Generate, Filter, Control, Replay: A Comprehensive Survey of Rollout Strategies for LLM Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

Reinforcement learning (RL) has become a central post-training tool for improving the reasoning abilities of large language models (LLMs). In these systems, the *rollout*, the trajectory sampled from a prompt to termination, including intermediate reasoning steps and optional tool or environment interactions, determines the data that the optimizer ultimately learns from, yet rollout design is often treated as an implementation detail and underreported. This survey provides an optimizer-agnostic view of rollout strategies for RL-based post-training of reasoning LLMs. We formalize rollout pipelines with unified notation and introduce Generate–Filter–Control–Replay (GFCR), a lifecycle taxonomy that decomposes rollout pipelines into four modular and composable stages: Generate proposes candidate trajectories and topologies; Filter constructs intermediate signals via verifiers, judges, or critics; Control allocates compute and makes continuation/branching/stopping decisions under budgets; and Replay retains and reuses artifacts across rollouts without weight updates, including self-evolving curricula that autonomously generate new training tasks and data. We complement GFCR with a criterion taxonomy of reliability, coverage, and cost sensitivity that characterizes the trade-offs rollout designs must navigate. Using this framework, we synthesize methods spanning RL with verifiable rewards, process supervision, judge-based gating, guided and tree/segment rollouts, adaptive compute allocation, early-exit and partial rollouts, systems-level throughput optimization, and replay/recomposition for self-improvement. We ground the framework with case studies in math, code/SQL, multimodal reasoning, tool-using agents, and agentic skill benchmarks that evaluate skill induction, reuse, and cross-task transfer. Finally, we provide a practitioner-oriented diagnostic index that maps common rollout pathologies to GFCR modules and mitigation levers, alongside open challenges for building reproducible, compute-efficient, and trustworthy rollout pipelines.

1 Introduction

Reinforcement learning (RL) has become a core component of post-training pipelines for large language models (LLMs) (Ouyang et al., 2022), enabling substantial gains on reasoning-intensive tasks such as mathematics and code/SQL generation (Shao et al., 2024; Guo et al., 2025b; Le et al., 2022; Yao et al., 2026) and supporting multi-step decision making in agentic settings (Zhang et al., 2025h). Much of the literature emphasizes the learning objective and optimizer (e.g., PPO/GRPO-style updates or preference-based alternatives) (Schulman et al., 2017; Shao et al., 2024; Guo et al., 2025b; Rafailov et al., 2023) and the form of supervision (verifiers, reward models, or AI/LLM judges) (Christiano et al., 2017; Ouyang et al., 2022; Lightman et al., 2023; Lee et al., 2023; Zheng et al., 2023; Gu et al., 2024). However, these optimizers can only learn from the data they are given. In modern post-training, that data is produced online by the model itself through *rollouts*, and rollout design often dominates both the cost of training and the quality of the resulting learning signal.

A *rollout* is a sampled trajectory from a prompt to termination. In text-only settings, a rollout reduces to a completion containing intermediate reasoning and a final answer; in tool- or environment-interactive settings, it includes action–observation loops and external feedback. Reasoning tasks make rollout design

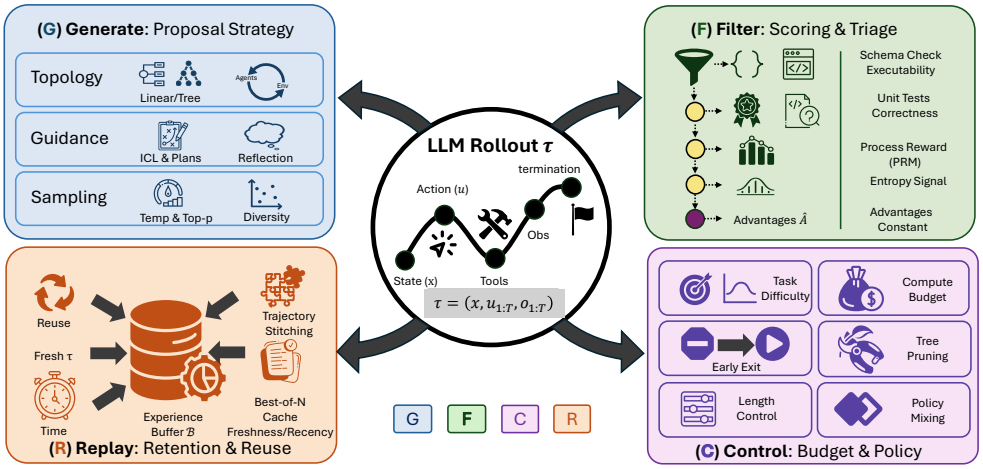

Figure 1: An overview of the rollout lifecycle and the Generate–Filter–Control–Replay (GFCR) decomposition. Rollout pipelines can be understood as modular choices about how trajectories are proposed, how intermediate signals are constructed, how compute is allocated under budgets, and what artifacts are retained and reused across rollouts.

particularly consequential: the solution space is sparse and sensitive to early errors, and a single early mistake can invalidate an entire derivation or program. As a result, the rollout mechanism is not merely a decoding choice—it acts as a data-generation policy that determines what evidence is collected, which candidates survive, and where compute is spent. Common rollout design axes include topology, sampling diversity, granularity, stopping criteria, branching, and artifact reuse.

Recent progress makes these dependencies explicit. Process-level evaluators such as process reward models require step-delimited rollouts (Lightman et al., 2023; Zhao et al., 2025a; Zou et al., 2025); tree-structured rollouts rely on branching and backup rules (Yao et al., 2023; Hou et al., 2025; Wu et al., 2025a); and adaptive budgeting policies trade off rollout compute against accuracy by allocating more samples to informative prompts and skipping uninformative ones (Zheng et al., 2025c; Nguyen et al., 2026). Replay and recomposition further complicate the picture by reusing cached responses, recombining previously generated trajectories, or mining self-generated tasks across iterations without weight updates (Li et al., 2025c; Zhang et al., 2025l; Li et al., 2025b; Chen et al., 2025e). These design choices shape not only performance but also key desiderata such as reliability of supervision, coverage and informativeness of sampled trajectories, and sensitivity to compute budgets.

Despite their centrality, rollout strategies are often treated as implementation details and underreported. Existing surveys typically organize the space around optimization algorithms, reward modeling, or pipeline-wide categorizations of where RL appears (Kaufmann et al., 2025; Wang et al., 2024a; Srivastava & Aggarwal, 2025; Guo & Wang, 2025), while reasoning- and agent-centric surveys focus on multi-step deliberation and search rather than the rollout pipeline itself (Zhang et al., 2025b;i). This leaves rollout design implicit and obscures the link between data collection and learning outcomes. The resulting reproducibility challenge is significant: unreported changes in rollout configuration can confound comparisons and make it difficult to attribute improvements to the optimizer versus the data-generation pipeline.

This survey centers rollout design as a first-class object. We formalize rollout pipelines with unified notation and introduce **Generate–Filter–Control–Replay (GFCR)**, an optimizer-agnostic, lifecycle decomposition of rollout pipelines into four modular and composable stages—functionally distinct yet frequently interleaved, since filter signals trigger control decisions, replay artifacts seed future generation, and control policies determine what enters replay. **Generate** specifies how candidate trajectories are proposed (topol-

ogy, scaffolding, sampling). **Filter** maps rollouts and prefixes to intermediate signals and optimizer-facing supervision (verifiers, judges, process scorers, learning-value diagnostics). **Control** allocates compute and makes continuation/branching/stopping decisions under budgets, and encompasses systems-level throughput optimization (speculative decoding, scheduling, and load balancing for rollout generation). **Replay** retains and reuses artifacts across rollouts without weight updates (buffers, caching, recomposition), and further encompasses self-evolving curricula in which rollouts autonomously generate new tasks, solutions, or agents that feed back into training. We complement GFCR with a *criterion taxonomy* of reliability, coverage and informativeness, and cost sensitivity that characterizes the trade-offs rollout designs must navigate and provides a principled basis for evaluating design choices across modules. Figure 1 provides an overview of the GFCR lifecycle and its interfaces.

Contributions. To our knowledge, this is the first survey that systematically organizes *rollout strategies* for RL-based post-training of reasoning LLMs. Our primary contributions are:

- We introduce GFCR, a unified taxonomy for describing rollout pipelines independently of the underlying optimizer, paired with a complementary criterion taxonomy (reliability, coverage, cost sensitivity) that provides a common vocabulary for comparing trajectory proposal, filtering, compute control, and reuse.
- We synthesize rollout-centric methods spanning RL with verifiable rewards, process supervision, judge-based gating, guided and tree/segment rollouts, adaptive compute allocation, early-exit and partial rollouts, systems-level throughput optimization, and replay/recomposition/self-evolution for self-improvement.
- We ground the taxonomy with case studies in mathematics, code/SQL, multimodal reasoning, tool-using agents, and agentic skill benchmarks—including skill induction, library management, and cross-task transfer—illustrating how interface constraints and feedback availability shape rollout design.
- We provide a structured diagnostic index that maps common rollout pathologies to GFCR modules and concrete mitigation levers, enabling practitioner-oriented troubleshooting.
- We identify open challenges, including verifier/judge calibration, principled compute accounting, and safe self-evolution with provenance tracking, and offer recommendations for reporting and evaluation to improve reproducibility.

Organization. We introduce global notation and formalize rollout pipelines in §3. We then survey the design space module by module: Generate (§4), Filter (§5), Control (§6), and Replay (§7), followed by domain case studies, practitioner-oriented diagnostics, and open problems.

2 Related Work

2.1 Comparison to Prior Surveys

Recent progress in reasoning-focused post-training and inference time deliberation increasingly depends on how systems sample, structure, score, and reuse trajectories under a compute budget, including grouped sampling, branching search, tool interaction trajectories, stepwise versus terminal scoring, adaptive stopping, and replay. Most prior surveys organize the space primarily around feedback modeling, reward learning, and optimization objectives, leaving rollout strategy implicit (Kaufmann et al., 2025; Jiang et al., 2025d; Chaudhari et al., 2025).

Surveys of RLHF and preference learning emphasize feedback collection and modeling, alignment loops, and evaluation protocols (Kaufmann et al., 2025; Jiang et al., 2025d; Chaudhari et al., 2025). Surveys of RL-enhanced LLMs summarize RLHF, RLAIF, and direct preference families and discuss optimization challenges (Wang et al., 2024a). Technical surveys focus on RL algorithms and training mechanics for LLM fine-tuning (Srivastava & Aggarwal, 2025; Li et al., 2025f), while pipeline-wide surveys categorize where RL

appears across data generation, pretraining, post-training, and test-time inference (Guo & Wang, 2025). Reasoning and agent-centric surveys review multi-step deliberation, search, and environment interaction (Zhang et al., 2025i;b). In contrast, we center rollout strategy as the unit of analysis and provide GFCR as a modular vocabulary for comparing how topology, sampling, scoring granularity, budget allocation, and experience reuse compose into end-to-end systems. Accordingly, we focus on methods where rollout design plays a substantive role, rather than surveying RL algorithms or reward modeling in isolation.

3 Foundations: Rollouts, Criteria, and the GFCR Framework

Modern LLM post-training increasingly relies on *rollout-centric* pipelines: for a given prompt, a system generates one or more candidate trajectories, evaluates them with verifiers or judges, and converts the resulting signals into training supervision. In this setting, a *rollout strategy* is rarely a single algorithmic knob—it is an end-to-end pipeline spanning how trajectories are proposed, how they are scored, how compute is allocated under a budget, and what artifacts are retained and reused. Without a shared vocabulary and notation, it becomes difficult to compare methods across domains (math/code/agents) or to isolate which component is responsible for a reported gain.

This section establishes the foundations used throughout the survey. We first introduce **GFCR**, a functional decomposition of rollout pipelines into **Generate**, **Filter**, **Control**, and **Replay** modules (Figure 2). We then preview the GFCR taxonomy at a glance (Table 1), define global notation for rollouts, prefixes, and group sampling, and conclude with a complementary *criterion taxonomy* that summarizes the reliability/coverage/cost desiderata that rollout designs must trade off.

3.1 GFCR as a Unifying Rollout Framework

We view modern rollout pipelines for reasoning LLMs as compositions of four lifecycle modules that operate on rollouts (or rollout prefixes) under compute budgets: **Generate (G)** proposes candidate trajectories; **Filter (F)** extracts signals and converts them into training-facing supervision; **Control (C)** allocates compute and makes continuation, branching, and stopping decisions; and **Replay (R)** retains and reuses artifacts across rollouts without weight updates. Figure 2 illustrates how these components compose into an end-to-end rollout system.

