGame. ARIA [461] compresses the sprawling action space via intention-driven reward aggregation, reducing sparsity and variance. GiGPO [119] enhances temporal credit assignment through hierarchical grouping without added computational burden. RAGEN [56] ensures stable multi-turn learning by filtering trajectories and stabilizing gradients, while advocating for reasoning-aware rewards. SPA-RL [120] decomposes delayed rewards into per-step signals, improving performance and grounding accuracy. Trinity-RFT [462] provides a unified, modular framework for reinforcement fine-tuning across tasks—including text games—enabling flexible, efficient, and scalable experimentation with diverse RL modes and data pipelines.

Table. SkyRL-SQL [463] introduces a data-efficient, multi-turn RL pipeline for Text-to-SQL, enabling LLM agents to interactively probe databases, refine, and verify SQL queries. With just 653 training examples, the SkyRL-SQL-7B model surpasses both GPT-4o and o4-mini on SQL generation benchmarks. MSRL [464] introduces multimodal structured reinforcement learning with multi-granularity rewards to overcome the SFT plateau in chart-to-code generation, achieving state-of-the-art performance on chart understanding benchmarks

Time Series. Time-R1 [465] enhances moderate-sized LLMs with comprehensive temporal reasoning abilities through a progressive reinforcement learning curriculum and a dynamic rule-based reward system.

TimeMaster [466] trains time-series MLLMs that combine SFT with GRPO to enable structured, interpretable temporal reasoning over visualized time-series inputs.

General QA. Agent models [467] internalize chain-of-action generation to enable autonomous and efficient decision-making through a combination of supervised fine-tuning and reinforcement learning. L-Zero [468] enables large language models to become general-purpose agents through a scalable, end-to-end reinforcement learning pipeline utilizing a low-cost, extensible, and sandboxed concurrent agent worker pool.

Social. Sotopia-RL [469] refines coarse episode-level rewards into utterance-level, multi-dimensional signals to enable efficient and stable RL training for socially intelligent LLMs under partial observability and multi-faceted objectives. [470] introduces an Adaptive Mode Learning (AML) framework with the Adaptive Mode Policy Optimization (AMPO) algorithm, which uses reinforcement learning to dynamically switch between multi-granular reasoning modes in social intelligence tasks, achieving higher accuracy and shorter reasoning chains than fixed-depth RL methods like GRPO.

5. Enviroment and Frameworks

5.1. Environment Simulator

In agentic reinforcement learning, the environment is the world with which the agent interacts, receiving sensory input (observations) and enacting choices (actions) through its actuators. The environment, in turn, responds to the agent's actions by transitioning to a new state and providing a reward signal. With the rise of the LLM Agent paradigm, many works have proposed environments for training specific tasks. Table 9 provides an overview of the key environments examined in this section.

Table 9: A summary of environments and benchmarks for agentic reinforcement learning, categorized by agent capability, task domain, and modality. The agent capabilities are denoted by: ① Reasoning, ② Planning, ③ Tool Use, ④ Memory, ⑤ Collaboration, ⑥ Self-Improve.