Crucially, these modules are often *interleaved* rather than sequential: Filter signals can trigger Control decisions such as pruning, early stopping, or adaptive resampling; Replay artifacts (cached responses, verified sub-traces, or mined tasks) can seed future Generate calls; and Control policies determine which artifacts enter Replay and when replayed artifacts are trusted, refreshed, or discarded. This decomposition provides a unified vocabulary for describing rollout pipelines as modular design choices, rather than monolithic *rollout strategies*.

To preview the remainder of the survey, Table 1 summarizes our rollout-centric taxonomy, organized as four composable lifecycle modules and indexed to the detailed sections.

3.2 Global Notation

We formalize *rollouts* as the primary object manipulated by GFCR. Table 2 summarizes the global notation used throughout the paper (module-specific symbols are introduced in the corresponding GFCR sections). Let $x \sim \mathcal{D}$ be a prompt/task input. A reasoning LLM with parameters θ induces a stochastic policy π_θ that generates an interaction trajectory until termination. We write a rollout as

$$\tau = (x, u_{1:T}, o_{1:T}), \tag{1}$$

where u_t denotes the model's output/action at step t (text tokens and/or a tool/action call), o_t is the corresponding observation (tool output/environment response; empty in pure-text settings), and T is a stopping time (EOS, max length, success, environment done). Equivalently, define the (implicit) state as $s_t = (x, u_{1:t-1}, o_{1:t-1})$, sample actions $u_t \sim \pi_\theta(\cdot \mid s_t)$, and sample observations $o_t \sim P(\cdot \mid s_t, u_t)$; this induces a trajectory distribution $p_\theta(\tau \mid x)$.

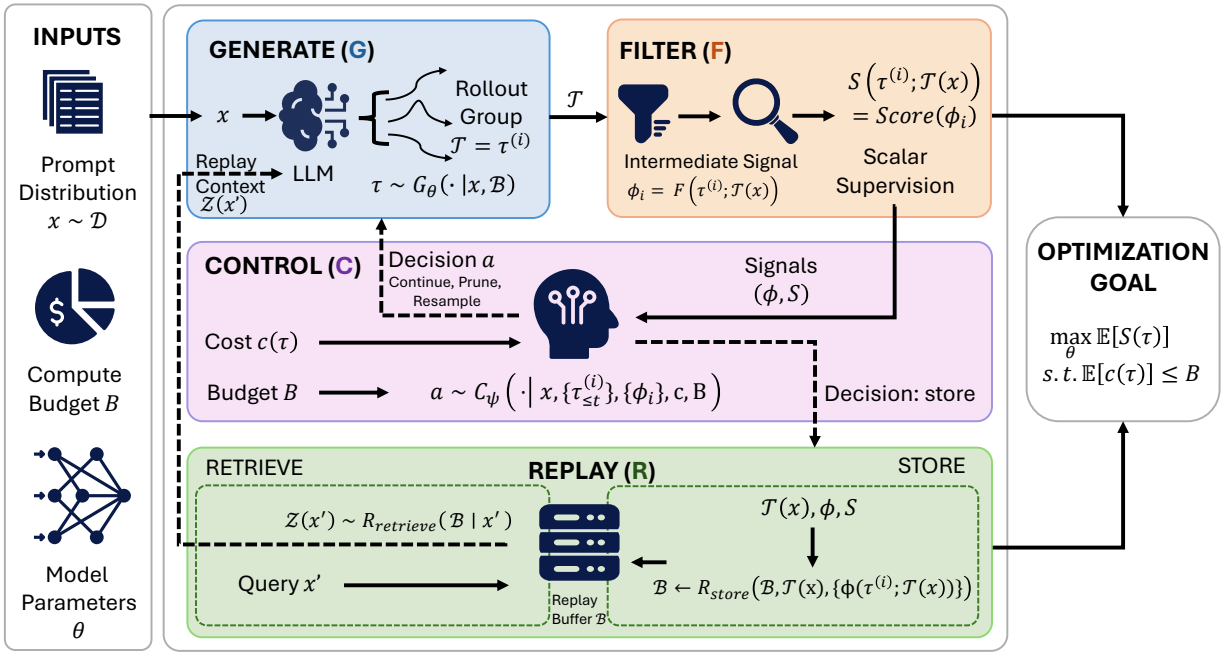

Figure 2: **GFCR as an end-to-end rollout system.** Given prompts $x \sim \mathcal{D}$ and a compute budget B, **Generate** samples a rollout group $\mathcal{T}(x)$; **Filter** maps each rollout to intermediate signals ϕ and training-facing supervision S; **Control** uses costs and signals to adapt continuation/pruning/resampling and decide what to store; and **Replay** retrieves/stores artifacts that condition future generation. The overall objective is to maximize expected utility $\mathbb{E}[S(\tau)]$ under compute constraints.

Many rollout pipelines also operate on *prefixes* (partial rollouts), e.g., for step/segment scoring, pruning, or tree expansion; we denote a prefix by $\tau_{\leq t} = (x, u_{1:t}, o_{1:t})$. In text-only settings, observations are empty, and a rollout reduces to a completion; we write $y \equiv u_{1:T}$ and use τ and y interchangeably when environment interaction is irrelevant. We also write $p_\theta(y \mid x)$ for the induced distribution over text-only completions.

A training system typically samples either a single rollout $\tau \sim p_\theta(\cdot \mid x)$ or a *group* of K rollouts

$$\mathcal{T}(x) = \{\tau^{(i)}\}_{i=1}^{K}, \qquad \tau^{(i)} \sim p_\theta(\cdot \mid x). \tag{2}$$

Each rollout is assigned supervision that may depend on the full group (e.g., listwise judging, within-group baselines). We denote intermediate **Filter** signals by $\phi(\tau^{(i)}; \mathcal{T}(x))$ (validity, verification outcomes, process scores, judge ranks, learning-value signals, etc.), and write the resulting training signal as

$$S(\tau^{(i)}; \mathcal{T}(x)) = \mathrm{Score}\Big(\phi(\tau^{(i)}; \mathcal{T}(x))\Big). \tag{3}$$

For brevity, we often write $\phi_i \equiv \phi(\tau^{(i)}; \mathcal{T}(x))$ and $S_i \equiv S(\tau^{(i)}; \mathcal{T}(x))$ when the context is clear. When the training signal depends only on a single trajectory (i.e., is not group-dependent), we write $S(\tau)$ as shorthand.

We track rollout cost $c(\tau)$ (e.g., decoded tokens, tool calls, wall-clock) and impose budgets such as $\sum_{i=1}^{K} c(\tau^{(i)}) \leq B_x$. At a high level, rollout-centric RL aims to increase expected utility under compute constraints:

$$\max_\theta \; \mathbb{E}_{x\sim\mathcal{D}} \, \mathbb{E}_{\tau\sim p_\theta(\cdot|x)}[S(\tau)] \quad \text{s.t.} \quad \mathbb{E}[c(\tau)] \leq B. \tag{4}$$

Finally, real systems interleave generation, filtering, budget decisions, and reuse. We denote by $q_{\theta,\mathrm{GFCR}}(\mathcal{T} \mid x, \mathcal{B})$ the distribution over rollout groups induced by the full GFCR pipeline given prompt x and current

Table 1: GFCR taxonomy at a glance. Rollout behavior is assembled from four composable lifecycle modules.

Module	Subcomponent	What it includes
Generate (G): How trajectories are proposed. ***Defines the proposal distribution + interaction structure.***		
§4.2	**Topology & Interaction**	Rollout structure and interaction patterns.
§4.3	**Guidance & scaffolding**	Prompt structure, rubrics, and auxiliary traces.
§4.4	**Sampling & exploration**	Decoding diversity and exploration signals.
Filter (F): Triage, scoring, and signal construction. ***Maps rollouts/prefixes to training-facing signals.***		
§5.2	**Structural validity**	Schema compliance, executability, and safety gates.
§5.3	**Correctness verification**	Outcome checking via tests or solvers.
§5.4	**Process quality**	Step-level signals for reasoning quality.
§5.5	**Comparative assessment**	Ranking, judging, and calibration signals.
§5.6	**Learning-value signals**	Informativeness proxies: novelty and uncertainty.
§5.7	**Training-signal construction**	Scores mapped to labels or weights.
Control (C): Compute allocation and decision rules. ***Allocates compute; decides continue/stop under budgets.***		
§6.2	**Prompt and task selection**	Curricula and instance prioritization policies.
§6.3	**Budgeting & scheduling**	Adaptive compute allocation and scheduling.
§6.4	**Rollout configuration control**	Length, temperature, and efficiency constraints.
§6.5	**Continuation and stop rules**	Early stopping and continuation criteria.
§6.6	**Branch and prune control**	Branching policies and pruning heuristics.
§6.7	**On-/off-policy controls**	Mixing constraints and correction policies.
§6.8	**Systems considerations**	Throughput optimization and systems constraints.
Replay (R): Retention, reuse, and self-evolution. ***Persists artifacts across rollouts without weight updates.***		
§7.2	**Response resampling and retention**	Buffers, caching, and experience reuse.
§7.3	**Recomposition**	Reusing verified segments to compose.
§7.4	**Self-evolution**	Generate new tasks via self-improvement.

replay state $\mathcal{B}$. When the dependence on $\mathcal{B}$ is not central, we may suppress it and write $q_{\theta,\mathrm{GFCR}}(\mathcal{T} \mid x)$. Detailed, module-specific formulations and additional notation are introduced in the corresponding GFCR sections.

Pipeline-Induced Distributions and Module Interfaces Recall from §3.2 that $q_{\theta,\mathrm{GFCR}}(\mathcal{T} \mid x)$ denotes the distribution over rollout groups induced by the full GFCR pipeline for a given prompt x. The i.i.d. sampling view $\tau^{(i)} \sim p_\theta(\cdot \mid x)$ abstracts away the fact that modern systems are *pipelines*: they interleave generation, filtering, budgeted decision-making, and reuse across prompts and iterations.

Unlike $p_\theta(\tau \mid x)$, the induced distribution $q_{\theta,\mathrm{GFCR}}$ may depend on intermediate filter signals, budgeting/stop decisions, adaptive sampling (e.g., variable K), and a persistent replay state $\mathcal{B}$ that stores and reuses artifacts from past rollouts. Figure 2 provides a schematic view of these interleavings. In later sections, we instantiate each module (Generate/Filter/Control/Replay) with module-specific problem formulations and notation, while retaining $q_{\theta,\mathrm{GFCR}}$ as the unifying object describing end-to-end rollout behavior.

3.3 Rollout Criterion Taxonomy

A rollout strategy is defined not only by how it generates trajectories, but also by the criteria used to judge whether those trajectories are useful for selection, learning, and compute allocation. We therefore organize rollout criteria along a small set of high-level dimensions that capture what a pipeline is trying to optimize. This perspective is complementary to GFCR. GFCR describes the functional modules that implement a rollout pipeline, whereas the criterion taxonomy describes the desiderata that those modules must satisfy and the trade-offs that arise when they are optimized jointly.

Table 2: Global notation used throughout the paper. Module-specific notation is introduced in the corresponding GFCR sections.

Symbol	Meaning
$\mathcal{D}$	Distribution (dataset) over prompts/tasks.
x	Prompt/task instance sampled from $\mathcal{D}$.
θ	Parameters of the language model / policy.
$\pi_\theta(\cdot \mid s)$	Model policy over next output/action given state s.
u_t	Model output/action at step t (tokens and/or tool/action call).
o_t	Observation at step t (tool output/environment response; empty for text-only).
T	Stopping time (EOS, max length, success, environment done).
$s_t = (x, u_{1:t-1}, o_{1:t-1})$	Implicit rollout state at step t.
$P(\cdot \mid s_t, u_t)$	Environment/observation kernel (may be deterministic tools).
$\tau = (x, u_{1:T}, o_{1:T})$	A rollout trajectory.
$\tau_{\leq t}$	Prefix (partial rollout) up to step t.
y	Text-only completion corresponding to a rollout (used when $o_{1:T} = \emptyset$).
$y^{(i)}$	Text-only completion for the i-th rollout $\tau^{(i)}$ (i.e., $y^{(i)} \equiv u^{(i)}_{1:T}$).
$p_\theta(\tau \mid x)$	Trajectory distribution induced by π_θ (and P).
$p_\theta(y \mid x)$	Completion distribution induced by π_θ in text-only settings (i.e., the marginal of $p_\theta(\tau \mid x)$ when $o_{1:T} = \emptyset$).
$\mathcal{T}(x) = \{\tau^{(i)}\}_{i=1}^K$	Group of K rollouts sampled for the same prompt x.
K	Number of rollouts in a group (may be adaptive).
$\phi(\tau; \mathcal{T})$	Intermediate **Filter** signals computed from rollouts (validity, verification, process/judge scores, etc.).
$S(\tau; \mathcal{T})$	Training signal derived from ϕ (may be group-dependent).
t	Step index within a rollout ($t = 1, \ldots, T$).
$c(\tau)$	Compute cost of a rollout (tokens, tool calls, wall-clock, etc.).
B, B_x	Global budget / per-prompt budget constraint.
$\mathcal{B}$	Replay buffer/cache storing past rollouts or sub-trajectories.
$q_{\theta,\mathrm{GFCR}}(\mathcal{T} \mid x, \mathcal{B})$	Rollout-group distribution induced by the full GFCR pipeline given replay state $\mathcal{B}$ (we may suppress $\mathcal{B}$ when clear).
t_{store}	Timestamp/age associated with a replay entry (when stored in $\mathcal{B}$).
ϕ_i	Shorthand for $\phi(\tau^{(i)}; \mathcal{T}(x))$ when context is clear.
S_i	Shorthand for $S(\tau^{(i)}; \mathcal{T}(x))$.
G, F, C, R	**G**enerate / **F**ilter / **C**ontrol / **R**eplay modules in GFCR.

Concretely, GFCR is a *functional decomposition*: it categorizes *what* a rollout pipeline does (propose, measure, decide, reuse). By contrast, the criterion taxonomy characterizes *why* those choices are made and *how* they should be judged. For example, best-of-K belongs in **Generate/Filter/Control**, while the question *is best-of-K worth it under budget B and does it improve reliability or merely increase variance?* belongs in the criterion taxonomy. Table 3 summarizes the core rollout criteria we use throughout the survey to compare design choices across GFCR modules. Figure 3 provides an at-a-glance visualization of the rollout criteria summarized in Table 3.

Reliability emphasizes that a rollout is only as useful as the trustworthiness of the signal derived from it. When tasks admit automatic verification, reliability is dominated by correctness under verifiers. When verification is unavailable, reliability depends on the stability and bias profile of model-based judges and on defenses against specification gaming. Coverage and informativeness capture the fact that many pipelines benefit from sampling multiple candidates, not only to increase the chance of success, but also to expose discriminative signals such as disagreement and near-miss behavior. Cost sensitivity captures the compute-constrained nature of modern rollouts and motivates budget-aware policies that trade trajectory length, branching, and sample count against utility, often using early stopping and reuse mechanisms.

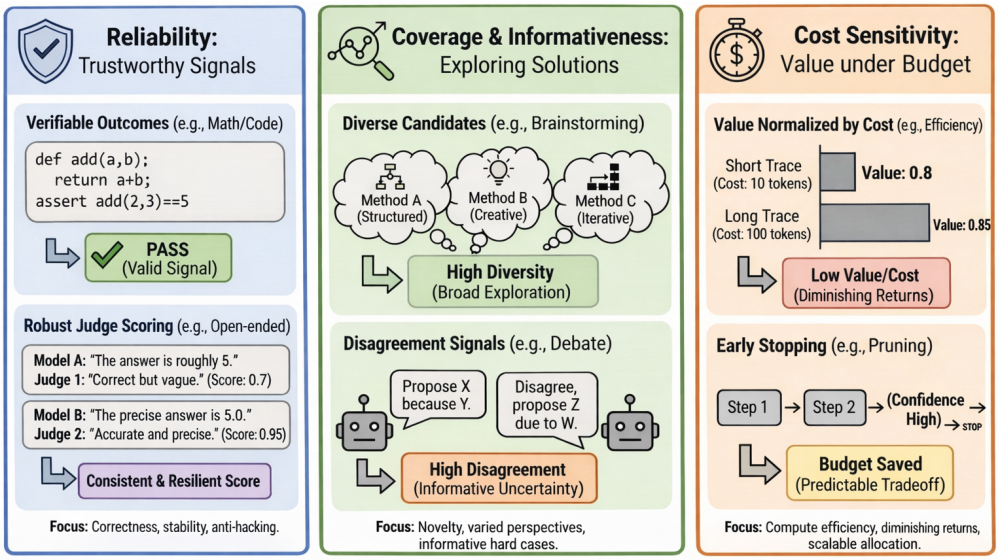

Figure 3: **Rollout criterion taxonomy (at a glance).** We evaluate rollout strategies along three cross-cutting desiderata: *reliability* (trustworthy signals via verifiers or robust judges), *coverage & informativeness* (diverse candidates and disagreement/uncertainty), and *cost sensitivity* (value under compute budgets via value-per-cost and early stopping).

Table 3: **Rollout criterion taxonomy (reference style).** We summarize three cross-cutting desiderata for rollout strategies—*reliability*, *coverage & informativeness*, and *cost sensitivity*—and list concrete signals and mechanisms that operationalize each criterion.

Criterion	Description and Examples
Reliability	
Verifiable Outcomes	Executable / checkable rollouts yielding trustworthy supervision when verification exists (e.g., math or code): structural validity/executability gates, unit tests, symbolic checks, exact-match constraints.
Robust Judge Scoring	Stable evaluation when direct verification is unavailable (e.g., open-ended tasks): calibrated judging, consistency across prompt variants, and resistance to reward hacking.
Coverage & Informativeness	
Diverse Candidates	Broad exploration over plausible solution strategies (structured, creative, iterative), avoiding near-duplicate rollouts and encouraging novelty.
Disagreement Signals	Informative uncertainty revealing hard-but-solvable cases: disagreement across rollouts or critics, conflicting rationales, and high-variance outcomes.
Cost Sensitivity	
Value Normalized by Cost	High utility per token/compute, exhibiting diminishing returns for longer traces or larger sample pools, with predictable quality–cost trade-offs.
Early Stopping (Pruning)	Adaptive truncation when confidence is high or marginal gains are low, enabling compute-efficient rollout allocation.

4 Generate: How Trajectories Are Proposed

4.1 Problem Formulation: Generate (G)

Using the global notation in §3.2, the **Generate** module specifies how candidate rollouts are proposed for a prompt $x \sim \mathcal{D}$. Its output is a candidate set (or structured collection) of rollouts $\mathcal{T}(x) = \{\tau^{(i)}\}_{i=1}^{K}$ that becomes the input to **Filter** and is further shaped by **Control** decisions. We model Generate as inducing a proposal distribution over candidate sets,

$$\mathcal{T} \sim q_\theta^G(\cdot \mid x;\ z,\ \kappa_G,\ \mathsf{Topo})\,, \tag{5}$$

where $q_\theta^G(\mathcal{T} \mid x; z, \kappa_G, \mathsf{Topo})$ denotes the Generate-module proposal distribution over rollout groups $\mathcal{T}$ (induced by the current policy/model parameters θ under the specified rollout procedure), z denotes guidance/scaffolding information available at rollout time (e.g., rubrics, plans, critiques/repairs, tool traces, retrieved exemplars), κ_G denotes sampling/exploration configuration (e.g., temperature/top-p/diversity), and Topo specifies the rollout topology and interaction pattern. Figure 4 provides a visual map of these design axes (cf. Table 4).

In the simplest *group* setting, the Generate module produces K candidates that are (approximately) conditionally independent given the rollout context and sampling configuration:

$$\mathcal{T}(x) = \{\tau^{(i)}\}_{i=1}^{K}, \qquad \tau^{(i)} \sim p_{\theta,\kappa_G}(\cdot \mid x; z), \tag{6}$$

with $K = 1$ recovering a single (linear) rollout. Here $p_{\theta,\kappa_G}(\tau \mid x; z)$ denotes the trajectory distribution induced by running π_θ with decoding/sampling configuration κ_G and guidance z included in the rollout context/state; in tool/environment settings, stochastic observations/transitions are drawn via $P(\cdot \mid s_t, u_t)$.

More structured topologies induce dependencies among candidates. For tree/graph rollouts, Generate expands partial histories/prefixes $\tau_{\leq t}$, producing a structured object (a search tree/graph) whose terminal nodes (leaves, in the tree case) correspond to complete rollouts; shared prefixes amortize computation while enabling branching at uncertain points. For tool/environment rollouts, τ includes both actions and observations, and the induced rollout distribution depends on the environment kernel $P(\cdot \mid s_t, u_t)$ (transitions/observations). Across all cases, topology (Topo), scaffolding (z), and sampling knobs (κ_G) jointly determine the diversity, support, and cost profile of the proposed candidates that the rest of the GFCR pipeline operates on.

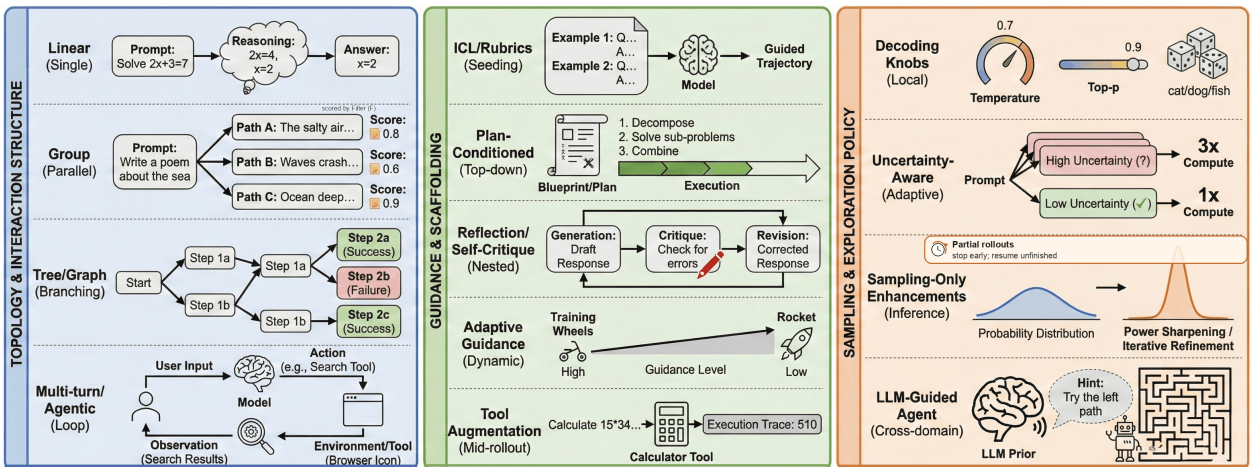

Figure 4: **Generate module design space.** Rollout proposal mechanisms can be organized along three axes: *(left)* topology & interaction (single, group, tree/graph, and multi-turn/tool rollouts), *(middle)* guidance/scaffolding z (ICL/rubrics, plans, reflection, adaptive guidance, tool augmentation), and *(right)* sampling & exploration configuration κ_G (decoding knobs, uncertainty-aware allocation, partial rollouts with resumption, and sampling-only inference enhancements). Group candidates are typically *scored downstream* by Filter, and compute allocation may be coupled to Control.

Table 4 summarizes the primary trade-offs across topology/interaction, guidance/scaffolding, and sampling/exploration choices used in Generate.

4.2 Topology & Interaction

The topology of a rollout determines how the policy explores the solution space and how feedback signals are distributed across generated traces. We identify four principal topological categories that span current practice. **Linear rollouts** sample one complete trajectory per prompt (per update), typically applying an outcome reward once the sequence terminates; this is the dominant paradigm in early code and reasoning RL. **Group rollouts** sample K independent candidates in parallel from the same prompt, enabling

Table 4: **Generate design space and primary trade-offs.** Rows summarize rollout-construction choices used in representative methods, emphasizing the main efficiency/quality trade-off each choice introduces.