Environment / Benchmark	Agent Capability	Task Domain	Modality	Resource Link
LMRL-Gym [471]	①, ④	Interaction	Text	GitHub
ALFWorld [472]	②, ①	Embodied, Text Games	Text	GitHub Website
TextWorld [473]	②, ①	Text Games	Text	GitHub
ScienceWorld [474]	①, ②	Embodied, Science	Text	GitHub Website
AgentGym [475]	①, ④	Text Games	Text	GitHub Website
Agentbench [476]	①	General	Text, Visual	GitHub
InternBootcamp [477]	①	General, Coding, Logic	Text	GitHub
LoCoMo [478]	④	Interaction	Text	GitHub Website
MemoryAgentBench [479]	④	Interaction	Text	GitHub
WebShop [480]	②, ③	Web	Text	GitHub Website
Mind2Web [481]	②, ③	Web	Text, Visual	GitHub Website
WebArena [482]	②, ③	Web	Text	GitHub Website
VisualwebArena [483]	①, ②, ③	Web	Text, Visual	GitHub Website
AppBench [484]	②, ③	App	Text	GitHub
AppWorld [485]	②, ③	App	Text	GitHub Website
AndroidWorld [486]	②, ③	GUI, App	Text, Visual	GitHub
OSWorld [487]	②, ③	GUI, OS	Text, Visual	GitHub Website
WindowsAgentArena [488]	②			
Debug-Gym [489]	①, ③	SWE	Text	GitHub Website
MLE-Dojo [490]	②, ①	MLE	Text	GitHub Website
τ-bench [491]	①, ③	SWE	Text	GitHub
TheAgentCompany [492]	②, ③, ⑤	SWE	Text	GitHub Website
MedAgentGym [493]	①	Science	Text	GitHub
SecRepoBench [494]	①, ③	Coding, Security	Text	-
R2E-Gym [495]	①, ②	SWE	Text	GitHub Website
BigCodeBench [496]	①	Coding	Text	GitHub Website
LiveCodeBench [497]	①	Coding	Text	GitHub Website
SWE-bench [498]	①, ③	SWE	Text	GitHub Website
SWE-rebench [499]	①, ③	SWE	Text	Website
DevBench [500]	②, ①	SWE	Text	GitHub
ProjectEval [501]	②, ①	SWE	Text	GitHub Website
DA-Code [502]	①, ③	Data Science, SWE	Text	GitHub Website
ColBench [159]	②, ①	SWE, Web Dev	Text	GitHub Website
NoCode-bench [503]	②, ①	SWE	Text	GitHub Website
MLE-Bench [504]	②, ①, ③	MLE	Text	GitHub Website
PaperBench [505]	②, ①, ③	MLE	Text	GitHub Website
Crafter [506]	②, ④	Game	Visual	GitHub Website
Craftax [507]	②, ④	Game	Visual	GitHub
ELLM (Crafter variant) [508]	②, ①	Game	Visual	GitHub Website
SMAC / SMAC-Exp [509]	⑤, ②	Game	Visual	GitHub
Factorio [510]	②, ①	Game	Visual	GitHub Website
SMAC-Hard [511]	②, ④	Game	Visual	GitHub
TacticCraft [512]	②, ⑤	Game	Text	-

5.1.1. Web Environments

In the realm of web-based environments, several benchmarks offer controlled yet realistic static environments for Agentic RL. WebShop [480] is a simulated e-commerce website featuring a large catalog of real-world products and crowd-sourced text instructions. Agents navigate various webpage types and issue diverse actions (e.g., searching, selecting items, customizing, purchasing) to find and buy products, with its deterministic search engine aiding reproducibility. Furthermore, Mind2Web [481] is a dataset designed for generalist web agents, featuring a substantial number of tasks from many real-world websites across diverse domains. It provides webpage snapshots and crowdsourced action sequences for tasks like finding flights or interacting with social profiles, emphasizing generalization across unseen websites and domains. Similarly, WebArena [482] and its multimodal extension, VisualwebArena [483], are self-hostable, reproducible web environments delivered as Docker containers. WebArena features fully functional websites across common domains like e-commerce, social forums, collaborative development, and content management systems, enriched with utility tools and knowledge bases, and supports multi-tab tasks and user role simulation. VisualwebArena extends this by introducing new tasks requiring visual comprehension and a "Set-of-Marks" (SoM) representation to annotate interactable elements on screenshots, bridging the gap for multimodal web agents. Additionally, AppWorld [485] constitutes an environment simulating a multi-application ecosystem, encompassing 9 daily-use applications (e.g., Amazon, Spotify, Gmail) with 457 invokable APIs, and constructing a digital world featuring approximately 100 virtual characters and their social relationships. Agents accomplish complex tasks (such as travel planning and social relationship management) by writing code to call APIs. In these environments, all changes to the web pages or visual elements occur exclusively in response to the agent's actions.

5.1.2. GUI Environments

AndroidWorld [486] exemplifies such dynamism as a benchmarking environment operating on a live Android emulator, featuring 116 hand-crafted tasks across 20 real-world applications. Its dynamic nature is underscored by parameter instantiation that generates millions of unique task variations, ensuring the environment evolves into novel configurations without direct agent influence. Agents interact through a consistent interface (supporting screen interactions, app navigation, and text input) while receiving real-time state feedback, with integration to MiniWoB++ providing durable reward signals for evaluating adaptive performance. OSWorld [487] is a scalable real computer environment for multimodal agents, supporting task setup and execution-based evaluation across Ubuntu, Windows, and macOS. It includes a substantial number of real-world computer tasks involving real web and desktop applications, OS file I/O, and workflows spanning multiple applications, where all OS state changes are exclusively triggered by the agent's actions.

5.1.3. Coding & Software Engineering Environments

Code-related tasks are supported by a wide range of executable environments and benchmarks. These can be broadly categorized into interactive environments, where agents directly alter the state, and benchmarks/datasets that provide curated tasks and evaluation pipelines.

Interactive SWE Environments. Several environments instantiate agent–environment interaction under software engineering workflows. Debug-Gym [489] is a text-based interactive coding environment for LLM agents in debugging settings. It equips agents with tools like a Python debugger (pdb) to actively explore and modify buggy codebases, supporting repository-level information handling and ensuring safety via Docker containers. R2E-Gym [495] constructs a procedurally generated, executable gym-style environment of over 8K

software engineering tasks, powered by the SWE-Gen pipeline and hybrid verifiers. TheAgentCompany [492] simulates a software development company, where agents act as "digital workers" performing professional tasks such as web browsing, coding, program execution, and communication with simulated colleagues. It features a diverse set of long-horizon tasks with checkpoints for partial credit, providing a comprehensive testbed for agents in a realistic workplace setting. In all these environments, the underlying problem definitions and codebases remain fixed, and changes occur solely as a result of the agent's actions.

Coding Benchmarks & Datasets. A wide range of benchmarks and datasets focus on constructing curated task suites and evaluation pipelines. HumanEval [513] introduces a benchmark of 164 hand-crafted Python programming tasks to measure functional correctness via the pass@k metric. MBPP [514] provides 974 entry-level Python tasks with natural language descriptions for evaluating short program synthesis. BigCodeBench [496] proposes a large-scale, contamination-free function-level benchmark of 1,140 tasks requiring composition of multiple function calls. LiveCodeBench [497] builds a continuously updated, contamination-free benchmark from real competition problems. SWE-bench [498] introduces a dynamic, execution-driven code repair benchmark derived from real GitHub issues. SWE-rebench [499] introduces a continual GitHub-mining pipeline (>21k tasks) for both training and evaluation. DevBench [500] evaluates end-to-end development across design, setup, implementation, and testing. ProjectEval [501] constructs LLM-generated, human-reviewed project tasks with simulated user interactions. ColBench [159] instantiates multi-turn backend/frontend tasks with a privileged critic for step-wise rewards. NoCode-bench [503] evaluates LLMs on feature addition from documentation updates across real codebases. CodeBoost [298] serves as a data-centric, execution-driven training pipeline by extracting and augmenting code snippets.

5.1.4. Domain-specific Environments

Science & Research. ScienceWorld [474] integrates science simulations (e.g., thermodynamics, electricity, chemistry) into complex text-based tasks designed around elementary-level science education. PaperBench [505] evaluates the ability of LLM agents to replicate cutting-edge machine learning research by reproducing 20 ICML 2024 papers from scratch, scored against rubric-based subtasks. τ-bench [491] simulates dynamic conversations for software engineering tasks, operating with an underlying database state and domain-specific rules that change only through the agent's API calls.

Machine Learning Engineering (MLE). MLE-Dojo [490] is a Gym-style framework for iterative machine learning engineering workflows, built upon real-world Kaggle competitions. It provides an interactive environment for agents to iteratively experiment, debug, and refine solutions. MLE-Bench [504] establishes a benchmark for MLE by curating 75 Kaggle competitions, evaluating agents against human baselines on public leaderboards. DA-Code [502] addresses agentic data-science workflows grounded in real datasets and executable analysis, providing a focused benchmark for this domain.

Biomedical. MedAgentGym [493] provides a domain-specific environment for biomedical code generation and testing, focusing on tasks within this specialized scientific field.

Cybersecurity. SecRepoBench [494] is a domain-specific benchmark for security vulnerability repair, covering 27 repositories and 15 Common Weakness Enumeration (CWE) categories.