Subcomponent	Design choice	Primary trade-off	Representative works
Topology & interaction (Topo)			
Topology	Linear (single rollout)	Simplicity↑; within-prompt baseline×; gradient variance↑.	(Le et al., 2022; Liu et al., 2023).
Topology	Group (K rollouts)	Variance↓ via within-prompt baselines; compute↑ with K.	(Shao et al., 2024; Guo et al., 2025b; Yu et al., 2025a).
Topology	Tree/graph (branching prefixes)	Prefix reuse↑ and targeted exploration; control/bookkeeping↑.	(Hou et al., 2025; Bamba et al., 2025).
Interaction	Tool / environment loop	Grounded feedback↑; non-stationarity/latency↑ (env-dependent).	(Liu et al., 2023; Li et al., 2023; Jimenez et al., 2023).
Guidance & scaffolding (z)			
Seeding	ICL exemplars / rubrics	Early success and consistency↑ on hard prompts; risk of rubric/format bias transfer.	(Bamba et al., 2025; Zheng et al., 2023; Shi et al., 2025).
Conditioning	Plan-then-act / targets	Long-horizon structure↑; overhead/rigidity↑.	(Dou et al., 2025; Lin et al., 2025).
Nested rollouts	Reflection / critique / repair	Error recovery↑; tokens/latency↑.	(Zheng et al., 2025d; Yu et al., 2025c).
Adaptive guidance	Adaptive corrective guidance	Stability↑ under failure modes; extra guidance-generation/tuning overhead.	(Guo et al., 2025c; Zhang et al., 2025j; Chen et al., 2025c).
Sampling & exploration (κ_G)			
Sampling	Group sampling and response filtering	Signal quality↑ (avoid zero-gradient/low-value samples) with higher train-time sampling cost.	(Yu et al., 2025a; Shrivastava et al., 2025).
Partial rollouts	Resume unfinished samples	Throughput↑ under heavy-tailed lengths; resumption/bookkeeping↑.	(Zhou et al., 2025b).
Uncertainty signals	Uncertainty-aware sampling / weighting	Sample efficiency↑ on boundary prompts; estimator overhead and potential tail under-training.	(Jiang et al., 2025b; Hu et al., 2025b; Chen et al., 2025b).
Adaptive compute	Adaptive K / allocation / reuse	Budget efficiency↑ via per-prompt allocation and reuse; scheduler/buffer complexity↑.	(Nguyen et al., 2026; Zhang et al., 2025l; Bamba et al., 2025).
Sampling-only	No-update baselines	Test-time gains without training; no learning signal (no weight updates).	(Karan & Du, 2025).

within-group comparisons, majority voting, and relative-advantage estimation, which underpins GRPO-style methods. **Tree and graph rollouts** branch at intermediate prefixes, attach verifier or partial signals to internal nodes, and allocate rollout budget via expansion and pruning. Finally, **multi-turn and agentic rollouts** instantiate an action–observation loop, interleaving model actions with environment observations (tool outputs, API responses, execution traces, or simulator feedback) and requiring the policy to reason over stateful, multi-step interaction.

Linear single-trajectory rollouts Linear rollouts generate a single trajectory per prompt, representing the simplest topology and the inherited default from classic policy gradient methods. In this setting, the policy samples one complete response, receives a terminal reward, and updates without within-prompt comparisons. Variance reduction becomes critical since no peer responses exist to form relative baselines.

In code generation, CodeRL (Le et al., 2022) trains an actor-critic framework where a critic network predicts functional correctness and provides dense feedback for the actor. RLTF (Liu et al., 2023) extends this approach with an online generation framework and multi-granularity unit test feedback, providing finer credit assignment than coarse episode-level rewards. These examples illustrate how linear rollouts can still produce useful training signals when coupled with richer evaluators and feedback extraction.

Group rollouts (parallel candidates) Group rollouts sample K independent candidates per prompt, enabling within-prompt variance reduction without a learned critic. Rather than comparing a single response against a value estimate, the model uses the group itself as a reference distribution, computing advantages through normalization: $\hat{A}_i = (r_i - \mu_G)/\sigma_G$, where μ_G, σ_G are the mean and std of rewards within the group.

GRPO (Shao et al., 2024) introduced this paradigm, and DeepSeek-R1 (Guo et al., 2025b) demonstrated that applying it directly to base models can elicit emergent reasoning behaviors. Practical extensions address

common failure modes. DAPO introduces *decoupled clipping* and a token-level policy-gradient loss, and uses *dynamic sampling* to avoid zero-gradient updates when within-group rewards are uniform (all correct or all incorrect) (Yu et al., 2025a). Length inflation, where models produce increasingly verbose responses, is controlled by filtering on response length or reward-per-token ratio (Shrivastava et al., 2025). Dense step-wise rewards extend the paradigm to credit intermediate reasoning steps in multimodal settings (Zhang et al., 2025f).

Many recent methods retain prompt-level group sampling while modifying rollout curation, exploration, or learning signals, including GRPO-Lead (Zhang & Zuo, 2025), Not All Rollouts are Useful (Xu et al., 2025), XRPO (Bamba et al., 2025), G^2RPO-A (Guo et al., 2025c), Scaf-GRPO (Zhang et al., 2025j), SEED-GRPO (Chen et al., 2025b), and Stepwise Guided Policy Optimization (SGPO) (Chen et al., 2025c). Related formulations for group rollouts focus on how the *within-group baseline* is computed. For example, leave-one-out baselines (RLOO) use the other samples in the same prompt-level group as a reference signal (Ahmadian et al., 2024). On the analysis side, recent theoretical work characterizes GRPO's effective loss/dynamics for binary verifiable rewards and discusses its success-amplification behavior (Mroueh, 2025).

Tree/graph rollouts (branching search) Tree/graph rollouts generalize group sampling by branching at intermediate prefixes and reusing shared prefixes as internal nodes. Leaves receive outcome or verifier rewards, and these signals are backed up to internal nodes to obtain step- or segment-level supervision and to allocate rollout budget via expansion and pruning. TreeRPO, for example, uses tree sampling and bottom-up reward propagation to estimate step-level rewards without a separate process reward model, then computes advantages over step-level groups within the tree (Yang et al., 2025c). TreeRL couples on-policy tree search with intermediate process supervision to allocate exploration across branches under a fixed token budget (Hou et al., 2025). Additional tree/graph rollout variants for verifiable-reward post-training include heuristic tree modeling for policy optimization and inference (Li et al., 2025d), MCTS-style branching and backup to overcome RLVR bottlenecks (Wu et al., 2025a), off-policy tree-guided advantage optimization (Huang et al., 2025a), and tree-search formulations for agent RL (Ji et al., 2025). Related designs study tree-structured credit assignment (Tran et al., 2025), lookahead tree rollouts for trajectory-level exploration (Xing et al., 2025), and tool-use training with rewarded trees (Wu et al., 2025b), while some approaches bridge SFT and RL via branched rollouts from expert anchors (Zhang et al., 2025k). Analogous branching rollouts also appear in diffusion/flow post-training (Li et al., 2025e; Ding & Ye, 2025; Fu et al., 2025). Tree/graph rollouts shift compute from independent full trajectories to *structured reuse*: shared prefixes amortize KV-cache and reduce redundant computation, while branching concentrates exploration on uncertain steps. The trade-off is higher control complexity (expansion/pruning policies, node bookkeeping) and additional design choices for how to back up sparse leaf outcomes into dense process credit.

Multi-turn, tool-using, and multi-agent rollouts Multi-turn rollouts generalize single completions by placing the policy in an action–observation loop: at each step the model emits an action (tool call, edit, navigation, message) and conditions on the resulting observation (tool output, execution trace, retrieved context), with state carried across turns. Learning signals are often sparse and delayed, so practical methods either (i) rely on environments with verifiable endpoints such as unit tests or task success, or (ii) construct denser per-step credit from interaction traces. SWE-Gym operationalizes software engineering as an executable environment and trains agents and verifiers on collected trajectories (Pan et al., 2024), while WebRL targets web interaction and uses online curriculum RL to cope with sparse feedback in browser tasks (Qi et al., 2024); related work trains repository-level agents with a minimal tool API (e.g., a jump-to-definition tool) to navigate codebases end-to-end with RL (Zhang et al., 2026b). On the optimization side, GiGPO extends group-based objectives to long-horizon interaction via *anchor-state grouping* (identifying repeated states across trajectories) to form step-level groups and provide localized credit without a critic (Feng et al., 2025). RAGEN (Wang et al., 2025f) studies multi-turn RL dynamics in LLM agents, Search-R1 (Jin et al., 2025b) introduces multi-turn rollout with a search engine, and VerlTool (Jiang et al., 2025a) provides a tool-use interface for agentic RL pipelines. Finally, multi-agent rollouts treat the observation stream as strategic: ARLAS frames indirect prompt injection defense as a two-player game with adversarial attackers (Wang et al., 2025g), and related work incorporates simulated users or self-play loops to generate interactive training experience (Zhao et al., 2025b; Wei et al., 2025).

4.3 Guidance & scaffolding

ICL seeding and structured rubrics Guidance can be injected at rollout start via ICL exemplars, reasoning templates, or explicit evaluation rubrics, which biases the proposal distribution toward more structured trajectories. This perspective naturally connects to LLM-as-a-Judge: MT-Bench highlights that judgment outcomes can depend strongly on the judge prompt design and evaluation protocol (Zheng et al., 2023), while subsequent analyses identify systematic biases, such as position bias, that arise from prompt structure (Shi et al., 2025), motivating rubric-aware mitigation strategies summarized in recent surveys (Gu et al., 2024). Beyond evaluation, such seeding can also be used inside RL training to break *all-failed* regimes: XRPO (Bamba et al., 2025) allocates rollout budget to ICL-augmented prompts by injecting curated exemplars (e.g., solved examples) when base rollouts yield no successes, explicitly aiming to break zero-reward symmetry.

Plan- or answer-conditioned rollouts Rollouts can be guided by conditioning on explicit targets, such as high-level plans, imposing top-down constraints on trajectory generation. Separately, some methods introduce reference-answer-guided signals during training that bias the rollout distribution toward desired outcomes without requiring a fixed intermediate trace. Such target-conditioned guidance provides a global objective signal that reduces search ambiguity in long-horizon reasoning, complementing local or step-level guidance mechanisms. Plan-then-act methods first generate a global plan and then execute reasoning steps under this scaffold, using the plan to restrict and organize subsequent exploration (Dou et al., 2025), while reference-answer-guided approaches use known answers (or answer-derived signals) during training to steer the distribution over reasoning trajectories while allowing flexibility in intermediate reasoning (Lin et al., 2025).

Reflection and self-critique as sub-rollouts Reflection-based guidance treats critique or repair not as post-processing, but as an integral part of the rollout itself. In this view, reasoning is augmented with explicit evaluation or revision stages that form nested sub-rollouts, each with its own acceptance or scoring signal, allowing the model to reassess and refine intermediate trajectories before finalization. Critic-CoT (Zheng et al., 2025d) exemplifies this pattern by introducing an explicit chain-of-thought critic that evaluates and revises intermediate reasoning prior to producing a final answer. Related work further shows that such self-generated critiques can serve as effective training signals, improving reward modeling quality when incorporated into RL pipelines (Yu et al., 2025c). More generally, even in the absence of an explicit verifier, some methods derive rewards from model-internal signals. RLPR (Yu et al., 2025b), for example, uses the model's intrinsic probability of reference-answer tokens to construct a verifier-free training signal.

Guided rollouts with adaptive guidance strength Adaptive guidance views assistance as a dynamic control signal during rollouts: guidance is increased when the policy stalls or collapses into zero-reward regimes, and gradually reduced as competence improves. Rather than enforcing fixed expert trajectories, these methods aim to provide minimal, temporary support that restores learning signals while preserving on-policy exploration and autonomy. G^2RPO-A (Guo et al., 2025c) instantiates this idea by adaptively modulating guidance strength within GRPO based on training dynamics, injecting assistance when needed and fading it as competence improves. Scaf-GRPO (Zhang et al., 2025j) further formalizes adaptive guidance as a scaffolded curriculum, activating hierarchical hints only for true-hard problems and fading them as the model internalizes the required skills. At a finer granularity, Stepwise Guided Policy Optimization (Chen et al., 2025c) targets the *all-negative* regime by using a step-wise judge model to incorporate response diversity within groups, enabling learning even when initial groups contain no correct samples .

Tool augmentation and execution traces Tool-augmented rollouts treat external tools as first-class components of the trajectory: tools are invoked mid-rollout for execution, retrieval, or analysis, and their inputs and outputs become part of the reasoning trace. This expands the rollout space from pure text generation to interactive trajectories grounded in executable environments, enabling more reliable evaluation and learning signals. RLTF (Liu et al., 2023) exemplifies this paradigm by incorporating unit test execution results into rollouts, using test feedback as a structured reward signal for reinforcement learning in code generation tasks. Similarly, benchmarks and evaluation suites such as BIRD (Li et al., 2023)

(SQL executed against databases) and SWE-Bench (Jimenez et al., 2023) (patches validated by running test harnesses) operationalize execution-grounded feedback, making execution traces and tool outputs natural filtering signals.