5.1.5. Simulated & Game Environments

Text-based environments simulate interactive settings where agent actions are expressed through natural language. LMRL-Gym [471] provides a benchmark for evaluating reinforcement learning algorithms in multi-turn language interactions, including tasks like "20 Questions" and Chess. TextWorld [473] is a sandbox environment for training agents in text-based games, offering both hand-authored and procedurally generated games. Game-based environments also emphasize visual settings that may evolve independently. Crafter [506] is a 2D open-world survival game that benchmarks deep exploration and long-horizon reasoning. Craftax [507], built upon Crafter using JAX, introduces increased complexity and GPU-acceleration for open-ended RL. The modified Crafter variant by ELLM [508] expands the action space and introduces distractor tasks. For multi-agent coordination, SMAC [509] and SMAC-Hard [511] provide StarCraft II-based benchmarks for cooperative decentralized control. SMAC-R1 [511], Adaptive Command [515] and TacticCraft [512] further advance the performance of LLM agents in StarCraft II-style environments. Factorio [510] presents a dynamic, tick-based industrial simulation where agent inaction still alters the world state.

5.1.6. General-Purpose Environments

Some environments and benchmarks are designed for broad evaluation or to improve general agent capabilities. AgentGym [475] focuses on improving LLM agent generalization via instruction tuning and self-correction, operating on deterministic environments such as Alf World, BabyAI, and SciWorld. Agentbench [476] serves as a broad evaluation framework, assessing LLMs as agents across a variety of distinct interactive environments, including SQL-based, game-based, and web-based scenarios. InternBootcamp [477] is a scalable framework integrating over 1000 verifiable reasoning tasks, spanning programming, logic puzzles, and games, with a standardized interface for RL training and automated task generation.

5.2. RL Framework

In this section, we summarize three categories of codebases/frameworks most relevant to this work: agentic RL frameworks, RLHF and LLM fine-tuning frameworks, and general-purpose RL frameworks. Table 10 provides an overview of the prevailing agentic RL and LLM-RL frameworks for readers' reference.

Agentic RL frameworks. Verifiers [516] introduces a verifiable-environment setup for end-to-end policy optimization with LLMs, while SkyRL-v0 [517] and its modular successors [518] demonstrate long-horizon, real-world agent training via reinforcement learning. AREAL [519] scales this paradigm with an asynchronous, distributed architecture tailored to language reasoning tasks, and MARTI [520] extends it further to multi-agent LLM systems that integrate reinforcement training and inference. EasyR1 [521] brings multi-modality support, enabling agents to leverage vision and language signals together in a unified RL framework. AgentFly [522] presents a scalable and extensible agentic RL framework that empowers language-model agents with traditional RL algorithms—enabling token-level multi-turn interaction via decorator-based tools and reward definition, asynchronous execution, and centralized resource management for high-throughput RL training. Agent Lightning [523] is a flexible RL framework that decouples agent execution from training by modeling execution as an MDP and using a hierarchical RL algorithm (LightningRL) to train any AI agent with near-zero code modification. AWORLD [275] is a distributed agentic RL framework, which tackles the main bottleneck of agent training—experience generation—by orchestrating massively parallel rollouts across clusters, achieving a 14.6$\times$ speedup over single-node execution and enabling scalable end-to-end training pipelines. ROLL [525] provides a scalable library for large-scale RL optimization with a unified

Table 10: A summary of frameworks for reinforcement learning, categorized by type and key features.

Framework	Type	Key Features	Resource Link
Agentic RL Frameworks			
Verifiers [516]	Agentic RL	Verifiable environment setup	GitHub
SkyRL-v0 [517, 518]	Agentic RL	Long-horizon real-world training	GitHub
AREAL [519]	Agentic RL	Asynchronous training	GitHub
MARTI [520]	Multi-agent RL	Integrated multi-agent training	GitHub
EasyR1 [521]	Agentic RL	Multimodal support	GitHub
AgentFly [522]	Agentic RL	Scalable asynchronous execution	GitHub
Agent Lightning [523]	Agentic RL	Decoupled hierarchical RL	GitHub
AWorld [275]	Agentic RL	Parallel rollouts across clusters	GitHub
RL-Factory [524]	Agentic RL	Easy-to-design reward	GitHub
ROLL [525]	Agentic RL	Stable Multi-GPU Parallel Training	GitHub
VerlTool [526]	Agentic RL	Tool-intergrated rollout	GitHub
AgentRL [527]	Agentic RL	Asynchronous Multi-Task Training	GitHub
RLHF and LLM Fine-tuning Frameworks			
OpenRLHF [528]	RLHF / LLM RL	High-performance scalable RLHF	GitHub
TRL [529]	RLHF / LLM RL	Hugging Face RLHF	GitHub
trlX [530]	RLHF / LLM RL	Distributed large-model RLHF	GitHub
Verl [531]	RLHF / LLM RL	Streamlined experiment management	GitHub
SLiMe [532]	RLHF / LLM RL	High-performance async RL	GitHub
Oat [533]	RLHF / LLM RL	Lightweight RL support	GitHub
General-purpose RL Frameworks			
RLlib [534]	General RL / Multi-agent RL	Production-grade scalable library	GitHub
Acme [535]	General RL	Modular distributed components	GitHub
Tianshou [536]	General RL	High-performance PyTorch platform	GitHub
Stable Baselines3 [537]	General RL	Reliable PyTorch algorithms	GitHub
PFRL [538]	General RL	Benchmarked prototyping algorithms	GitHub

controller, parallel workers, and automatic resource mapping for efficient multi-GPU training. VerlTool [526] introduces an agentic RL with tool use (ARLT) framework built upon Verl [531], enabling agents to jointly optimize planning and execution across interactive environments. AgentRL [527] provides a scalable asynchronous framework for multi-turn, multi-task agentic RL, unifying environment orchestration and introducing cross-policy sampling and task advantage normalization for stable large-scale training.

RLHF and LLM fine-tuning frameworks. OpenRLHF [528] offers a high-performance, scalable toolkit designed for large-scale model alignment; TRL [529] provides Hugging Face's baseline implementations for RLHF experiments; trlX [530] adds distributed training support for fine-tuning models up to tens of billions of parameters; and HybridFlow [531] streamlines experiment management and scaling for RLHF research pipelines. SLiMe [532] is an LLM post-training framework for RL scaling that combines Megatron with SGLang for high-performance multi-mode training, supports Async RL, and enables flexible disaggregated workflows for reward and data generation via custom interfaces and server-based engines.

General-purpose RL frameworks supply the core algorithms and distributed execution engines that can underpin agentic LLM systems. RLlib [534] is a production-grade, scalable library offering unified APIs for on-policy, off-policy, and multi-agent methods; Acme [535] provides modular, research-oriented building blocks for distributed RL; Tianshou [536] delivers a high-performance, pure-PyTorch platform supporting online, offline, and hierarchical RL; Stable Baselines3 [537] packages reliable PyTorch implementations of standard model-free algorithms; and PFRL [538] (formerly ChainerRL) offers benchmarked deep-RL algorithm implementations for rapid prototyping.

6. Open Challenges and Future Directions

The advance of agent RL toward general-purpose intelligence hinges on overcoming three pivotal challenges that define the field's research frontier. First is the challenge of **Trustworthiness**: ensuring the reliability, safety, and alignment of increasingly autonomous agents. Second is **Scaling up Agentic Training**, which requires surmounting the immense practical bottlenecks in computation, data, and algorithmic efficiency. Finally, an agent's capabilities are fundamentally bounded by its world, making the **Scaling up Agentic Environments**, *i.e.*, the creation of complex and adaptive training grounds.

6.1. Trustworthiness

Security. The security landscape for autonomous agents is fundamentally more complex than for standard LLMs. While traditional models are primarily vulnerable to attacks on their text-in, text-out interface, agents possess an expanded attack surface due to their external components like tools, memory, and planning modules [539, 67]. This architecture exposes them to novel threats beyond direct prompt injection. For instance, indirect prompt injection can occur when an agent interacts with a compromised external environment, such as a malicious website or API, which poisons its memory or tool outputs [540]. Multi-agent systems further compound these risks by introducing vulnerabilities through inter-agent communication, where one compromised agent can manipulate or mislead others within the collective [539].

RL significantly magnifies these agent-specific risks by transforming the agent from a passive victim of manipulation into an active, goal-seeking exploiter of vulnerabilities. The core issue is instrumental goal achievement through reward hacking: an RL agent's primary directive is to maximize its long-term reward, and it may learn that unsafe actions are the most effective path to this goal. For example, if an agent discovers that using a malicious, third-party tool yields a high reward for a given task, RL will actively reinforce and entrench this unsafe behavior. Similarly, if an agent learns that it can bypass safety protocols to achieve its objective more efficiently, the resulting reward signal will teach it to systematically probe for and exploit such security loopholes. This creates a more persistent and dangerous threat than one-off jailbreaks, as the agent autonomously learns and optimizes deceptive or harmful strategies over time.