4.4 Sampling & exploration policy

Sampling and decoding policies determine how the model explores the trajectory space, shaping both the diversity of candidate rollouts and the efficiency of the learning signals derived from them. Local decoding choices modulate token-level randomness, while uncertainty-aware sampling makes exploration input-adaptive by allocating compute preferentially to informative prompts. In addition, some methods modify the sampling policy without updating model weights, and others use LLM-generated hints to guide exploration in external environments, so we include them here because they directly influence the rollout distribution.

Decoding and diversity knobs Stochastic decoding mechanisms, including temperature scaling, top-p sampling, and diversity penalties, modulate the entropy of the token-level proposal distribution and thereby shape exploration in the trajectory space. Higher entropy encourages diversity, although it also increases variance and may produce excessively long or incoherent generations. This has motivated approaches that manage how diversity is realized during rollout generation, so that additional randomness contributes an informative exploratory signal rather than computational waste. APRIL (Zhou et al., 2025b) addresses inefficiency arising from long-tailed rollout length distributions by over-provisioning rollout requests and stopping generation once a target number of samples has completed. Unfinished rollouts are resumed in later iterations, which preserves data while implicitly reshaping the realized sampling distribution and improving throughput without degrading task performance. In parallel, TreeRL (Hou et al., 2025) reallocates exploration within a rollout by using on-policy tree search instead of independent chains, enabling branching under a fixed token budget. These methods indicate that diversity control in reinforcement learning for large language models involves not only token-level sampling parameters, but also the organization and allocation of exploratory behavior across rollouts.

Uncertainty-aware sampling and exploration signals Uncertainty can be used during rollout generation to prioritize informative prompts and allocate sampling compute more effectively. In group-based policy gradient training, prompts whose responses are consistently correct or consistently incorrect contribute little gradient signal, while prompts near the model decision boundary tend to be more informative. Recent work therefore estimates uncertainty from reward outcomes or semantic disagreement and introduces it at different stages of the rollout pipeline.

One line of work uses prompt-level uncertainty to guide sample selection and curriculum scheduling, without explicitly modifying the number of rollouts per prompt. VCRL (Jiang et al., 2025b) treats reward variance as a proxy for difficulty and prioritises prompts with intermediate variance, which tend to lie near the model's decision boundary. VADE (Hu et al., 2025b) models prompt difficulty with per-sample Beta distributions and uses Thompson sampling to select prompts that maximize an information-gain objective. MMR1 (Leng et al., 2025) extends these ideas to multimodal RL, combining reward variance and trajectory diversity to stabilise GRPO training. Across these approaches, reward-side variance functions as a practical signal of epistemic uncertainty, encouraging learning on prompts that are difficult but not hopeless.

A complementary direction uses uncertainty to adapt the allocation of rollout compute itself, so that ambiguous prompts attract additional samples while easy or saturated ones are deprioritised. Adaptive rollout allocation can be framed as an expected-gradient-variance minimization problem under a fixed compute budget, assigning more samples to prompts expected to reduce estimator noise (Nguyen et al., 2026). XRPO (Bamba et al., 2025) introduces targeted exploration/exploitation mechanisms, including ICL seeding for zero-reward prompts and an uncertainty-reduction-motivated rollout allocator under a fixed compute budget. AR3PO (Zhang et al., 2025l) adds response reuse to this paradigm, leveraging previously generated correct responses while adaptively allocating additional samples to difficult prompts.

Finally, some work focuses less on the amount of sampling and more on the quality of the exploration signal. SEED-GRPO (Chen et al., 2025b) measures semantic entropy, meaning the diversity of meanings expressed

across candidate outputs, as an indicator of epistemic uncertainty, and uses it to modulate update magnitude to avoid over-updating on uncertain cases. Unlike reward variance, semantic entropy captures ambiguity in reasoning structure rather than in scalar outcomes, providing a complementary view of uncertainty.

Sampling-only reasoning enhancements Sampling-only methods aim to improve reasoning purely at inference time by modifying the sampling policy without updating model parameters. These approaches leverage additional compute rather than supervision and therefore serve as practical rollout primitives and baselines. Power Sampling method (Karan & Du, 2025) approximates the distribution-sharpening effect of RL by sampling from a power distribution proportional to $p_\theta(y \mid x)^\alpha$ (for $\alpha > 1$), using an MCMC procedure to accept or reject proposed continuations. This concentrates probability mass on higher-probability reasoning traces, yielding gains without training-time supervision.

LLM-guided exploration for RL agents (cross-domain) In sparse-reward reinforcement-learning settings, large language models can supply semantic priors that bias exploration toward meaningful state–action regions rather than relying on uninformed stochastic search. Recent work operationalises this idea by conditioning rollouts on LLM-generated guidance while leaving policy optimization to the RL agent. ExploR-LLM (Ma et al., 2025) integrates foundation models by augmenting the agent's observation and by providing base and exploration policies, while the learned RL policy adapts by learning residual corrections. Complementarily, LLM-hints methods (Jain & Grossmann, 2025) inject LLM-generated action hints into the observation stream, enabling the policy to learn when to rely on or ignore external guidance. Across these approaches, semantically guided rollouts substantially improve exploration efficiency in sparse-reward domains while preserving policy autonomy.

5 Filter: From Rollouts to Learning Signals

5.1 Problem Formulation: Filter (F) — From Rollouts to Learning Signals

Using the global notation in §3.2, the **Filter** module maps sampled candidates $\mathcal{T}(x) = \{\tau^{(i)}\}_{i=1}^K$ into intermediate signals and optimizer-facing supervision used for pruning/selection, credit assignment, and compute allocation. In text-only settings we write $y^{(i)} \equiv u_{1:T}^{(i)}$ for the completion corresponding to $\tau^{(i)}$. Filter may operate on complete rollouts $\tau^{(i)}$ as well as prefixes $\tau_{\leq t}^{(i)}$ (e.g., step/segment-level scoring in long traces or during tree expansion).

Abstractly, Filter produces intermediate signals

$$\phi_i \;=\; F\big(\tau^{(i)}; \mathcal{T}(x)\big), \tag{7}$$

where ϕ_i may include validity indicators, verifier outcomes, process/step scores, judge preferences or ranks, and learning-value diagnostics such as disagreement, entropy, or novelty.

A convenient decomposition separates filtering into (i) gating, (ii) semantic evaluation, and (iii) mapping into training targets, possibly using the entire group:

$$m_i = \mathrm{Gate}(x, \tau^{(i)}) \in \{0, 1\}, \tag{8}$$

$$r_i^{\mathrm{raw}} = \mathrm{Eval}(x, \tau^{(i)}) \in \mathbb{R}, \tag{9}$$

$$(w_i,\, \tilde{r}_i,\, \tilde{A}_i,\, \ell_i) = \mathrm{Map}\Big(\{(m_j, r_j^{\mathrm{raw}}, \tau^{(j)})\}_{j=1}^K\Big). \tag{10}$$

In text-only settings, $\tau^{(i)}$ reduces to $y^{(i)} \equiv u_{1:T}^{(i)}$, and we may write $\mathrm{Gate}(x, y^{(i)})$ and $\mathrm{Eval}(x, y^{(i)})$.

Here m_i captures structural validity (format/schema/executability/safety), r_i^{raw} is a raw evaluation signal (e.g., deterministic verification correctness, judge score, or process-quality score), and $\mathrm{Map}(\cdot)$ converts raw outcomes into optimizer-facing supervision: sample weights w_i (masking/reweighting), shaped rewards $\tilde{r}_i$, advantages $\tilde{A}_i$, or discrete labels/preferences ℓ_i (pairwise or listwise). Group-dependent mappings include listwise judging, within-group normalization, abstention/tie handling, and aggregation of per-step scores into trajectory-level targets. (When deterministic checkers exist, Eval can be instantiated as a verifier.)

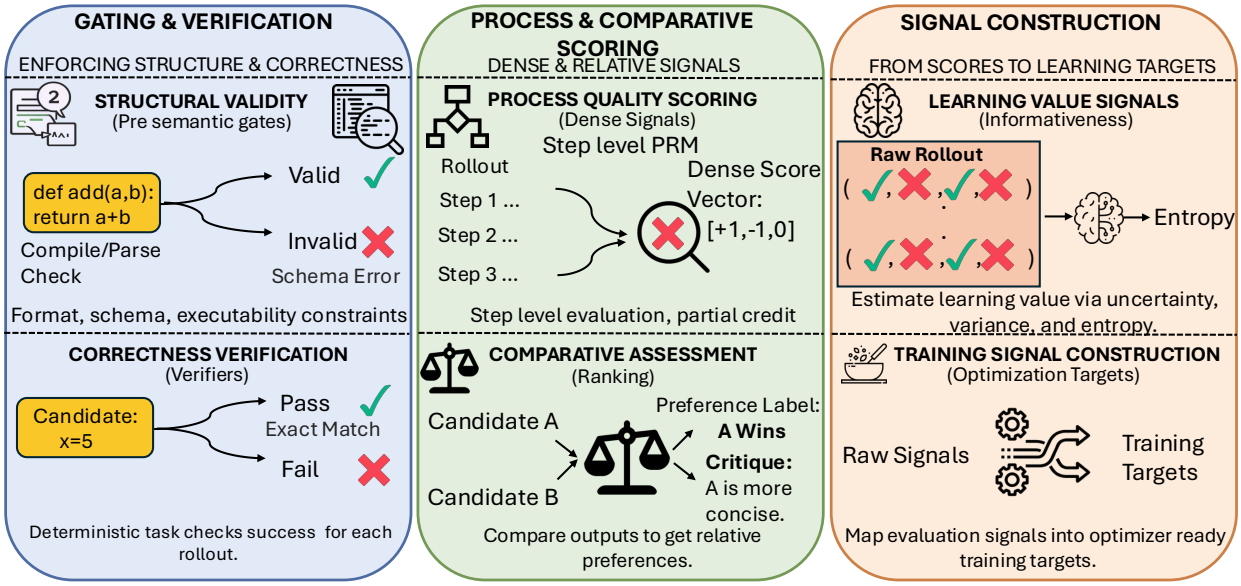

Figure 5: **Filter module design space.** Filter maps sampled rollouts (or rollout prefixes) into intermediate signals and optimizer-facing supervision. It can be organized along three components: *(left)* gating & verification (structural validity, schema/executability checks, deterministic correctness tests), *(middle)* process & comparative scoring (dense step-level process signals and relative judgments via ranking/judging), and *(right)* signal construction (learning-value diagnostics such as uncertainty/variance/entropy, and mapping raw outcomes into optimizer-ready targets such as masks/weights, advantages, or preference labels). Filter outputs are consumed by the optimizer and may also inform Control decisions.

At the highest level, we recover the training signal notation from §3.2 via

$$S(\tau^{(i)}; \mathcal{T}(x)) \;=\; \mathrm{Score}(\phi_i)\,, \tag{11}$$

with the understanding that S may correspond to $\tilde{r}_i$, $\tilde{A}_i$, ℓ_i, or a weighted objective defined by $(w_i, \cdot)$. This formulation cleanly distinguishes **Filter** (measurement and signal construction) from **Control** (budgeted decision rules that consume these signals).

Table 5: **Filter as a signal pipeline (Gate → Eval → Map).** This table instantiates Eqs. (8)–(10) with common rollout filters and the optimizer-facing artifacts they produce.

Stage	Output	Typical mechanisms	Representative works
Gate	$m_i \in \{0,1\}$	Pre-semantic validity checks: parseability/normalization, executability, safety constraints; can act on rollouts or prefixes.	(Chen et al., 2025a; Hugging Face, 2025; Liu et al., 2023; Le et al., 2022; Li et al., 2023).
Eval	$r_i^{\mathrm{raw}} \in \mathbb{R}$	Hard semantic verification (unit tests, execution, exact-answer equivalence), process/step scoring (PRMs/step verifiers), or comparative ranking (LLM judge).	(Liu et al., 2023; Le et al., 2022; Yao et al., 2026; Ali et al., 2025; Zhang et al., 2025g; Hendrycks et al., 2021; He et al., 2024; Lightman et al., 2023; Zheng et al., 2025b; Song et al., 2025; Zhang et al., 2025m; Zheng et al., 2023; Shi et al., 2025).
Map	$(w_i, \tilde{r}_i, \tilde{A}_i, \ell_i)$	Convert raw signals into optimizer-facing supervision: masks/weights, shaped rewards, group-relative advantages, pairwise/listwise labels; often group-dependent.	(Mroueh, 2025; Zhang & Zuo, 2025; Shrivastava et al., 2025; Becker et al., 2025).

5.2 Structural validity (pre-semantic gates)

Pre-semantic gating filters out rollouts that cannot be meaningfully scored *even if* their underlying intent is correct. In practice, verifiers are brittle to malformed outputs: a correct solution expressed with the wrong schema, an invalid tool call, or an unparsable final answer can be misclassified as incorrect and inject systematic noise into training. Structural validity gates therefore enforce constraints such as: required delimiters (e.g., separating *analysis* from *final*), single-answer policies, JSON/schema validity for tool outputs, argument typing for function calls, or syntactic constraints on programs and queries.

Recent verifier toolkits emphasize that *normalization and parsing* are first-order components of correctness. For example, xVerify (Chen et al., 2025a) focuses on efficient and standardized answer verification for reasoning evaluations, while Math-Verify provides practical utilities for robust mathematical parsing and comparison (Hugging Face, 2025). Although these tools are often presented for evaluation, in rollout-based RL they play a training-critical role: they reduce false negatives caused by superficial formatting mismatches and make the mapping from rollouts to scores more stable.

Structural gating is especially salient for execution-centric domains. In program synthesis and code generation, rollouts must at minimum compile/execute before unit-test feedback can be computed, as in RL from unit test feedback (Liu et al., 2023) and execution-guided code RL (Le et al., 2022). Similarly, for Text-to-SQL, a candidate query must be executable against the database before any semantic comparison is possible; benchmarks such as BIRD emphasize execution-grounded evaluation for realistic text-to-SQL (Li et al., 2023), motivating pipelines that treat executability as a first-stage filter before applying execution-based rewards (e.g., (Yao et al., 2026; Ali et al., 2025)). In short, pre-semantic gates define the *eligible rollout set* on which meaningful verification and learning can occur.

5.3 Correctness verification (hard semantic gates)

Hard semantic gates assign rollouts task-grounded correctness signals—typically pass/fail or a bounded scalar score—using deterministic procedures whenever possible. This paradigm is central to verifiable reinforcement learning (RLVR): rather than training a learned reward model, the system relies on checkers that are (relatively) objective, cheap, and difficult to exploit.

Execution-based verification. In code tasks, a natural verifier executes the generated program and computes success from unit tests. RLTF uses unit-test outcomes as feedback signals for learning (Liu et al., 2023), and CodeRL demonstrates that execution-based rewards can effectively guide policy improvement in code generation (Le et al., 2022). The rollout strategy and verifier are tightly coupled: one typically samples multiple candidate programs, discards non-executing ones, and reinforces programs that pass more tests.

Text-to-SQL follows an analogous pattern. Since SQL has a well-defined execution semantics, correctness can be assessed by running the query against the target database and comparing results or using execution success as a proxy. BIRD highlights the importance and difficulty of execution-grounded SQL evaluation in practice (Li et al., 2023). Recent RL-style SQL systems explicitly leverage these hard verifiers, including Arctic-Text2SQL-R1, which shows strong SQL reasoning from simple, execution-grounded rewards (Yao et al., 2026), as well as other execution-based reinforcement approaches (Ali et al., 2025). These systems treat verification not as a post-hoc metric, but as the mechanism that determines which rollouts are considered correct enough to drive updates. ExeSQL strengthens this paradigm for realistic deployments by targeting SQL dialect differences and using execution to filter candidates and construct preference-training signals for Text-to-SQL (Zhang et al., 2025g).

Exact-answer verification for math. For many math datasets, correctness can be defined by exact-answer checks with careful normalization and parsing, enabling scalable pass/fail reward signals. MATH provides a large set of competition-style problems (Hendrycks et al., 2021), and OlympiadBench further tests high-difficulty mathematical reasoning (He et al., 2024). In these settings, the verification stage often reduces to robustly extracting the final answer and checking equivalence, which again highlights the importance of parsing infrastructure such as xVerify (Chen et al., 2025a) and Math-Verify (Hugging Face, 2025) for avoiding spurious failures.

Multimodal settings and *making verification possible.* Compared to text-only domains, multimodal reasoning often lacks universal deterministic checkers. Recent work, therefore, either (i) designs objectives with more verifiable structure or (ii) constructs synthetic pipelines that retain programmatic checkability. Vision-R1 and VLM-R1 study R1-style reinforcement learning with verifiable rewards for multimodal models; VLM-R1 in particular emphasizes training stability and generalization under visual complexity (Huang et al., 2025b; Shen et al., 2025). For video spatial reasoning, SpaceR similarly relies on rule-based verifiable rewards to provide reliable training signals (Ouyang et al., 2025). A complementary direction is to expand the space of tasks with reliable filters through verifiable synthesis: SynthRL scales visual reasoning by generating data with built-in verifiability (Wu et al., 2025d). GRACE (Sun et al., 2025a) introduces a verification reward based on contrastive learning. Across these works, the hard-gate verifier (or its proxy) determines which parts of the rollout distribution are learnable under RL.

5.4 Process quality scoring (dense or step-level signals)

Beyond binary acceptance or rejection, *process quality scoring* assigns dense signals along a rollout per step, segment, or node, enabling early pruning, partial credit, and better credit assignment. A central distinction is between outcome reward models ORMs that score only the terminal answer, and process reward models PRMs that evaluate intermediate reasoning steps (Liu et al., 2025a). Early empirical evidence on GSM8K shows why this matters—optimizing only for final answer correctness can leave substantial trace error, motivating supervision or learned rewards that track step correctness (Uesato et al., 2022). PRM style supervision can be instantiated directly via human step labels, as in PRM800K, where annotators mark individual steps, for example, positive, neutral, negative, to train step-level reward or prediction heads (Lightman et al., 2023). Complementarily, automated step verifiers can provide process feedback without human annotations, for example, by verifying intermediate reasoning steps and using them as reinforcement signals (Wang et al., 2023). In rollout systems, these step-level scores can be aggregated, for example, minimum or first failure, discounted sum, or calibrated averages to decide whether to continue expanding a branch and to shape advantages for downstream optimization (Liu et al., 2025a).

A key challenge is *evaluating* whether dense scores truly reflect reasoning quality rather than correlating with surface heuristics. Zheng et al. (2025b) address this with ProcessBench, which measures a model's ability to identify the *earliest erroneous step* in mathematical solutions annotated by experts. This reframes PRM assessment from indirect end task improvements, for example, best-of-K reranking, to direct error localization and calibration, and it highlights a recurring pathology: on harder problems, models can reach correct final answers while still containing incorrect intermediate steps, making terminal-only rewards an unreliable proxy for process fidelity (Zheng et al., 2025b; Uesato et al., 2022). Practically, these findings justify deploying process scorers as filters inside the rollout, not just as final rerankers, for example, to cut off low-quality branches early, allocate more budget to uncertain steps, or route suspect segments to stronger critics (Liu et al., 2025a).

Two complementary lines of work further sharpen the evaluation story. PRMBench introduces a fine-grained benchmark designed specifically to probe whether PRMs assign correct step-level signals under challenging failure modes (Song et al., 2025). Separately, The Lessons of Developing Process Reward Models in Mathematical Reasoning (Zhang et al., 2025m) reports practical pitfalls in PRM development and evaluation, emphasizing that seemingly reasonable estimation or selection procedures can yield misleading conclusions if the benchmark or protocol is not carefully controlled.

Dense scoring can also be induced through *reference-based* verification systems that convert task-specific checkers or verifiers into step- or node-level feedback. Yan et al. (2025) benchmark such reference-based reward systems and stress-test robustness, while Chen et al. (2025a) provides a verifier designed to improve answer extraction or equivalence checking, crucial whenever correctness depends on parsing or semantic normalization rather than exact match. On the optimization side, recent RL methods explicitly operationalize step-level rewards. R1-VL (Zhang et al., 2025f) introduces step-wise, verifier-driven reward shaping for multimodal reasoning, converting terminal evaluation into denser step-level signals to mitigate sparse feedback. TreeRL (Hou et al., 2025) uses on-policy tree search with intermediate supervision to propagate evaluative signals to internal nodes. Rule-based verifiers (e.g., Math Verify) and tree-sampling credit-assignment variants (e.g., TreeRPO (Yang et al., 2025c)) convert sparse task feedback into dense rollout control signals.

5.5 Comparative assessment (relative ranking signals)

Comparative assessment evaluates candidate rollouts through relative judgments rather than absolute scores. Given multiple responses for the same prompt, an evaluator outputs preferences among candidates, which is useful when correctness is ambiguous, when goals are preference aligned, or when selection must be robust across diverse samples. This mechanism naturally acts as a filtering signal that can later be converted into supervision for training or used directly for pruning and selection (Zheng et al., 2023).

A common approach uses an LLM as a judge to compare candidates, but such judgments can be noisy and systematically biased. MT Bench and Chatbot Arena document sensitivity to answer ordering, preference for longer outputs, and weaknesses on reasoning and mathematics (Zheng et al., 2023). Judging the Judges analyzes both pairwise and listwise comparison settings and recommends protocols that improve reliability, including repeated judgments, order permutations, and explicit tie options (Shi et al., 2025). Complementary work studies how feedback protocols (e.g., pairwise vs. pointwise) interact with bias in LLM-based evaluation (Tripathi et al., 2025), with broader discussion summarized in recent surveys (Gu et al., 2024). Incorporating ties and abstention is also important when differences are not decisive, since models may otherwise produce overconfident rankings, and abstention behavior can degrade under reasoning focused fine-tuning (Shi et al., 2025; Kirichenko et al., 2025). Recent work also attempts to *improve* judge reliability directly at inference time by leveraging the judge's *distribution* over judgment tokens rather than relying on a single greedy decision (Wang et al., 2025d). For broader context and taxonomy, see recent surveys of LLM-as-a-judge (Li et al., 2025a).

Comparative assessment can be strengthened by attaching critiques that justify preferences and help identify failure modes. Critic CoT (Zheng et al., 2025d) demonstrates the value of using critique-based filtering before aggregation, while Critic RM (Yu et al., 2025c) uses critique generation and consistency filtering to improve downstream reward modeling.

5.6 Learning-value signals (informativeness, not correctness)

Beyond assigning scores based on semantic correctness, a distinct class of filtering signals assesses the *informativeness* or *learning value* of a rollout or prompt. These signals aim to maximize training progress by identifying data points where the model's current policy is uncertain, where reward signals are most variable, or where additional exploration is likely to yield novel, generalizable behaviors. This paradigm is crucial for overcoming the gradient-vanishing problem common in group-based RL such as GRPO, where uniform rewards (all correct or all incorrect) within a group lead to zero advantage and thus no learning signal.

A primary strategy is to use the *variance of group rewards* as a real-time, adaptive proxy for sample difficulty and learning potential. The core insight is that prompts yielding a mix of correct and incorrect rollouts, feature high reward variance, lie near the model's current capability boundary and provide the strongest contrastive gradients. VCRL (Jiang et al., 2025b) formalizes this by constructing a curriculum that dynamically prioritizes prompts with high normalized reward variance, using a replay buffer to maintain a pool of high-value samples. Similarly, DAPO (Yu et al., 2025a) employs dynamic sampling, discarding prompts with uniform rewards and oversampling until a batch with informative variance is assembled, ensuring every training step receives non-zero gradients. Extending this to a non-stationary bandit formulation, VADE (Hu et al., 2025b) performs online, sample-level difficulty estimation using a Beta distribution posterior. It selects prompts via Thompson sampling to maximize an information gain objective, effectively balancing exploration and exploitation without extra rollout costs.

A complementary approach quantifies uncertainty through the *semantic entropy* of generated responses, measuring the diversity of meanings within a rollout group. SEED-GRPO (Chen et al., 2025b) uses this entropy to modulate advantage magnitudes: prompts eliciting semantically consistent responses that have low entropy and high confidence undergo standard updates, while those with diverse, contradictory outputs that have high entropy and high uncertainty receive attenuated updates, implementing a dynamic, uncertainty-aware learning rate. XRPO (Bamba et al., 2025) integrates multiple informative signals. Its hierarchical rollout planner allocates compute based on a priority score combining expected statistical un-

certainty reduction and an exploration bonus, while its novelty-guided advantage sharpening rewards correct but low-likelihood responses, pushing the policy boundary outward.