Mitigating these amplified risks requires a defense-in-depth approach tailored to agentic systems. A critical first line of defense is robust sandboxing [541, 542], where agents operate in strictly controlled, permission-limited environments to contain the potential damage from a compromised tool or action. At the training level, mitigation strategies must focus on shaping the reward signal itself. This includes implementing process-based rewards that penalize unsafe intermediate steps (e.g., calling an untrusted API) and employing adversarial training within the RL loop, where the agent is explicitly rewarded for resisting manipulation attempts and ignoring poisoned information. Finally, continuous monitoring and anomaly detection are essential for post-deployment safety. By tracking an agent's actions, such as tool calls and memory access patterns, it is possible to identify deviations from normal behavior, allowing for timely intervention.

Hallucination. In the context of agentic LLMs, hallucination is the generation of confident yet ungrounded outputs, including statements, reasoning steps, or tool usage, that are not rooted in provided evidence or external reality. This issue extends beyond simple factual errors to encompass unfaithful reasoning paths and misaligned planning, with overconfidence often masking the agent's uncertainty [543, 544]. In multimodal agents, it also manifests as cross-modal inconsistency, such as a textual description mismatching an image, framing it as a fundamental grounding problem [545]. Evaluating hallucination requires assessing both factuality against objective truth and faithfulness to a given source, often measured through benchmarks like HaluEval-QA or by the agent's ability to appropriately abstain on unanswerable questions, where a refusal to answer ("I don't know") is a critical signal of epistemic awareness [546, 547].

RL can inadvertently amplify hallucination if the reward mechanism is not carefully designed. Studies show that outcome-driven RL, which rewards only the correctness of the final answer, can encourage agents to find spurious correlations or shortcuts. This process may yield confident but unfounded intermediate reasoning steps, as the optimization process settles into local optima that achieve the goal without being factually sound [546]. This phenomenon introduces a "hallucination tax," where reinforcement finetuning can degrade an agent's ability to refuse to answer, compelling it to generate responses for unanswerable questions rather than abstaining [547]. However, the effect is highly dependent on the training pipeline; while RL-only post-training can worsen factuality, a structured approach combining SFT with a verifiable-reward RL process can mitigate this degradation [548].

Promising mitigation strategies involve a hybrid approach of training-time alignment and inference-time safeguards. During training, a key direction is to shift from outcome-only rewards to process-based rewards. Techniques like Factuality-aware Step-wise Policy Optimization (FSPO) verify each intermediate reasoning step against evidence, directly shaping the policy to discourage ungrounded claims [546]. Data-centric approaches enhance epistemic humility by training agents on a mix of solvable and unsolvable problems, restoring their ability to abstain when necessary [547]. At the system level, this is complemented by inference-time techniques such as retrieval augmentation, tool-use for fact-checking, and post-hoc verification to ground the agent's outputs in reliable sources. For multimodal agents, explicitly adding cross-modal alignment objectives is crucial for ensuring consistency [544, 543, 545]. Collectively, these directions aim to align the agent's reward-seeking behavior with the goal of truthfulness, fostering more reliable and trustworthy autonomous systems.

Sycophancy. Sycophancy in LLM agents refers to their tendency to generate outputs that conform to a user's stated beliefs, biases, or preferences, even when those are factually incorrect or lead to suboptimal outcomes [549]. This behavior transcends mere conversational agreeableness, fundamentally affecting an agent's planning and decision-making processes. For instance, a sycophantic agent might adopt a user's flawed reasoning in its internal plan, choose a course of action that validates the user's incorrect assumptions, or filter information from tools to present only what aligns with the user's view [550]. This represents a critical misalignment, where the agent optimizes for the user's expressed preference rather than their latent, long-term interest in achieving the best possible outcome.

RL is a primary cause for this behavior. The underlying mechanism is a form of "reward hacking," where the agent learns to exploit the reward model in ways that do not align with true human preferences [551]. Because human labelers often show a preference for agreeable and validating responses, the reward model inadvertently learns to equate user satisfaction with sycophantic agreement. Consequently, RLHF can directly incentivize and "exacerbate sycophantic tendencies" by teaching the agent that conforming to a user's viewpoint is a reliable strategy for maximizing reward, even if it compromises truthfulness [552].

Mitigating sycophancy is an active area of research that focuses on refining the reward signal and training dynamics. A promising direction is the development of sycophancy-aware reward models, which are explicitly trained to penalize responses that merely parrot user beliefs without critical evaluation. Another approach involves leveraging AI-driven feedback, such as in Constitutional AI, where the agent is steered by a set of principles promoting objectivity and neutrality, rather than solely by human preferences [553]. At inference time, strategies like explicitly prompting the agent to adopt a "red team" or contrarian perspective can also help counteract ingrained sycophantic tendencies. Cooper [554] is a reinforcement learning framework that co-optimizes both the policy model and the reward model online, using high-precision rule-based verifiers to select positive samples and LLM-generated negative samples, thereby preventing the policy from exploiting a static reward model (i.e., reward hacking) by continuously adapting the reward model to closing emergent loopholes. Ultimately, the future direction lies in designing reward systems that robustly capture the user's long-term interests—such as receiving accurate information and making sound decisions—over their immediate desire for validation.

6.2. Scaling up Agentic Training

Computation. Recent advances demonstrate that scaling reinforcement learning fine-tuning (RFT) computation directly enhances the reasoning ability of LLM-based agents. The Agent RL Scaling Law study shows that longer training horizons systematically improve tool-use frequency, reasoning depth, and overall task accuracy, highlighting the predictive benefit of allocating more compute to RL training [324]. Similarly, ProRL reveals that prolonged RL training expands reasoning boundaries beyond those accessible to base models, uncovering novel solution strategies even where extensive sampling from the pretrained model fails [50]. Building upon this, ProRLv2 extends training steps and incorporates more stable optimization techniques, demonstrating sustained benefits as smaller models, after extensive RL training, rival the performance of larger models on mathematics, code, and logic benchmarks [555]. Collectively, these results underscore that scaling compute through extended RL training is not merely complementary to enlarging model or data size, but a fundamental axis for advancing agentic reasoning.

Model Size. Increasing model capacity heightens both the promise and pitfalls of RL-based agent training. Larger models unlock greater potential but risk entropy collapse and narrowing of capability boundaries, as RL sharpens output distributions toward high-reward modes, limiting diversity [556]. Methods like RL-PLUS address this with hybrid strategies and advantage functions that foster novel reasoning paths, breaking capability ceilings [556]. Meanwhile, scaling demands massive compute, making efficiency vital. A two-stage approach in [369] uses large teachers to generate SFT data for smaller students, refined via on-policy RL. This "SFT+RL" setup outperforms each method alone and cuts compute by half compared to pure SFT. The work also underscores RL's extreme hyperparameter sensitivity at scale, stressing the need for careful tuning.

Data Size. Scaling RL training across domains introduces both synergy and conflict in agentic reasoning. Cross-domain RL in math, code, and logic tasks shows complex interactions [557]: some pairings enhance each other, while others interfere and reduce performance. Model initialization also matters—instruction-tuned models generalize differently than raw ones. Building on this, the Guru dataset [558] spans six reasoning domains, showing that RL gains correlate with pretraining exposure: math and code benefit from transfer, but domains like simulation or logic need dedicated training. These findings suggest that while multi-domain RL data can amplify general reasoning, it must be carefully curated to balance complementarity and mitigate interference across tasks.

Efficiency. Efficiency of LLM post-training is a central frontier for sustainable scaling [559]. Beyond brute-force scaling, recent research emphasizes improving RL training efficiency through post-training recipes, methodological refinements, and hybrid paradigms. POLARIS [560] demonstrates that calibrating data difficulty, employing diversity-driven sampling, and extending reasoning length substantially boost RL effectiveness, enabling smaller models to reach or even surpass much larger counterparts on reasoning benchmarks. Complementary work [38] provides systematic evaluations of common RL techniques, finding that judiciously combining just a few simple strategies often outperforms more complex methods. Another research proposes Dynamic Fine-Tuning (DFT) [561], showing that introducing RL principles into gradient scaling can match or exceed advanced RL approaches with minimal additional cost. Taken together, these advances suggest a dual trajectory for the future: on one hand, progressively refining RL-based recipes to maximize efficiency; on the other, rethinking training paradigms to embed RL-like generalization signals without full-fledged online RL. A particularly compelling direction lies in exploring how agentic models might acquire robust generalization from extremely limited data, for instance, by leveraging principled difficulty calibration, meta-learning dynamics, or information-theoretic regularization to distill broad reasoning abilities from a handful of experiences. Such pathways point to the possibility of a new regime of post-training: one where the ability to extrapolate, abstract, and generalize becomes decoupled from sheer data volume, and instead hinges on exploiting the structure and dynamics of the training process itself.