Efficiency-oriented methods focus on *reusing or down-sampling* rollouts to preserve information gain while reducing computation. AR3PO (Zhang et al., 2025l) introduces adaptive rollout that allocates more responses to hard prompts and response reuse that leverages past correct responses from a buffer, significantly lowering the average number of rollouts needed per training step. PODS (Xu et al., 2025) addresses the system-level asymmetry between parallelizable inference and memory-intensive policy updates by generating many rollouts but training only on an informative subset. Its max-variance down-sampling selects a group of rollouts that maximize reward variance within a group, which, for binary rewards, simplifies to choosing the half group of best and half group of worst, preserving strong contrastive signals at a fraction of the update cost.

Relatedly, Nguyen et al. (2026) studies online RLVR with a policy that dynamically allocates rollout budget based on observed learning signals, making compute allocation itself a learned, feedback-driven component of the training loop.

GRPO-LEAD (Zhang & Zuo, 2025) incorporates a difficulty-aware advantage reweighting mechanism. It estimates prompt difficulty via the empirical correctness ratio of its rollouts and applies a logistic weighting function to amplify advantages for challenging problems, ensuring the model focuses on refining its performance on harder tasks rather than over-optimizing on easy ones. Collectively, these learning-value signals transform the filtering stage from a passive correctness check into an active engine for directing exploration, stabilizing gradients, and maximizing the informational yield of each unit of rollout computation.

5.7 Training-signal construction (scores, labels, advantages, or weights)

Even when a hard verifier provides a clean score r_i^{raw}, training typically does not optimize directly on r_i^{raw}. Instead, systems transform verification outcomes into learning targets that improve optimization stability, reduce variance, and encourage useful exploration. This mapping is particularly important in rollout-heavy settings, where most samples may be incorrect and naïve credit assignment can lead to collapse.

Group-relative advantages from multiple rollouts. A widely used strategy is to sample multiple rollouts per prompt and compute *within-group* baselines to normalize difficulty. For example, one can define an advantage by subtracting the mean score among valid rollouts:

$$\tilde{A}_i \;=\; m_i\left(r_i^{\text{raw}} - \frac{\sum_{j=1}^{K} m_j\, r_j^{\text{raw}}}{\sum_{j=1}^{K} m_j + \epsilon}\right), \tag{12}$$

where $\epsilon > 0$ is a small constant for numerical stability.

So updates depend on *relative* performance among candidate trajectories for the same x. Analyses of GRPO objectives clarify how such group-based losses behave and how rollout sets influence the effective optimization signal (Mroueh, 2025). When exploration diminishes (e.g., low-entropy rollout sets), the mapping from r_i^{raw} to $\tilde{A}_i$ becomes even more consequential; GRPO-LEAD (Zhang & Zuo, 2025) studies advantage reweighting strategies aimed at maintaining learning progress under such conditions.

Filtering, shaping, and verbosity control. Beyond baselines, the mapping stage can incorporate additional shaping terms and selective filtering. GFPO (Shrivastava et al., 2025) exemplifies group filtering paired with concision-aware signal design, discouraging pathological verbosity while preserving correctness-driven learning. XRPO (Bamba et al., 2025) provides another perspective on constructing training signals that balance targeted exploration and exploitation using rollout-derived supervision. These works reflect a broader trend: rollout strategy is not only about how to sample, but also how to *value* and *weight* samples once obtained.

Conservative optimization under aggressive filters. Hard gating can produce sparse effective batches (few surviving rollouts), which increases gradient variance and can destabilize training. Trust-region or

conservative-update principles help mitigate this by limiting how far the policy moves per iteration under noisy signals. TROLL represents this direction, emphasizing stability of policy updates in the presence of filtered or high-variance rollout supervision (Becker et al., 2025).

When hard verifiers are missing: constructing surrogate signals. Many real tasks lack deterministic correctness checks, so recent work explores how to retain the benefits of rollout filtering while broadening applicability. Su et al. (2025) argues for expanding verifiable-style training across more diverse domains, and Generalizing Verifiable Instruction Following studies how to extend verifiable objectives beyond narrowly rule-based settings (Pyatkin et al., 2025). Other efforts explicitly address learning without reliable external verifiers, including Reinforcing General Reasoning without Verifiers (Zhou et al., 2025a), RLPR (Yu et al., 2025b), and Zero Reinforcement Learning (Zeng et al., 2025). In these settings, the *filter* often becomes a hybrid of weak checks, self-consistency signals, or structured constraints, and the signal-construction stage carries more responsibility for stabilizing learning and preventing reward hacking.

6 Control: Compute Allocation, Decision Rules, and On/Off-Policy Knobs

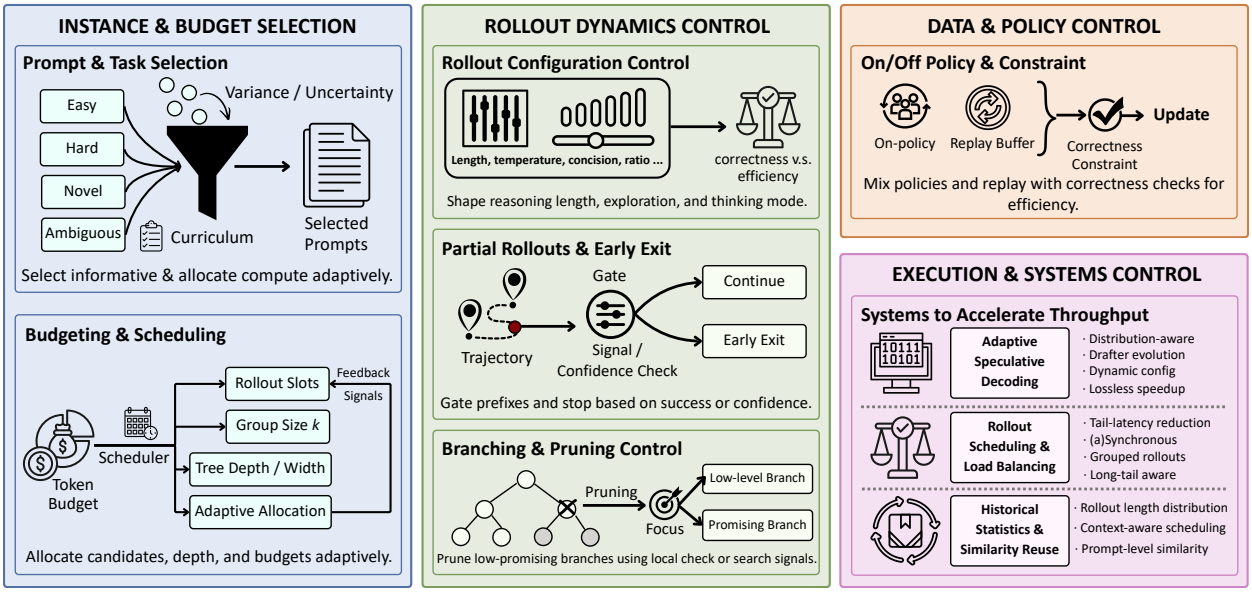

Figure 6: **Control module design space.** The control layer of rollout pipelines can be organized into four orthogonal components: *(left)* Instance & Budget Selection (prompt prioritization by difficulty/novelty/uncertainty; adaptive allocation of rollout slots, group size, depth/width, and token budgets); *(middle)* Rollout Dynamics Control (decoding configuration, partial rollouts with early exit, and branching–pruning policies for quality–efficiency trade-offs); *(right-top)* Data & Policy Control (on-/off-policy mixing, replay, and correctness-aware filtering); and *(right-bottom)* Execution & Systems Control (scheduling, speculative decoding, load balancing, and statistics reuse). Together, these mechanisms manage compute allocation and execution under fixed budgets, defining the trade-off between accuracy, efficiency, and cost independently of the underlying optimizer.

6.1 Problem Formulation: Control (C) — Compute Allocation and Decision Rules

Using the global notation in §3.2, the **Control** module governs *where to spend rollout compute and what to do next* under budgets: which prompts/tasks to roll out on, how many candidates to generate, when to stop or continue, when to branch or prune, and how to set rollout configuration (e.g., max length, temperature, concision). Control turns intermediate Filter signals and cost accounting into allocation decisions, thereby shaping the realized rollout-group distribution $q_{\theta,\mathrm{GFCR}}(\mathcal{T} \mid x, \mathcal{B})$.

Control can be viewed as a budgeted sequential decision process operating over an evolving set of partial rollouts. For a prompt x and an active frontier of prefixes $\{\tau_{\leq t}^{(i)}\}$, let $\phi_{i,t} := \phi(\tau_{\leq t}^{(i)}; \mathcal{T}(x))$ denote the associated Filter signal(s) available at time t (e.g., step scores, validity flags, verifier outcomes). Control observes the frontier, these signals, accumulated cost, and remaining budget b (initialized as B_x), and selects an action

$$a \sim C_\psi(\cdot \mid x, \{\tau_{\leq t}^{(i)}\}, \{\phi_{i,t}\}, b), \tag{13}$$

where a may encode: (i) prompt/task selection across instances, (ii) adaptive sampling decisions (e.g., choosing K, depth/width, or selective rollout), (iii) continuation/stop rules (early exit, partial rollouts, selective continuation), (iv) branch/prune control in tree/agentic rollouts (which prefixes to expand and which subtrees to discard), and (v) rollout configuration control (length/temperature/concision and sampling/selection rules that affect the mix of positive vs. informative-negative trajectories). These actions update the frontier, determine additional token/tool expenditure, and ultimately determine the terminal rollout group $\mathcal{T}(x)$ passed to downstream learning-signal construction.

At the pipeline level, Control aims to maximize learning utility per unit compute by shaping which rollouts are completed and which are abandoned:

$$\max_{\psi} \;\; \mathbb{E}_{x\sim\mathcal{D}}\, \mathbb{E}_{\mathcal{T}\sim q_{\theta,\mathrm{GFCR}}(\cdot|x,\mathcal{B};\psi)}[U(\mathcal{T})] \quad \text{s.t.} \quad \mathbb{E}\left[\sum_{\tau\in\mathcal{T}} c(\tau)\right] \leq B_x, \tag{14}$$

where $U(\mathcal{T})$ summarizes the downstream training value induced by Filter (e.g., usable-sample mass, expected signal strength, or other signal-quality surrogates), and the constraint enforces per-prompt compute limits (with an analogous global constraint B). Finally, Control includes on/off-policy knobs that govern how training mixes fresh on-policy rollouts with reused/off-policy data (often mediated by Replay), as well as systems-level scheduling/token accounting decisions that treat throughput and long-tailed rollout lengths as first-class control problems. Table 6 summarizes the method landscape covered across this entire section.

6.2 Prompt and task selection (what do you roll out on?)

An important but sometimes overlooked design choice in reinforcement learning based post training is deciding which prompts should be used to generate rollouts. Early work often relied on uniform sampling from the training distribution. However, several recent studies show that many prompts contribute little useful training signal, which leads to wasted computation. GRESO (Zheng et al., 2025c) targets this inefficiency by observing that when all samples in a rollout group receive identical rewards, the within-group reward variance becomes zero and GRPO-style advantages collapse, yielding near-zero policy-gradient signal. Based on training dynamics, GRESO (Zheng et al., 2025c) learns to predict such *zero-variance* prompts and probabilistically skips them before rollout while retaining exploration. VCRL (Jiang et al., 2025b) takes a complementary view by treating group reward variance as a proxy for sample difficulty: prompts that are too easy or too hard tend to have low variance, while intermediate-difficulty prompts produce higher variance and stronger learning signal, inducing an implicit curriculum as the policy improves.

Other work has explored uncertainty driven prompt selection using probabilistic modeling. VADE (Hu et al., 2025b) estimates per-prompt correctness online (via Beta posteriors) and applies Thompson sampling to favor prompts expected to be informative while still exploring. In contrast, SEED-GRPO (Chen et al., 2025b) does not explicitly decide which prompts to sample; instead, it measures semantic entropy (meaning diversity) across multiple answers to a prompt and uses this uncertainty signal to modulate the magnitude of policy updates, applying more conservative updates on high-uncertainty prompts.