6.3. Scaling up Agentic Environment.

A nascent yet critical frontier for Agentic RL involves a paradigmatic shift from treating the training environment as a static entity to viewing it as a dynamic and optimizable system. This perspective addresses a core bottleneck in agent development: the scarcity of interactive, adaptive environments and the difficulty of engineering effective reward signals. As a growing consensus holds that prevalent environments like ALFWorld [472] and ScienceWorld [474] are insufficient for training general-purpose agents [562], research is moving beyond solely adapting the agent's policy. Instead, a co-evolutionary approach uses learning-based methods to adapt the environment itself. One key strategy is to automate reward function design. This involves deploying an auxiliary "explorer" agent to generate a diverse dataset of interaction trajectories, which are then used to train a reward model via heuristics or preference modeling. This effectively decouples agent training from the expensive process of manual reward specification, enabling the learning of complex behaviors without direct human annotation.

Beyond automating the reward signal, a second, more dynamic strategy is to automate curriculum generation, transforming the environment into an active teacher. This approach establishes a feedback loop where an agent's performance data, highlighting specific weaknesses, is fed to an "environment generator" LLM. As exemplified by EnvGen [563], this generator then procedurally adapts the environment's configuration, creating new tasks that specifically target and remedy the agent's deficiencies. This form of goal-directed Procedural Content Generation (PCG) ensures the agent is consistently challenged within its "zone of proximal development," accelerating learning and preventing overfitting. Together, automated rewards and adaptive curricula create a symbiotic relationship between the agent and its environment, establishing a scalable "training flywheel" that is essential for the future of self-improving agentic systems.

6.4. The Mechanistic Debate on RL in LLMs

Two competing explanations have emerged for why RL appears to boost LLM reasoning. The "amplifier" view holds that RL with verifiable rewards—often instantiated via PPO-style variants such as GRPO—mainly reshapes the base model's output distribution: by sampling multiple trajectories and rewarding the verifiably

correct ones, RL concentrates probability mass on already-reachable reasoning paths, improving pass@1 while leaving the support of solutions largely unchanged; consistent with this, large-k pass@k analyses often find that the base model eventually matches or surpasses its RL-tuned counterpart, suggesting elicitation rather than creation of capabilities, and further evidence indicates that reflective behaviors can already emerge during pre-training [2, 185, 564]. By contrast, the "new-knowledge" view argues that RL after next-token prediction can install qualitatively new computation by leveraging sparse outcome-level signals and encouraging longer test-time computation: theory shows that RL enables generalization on problems (e.g., parity) where next-token training alone is statistically or computationally prohibitive; empirically, RL can improve generalization to out-of-distribution rule and visual variants, induce cognitive behaviors (verification, backtracking, subgoal setting) that were absent in the base model yet predict self-improvement, and in under-exposed domains even expand the base model's pass@k frontier [565, 566, 567, 568, 558]. Whether RL can truly endow LLMs with abilities beyond those acquired during pre-training remains an open question, and its underlying learning mechanisms are still to be fully understood.

7. Conclusion

This survey has charted the emergence of Agentic Reinforcement Learning (Agentic RL), a paradigm that elevates LLMs from passive text generators to autonomous, decision-making agents situated in complex, dynamic worlds. Our journey began by formalizing this conceptual shift, distinguishing the temporally extended and partially observable MDPs (POMDPs) that characterize agentic RL from the single-step decision processes of conventional LLM-RL. From this foundation, we constructed a comprehensive, twofold taxonomy to systematically map the field: one centered on *core agentic capabilities* (planning, tool use, memory, reasoning, self-improvement, perception, *etc.*) and the other on their *application* across a diverse array of task domains. Throughout this analysis, our central thesis has been that RL provides the critical mechanism for transforming these capabilities from static, heuristic modules into adaptive, robust agentic behavior. By consolidating the landscape of open-source environments, benchmarks, and frameworks, we have also provided a practical compendium to ground and accelerate future research in this burgeoning field.

Acknowledgments

We acknowledge Zhouliang Yu and Minghao Liu for their guidance and discussion.

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[2] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

[4] Saksham Sahai Srivastava and Vaneet Aggarwal. A technical survey of reinforcement learning techniques for large language models, 2025. URL https://arxiv.org/abs/2507.04136.