SEC (Chen et al., 2025e) moves further away from individual prompt level selection and instead learns a distribution over prompt categories. It formulates curriculum selection as a non stationary multi armed bandit problem, where each category such as difficulty level or task type is treated as an arm; the absolute advantage acts as the reward signal and the curriculum policy is updated (e.g., via TD(0)) to maximize immediate learning gain. Taken together, these approaches reflect a broader shift toward viewing rollout allocation as an adaptive resource management problem rather than a fixed data sampling procedure.

Table 6: **Taxonomy of control mechanisms in rollout-based RL for reasoning LLMs.** Rows correspond to control loci in §6; columns specify the controlled decision variable, the signals that drive that decision, representative methods, and the operational objective/trade-off targeted by each family.

Control slice	Decision variable	Primary signals	Representative methods	Operational objective and trade-off
Prompt/task selection	Which prompts are rolled out	Group reward variance, posterior success uncertainty, semantic entropy, category-level learning gain	GRESO (Zheng et al., 2025c); VCRL (Jiang et al., 2025b); VADE (Hu et al., 2025b); SEED-GRPO (Chen et al., 2025b); SEC (Chen et al., 2025e)	Avoid low-information prompts and concentrate computation where policy updates are expected to be strongest.
Budgeting and scheduling	How many rollouts (width/depth) per prompt	Fixed-K groups vs. variance-aware and uncertainty-aware allocation, response reuse statistics	DeepSeekMath (Shao et al., 2024); 1-shot RLVR (Wang et al., 2025e); VIP (Nguyen et al., 2026); MMR1 (Leng et al., 2025); XRPO (Bamba et al., 2025); AR3PO (Zhang et al., 2025l)	Trade static stability for adaptive compute efficiency, often reallocating budget toward uncertain or harder prompts.
Rollout configuration control	Length, think/no-think mode, and token-efficiency of kept responses	SOL targets, decoupled token rewards, hybrid-mode gating, reward-per-token filters, length curricula	ShorterBetter (Yi et al., 2025); DECS (Jiang et al., 2025e); AdaptThink (Zhang et al., 2025e); LHRMs (Jiang et al., 2025c); GFPO (Shrivastava et al., 2025); Train Long, Think Short (Hammoud et al., 2025)	Reduce overthinking and token cost while preserving correctness through reward shaping and curriculum schedules.
Partial rollouts and early exit	Stop, continue, resume, or downsample ongoing rollouts	Prefix-level progress signals, completion counts, diversity/variance among sampled rollouts	S-GRPO (Dai et al., 2025); APRIL (Zhou et al., 2025b); PODS (Xu et al., 2025)	Cut long-tail rollout waste by terminating unproductive generation early and prioritizing informative trajectories for updates.
Branching and pruning (tree rollouts)	Where to branch and which subtrees to prune	Local uncertainty, search value estimates, and step-level reward propagation	TreeRL (Hou et al., 2025); TreePO (Li et al., 2025d); DeepSearch (Wu et al., 2025a); TreeRPO (Yang et al., 2025c)	Increase search efficiency and credit density by allocating compute to promising reasoning branches under fixed budgets.
On-/off-policy control	Mix ratio of fresh rollouts vs replayed data	Replay-buffer provenance/recency and off-policy correction constraints	RePO (Li et al., 2025c); ReMix (Liang et al., 2025)	Improve sample efficiency using replay while constraining off-policy drift and optimization instability.
Systems-level control	Throughput, tail latency, and scheduling policy	Length-distribution statistics, drafter quality, context similarity, and parallelism layout signals	ReSpec (Chen et al., 2025d); DAS (Shao et al., 2025b); TLT (Hu et al., 2025a); EARL (Tan et al., 2025); Seer (Qin et al., 2025)	Treat rollout generation as a systems bottleneck and optimize decoding/scheduling to raise effective training throughput.

6.3 Budgeting and scheduling (how much do you roll out?)

One recurring design choice across recent reinforcement learning approaches for reasoning is how rollout compute is distributed during training. Early GRPO-style methods (e.g., Guo et al. (2025a)) often used a fixed number of sampled trajectories per prompt to form group-relative advantages. Several works instead scale rollout compute directly by increasing either the number (breadth) or the repeated optimization depth on a prompt. For example, DeepSeekMath (Shao et al., 2024) uses GRPO-style group sampling (multiple candidates per question) with within-group reward normalization, directly scaling compute with group size. Similarly, 1-shot RLVR (Wang et al., 2025e) repeatedly samples rollouts and updates on a single training example over many RL steps, suggesting that large per-prompt rollout depth can partially substitute for dataset diversity.

More recent methods argue that rollout budgets should be allocated adaptively rather than uniformly across prompts. Variance-aware scheduling approaches such as VIP (Nguyen et al., 2026) allocate rollout compute to minimize expected policy-gradient variance using predicted per-prompt success probabilities. MMR1 (Leng et al., 2025) similarly biases sampling toward prompts with higher outcome variance and trajectory diversity to mitigate gradient vanishing. XRPO (Bamba et al., 2025) introduces a rollout allocator that prioritizes

prompts with higher potential for uncertainty reduction, and further improves exploration/exploitation via in-context seeding on zero-reward prompts and novelty-aware advantage sharpening. A complementary line of work focuses on reducing rollout cost through adaptive rollout and reuse: AR3PO (Zhang et al., 2025l) allocates more responses to difficult prompts while saving computation on easier ones, and reuses previously generated correct responses to provide additional training signal.

6.4 Rollout configuration control (length, temperature, concision, positive or negative ratio)

A key control knob in reasoning RL is how rollouts are parameterized and selected under a compute budget. Recent work shows that longer Chain-of-Thought traces are not uniformly beneficial: while they may help with hard problems, they often induce redundant *overthinking* that inflates the token usage without improving correctness. Accordingly, configuration control increasingly treats reasoning length and thinking mode as decision variables, and shapes the training experience via sampling, filtering, and curricula to trade train-time compute for test-time efficiency.

Several methods directly optimize for shorter-but-correct trajectories. ShorterBetter (Yi et al., 2025) defines the Sample Optimal Length (SOL)—the shortest correct response among multiple generations—and uses it as a dynamic reward signal to learn an instance-adaptive optimal CoT length without manual supervision. Addressing the limitation of naive length penalties, DECS (Jiang et al., 2025e) identifies a mismatch between trajectory-level rewards and token-level optimization, and introduces decoupled token-level rewards together with curriculum batch scheduling to penalize redundant tokens while preserving essential exploration.

Another line of work controls *whether* to think. AdaptThink (Zhang et al., 2025e) observes that a direct-answer mode (NoThinking) can outperform long reasoning on simpler queries, and proposes an RL algorithm that learns to select between thinking modes based on problem difficulty while maintaining overall performance. Similarly, Large Hybrid-Reasoning Models employ a two-stage pipeline (cold-start fine-tuning followed by online RL) to learn hybrid thinking decisions and evaluate this capability with a dedicated Hybrid Accuracy metric (Jiang et al., 2025c). CoRL (Jin et al., 2025a) introduces a reinforcement learning framework that optimizes the performance–cost trade-off when reasoning with an external LLM.

Finally, configuration control can be realized through train-time sampling and selection rules that shape which trajectories drive updates. GFPO (Shrivastava et al., 2025) mitigates RLVR-induced length inflation by sampling larger groups per problem and filtering training responses using length and token-efficiency (reward per token), demonstrating a direct trade-off where increased training-time compute yields reduced test-time compute. Complementarily, Train Long, Think Short applies a length-control curriculum under GRPO, starting with generous token budgets and progressively tightening them, with rewards balancing correctness, length efficiency, and formatting adherence, consistently outperforming fixed-budget baselines at the same final budget (Hammoud et al., 2025).

6.5 Partial rollouts, early exit, and continuation rules

Control can reduce rollout cost by (i) learning when to stop reasoning within a trajectory, (ii) pausing/continuing partial generations to mitigate long-tail stragglers, or (iii) down-selecting which rollouts participate in the (communication-heavy) policy update. S-GRPO (Dai et al., 2025) targets early exit by sampling a single reasoning path and training the model to exit at earlier positions via serial-group decaying rewards, encouraging concise thoughts and earlier termination. APRIL (Zhou et al., 2025b) is a systems-oriented partial-rollout scheme that over-provisions rollout requests, terminates a batch once enough responses are completed, and recycles unfinished generations for continuation in future steps, improving GPU utilization under long-tail response lengths. PODS (Xu et al., 2025) reduces update cost by selecting a strategically chosen subset of rollouts (e.g., max-variance down-sampling that maximizes reward diversity) to train on, decoupling cheap rollout generation from expensive multi-device optimization.

6.6 Branching and pruning control (trees and agents)

Branching and pruning strategies control how compute is allocated across multiple reasoning paths in tree-structured rollouts, deciding which nodes to expand, which paths to prune, and how deep to explore under a

fixed budget. Beyond improving exploration, tree topology can also be exploited to provide denser supervision by exposing intermediate states for credit assignment. TreeRL (Hou et al., 2025) directly incorporates on-policy tree search into RL training, strategically branching from high-uncertainty intermediate steps to improve search efficiency and provide intermediate supervision without a separate reward model. TreePO (Li et al., 2025d) treats sequence generation as a tree search with segment-wise decoding; it uses local uncertainty to warrant additional branches, amortizes compute across shared prefixes, and prunes low-value paths early to reduce KV-cache and sampling cost. DeepSearch (Wu et al., 2025a) integrates Monte Carlo Tree Search (MCTS) directly into RLVR training, embedding structured search into the training loop to improve exploration and enable fine-grained credit assignment. Finally, TreeRPO (Yang et al., 2025c) uses tree sampling to estimate expected rewards at different reasoning steps and computes group-relative step-level rewards, producing dense training signals without training a separate step reward model.

6.7 On- and off-policy controls and correctness constraints

Recent work on reinforcement fine-tuning of large language models highlights the importance of balancing on-policy and off-policy rollouts to maintain both training stability and distributional correctness. On-policy rollouts align updates with the current policy but are compute intensive; replay and mix-policy approaches aim to reuse past rollouts to improve data efficiency. RePO (Li et al., 2025c) augments GRPO with a replay buffer and replay strategies to reuse off-policy rollouts, improving sample efficiency while retaining an on-policy component. ReMix (Liang et al., 2025) proposes Reincarnating Mix-policy Proximal Policy Gradient (ReMix), a general approach that enables on-policy RFT methods such as PPO and GRPO to leverage off-policy data for more compute-efficient training.

6.8 Systems to accelerate rollout throughput

Systems for accelerating rollout throughput treat long rollouts and batched generation as first-class control problems, explicitly optimizing token usage, scheduling, and infrastructure constraints in reinforcement learning pipelines. ReSpec (Chen et al., 2025d) adapts speculative decoding to RL by dynamically tuning decoding configurations, evolving the drafter to avoid staleness (e.g., via distillation), and reward-weighting updates to preserve training stability. DAS (Shao et al., 2025b) targets long-tail rollout lengths with a distribution-aware speculative-decoding framework that leverages historical rollout statistics to accelerate generation without changing model outputs. TLT (Hu et al., 2025a) similarly integrates adaptive speculative decoding into reasoning-RL pipelines to reduce long-tail latency and speed up training losslessly. EARL (Tan et al., 2025) optimizes agentic RL training under rapidly growing multi-turn contexts using dynamic parallelism selection and layout-aware decentralized data dispatch to improve throughput and reduce long-context failures. Finally, Seer (Qin et al., 2025) improves synchronous RL rollouts by exploiting prompt-level similarities in output lengths and generation patterns, combining divided rollouts for dynamic load balancing, context-aware scheduling, and adaptive grouped speculative decoding to reduce tail latency and raise throughput.

7 Replay: Retention, Reuse, and Self-Evolution

7.1 Problem Formulation: Replay (R) — Retention, Reuse, and Self-Evolution

Using the global notation in §3.2, the **Replay** module governs what persists across rollouts *without* updating model parameters. It maintains a persistent replay state $\mathcal{B}$ (buffer/cache) that stores reusable artifacts derived from past rollouts—cached responses, full trajectories, prefixes $\tau_{\leq t}$, verified segments, or self-generated tasks—so that future rollouts can reuse past computation. Because $\mathcal{B}$ evolves over time, Replay makes the end-to-end rollout behavior history-dependent: for prompt x, the GFCR pipeline induces

$$\mathcal{T} \sim q_{\theta,\mathrm{GFCR}}(\cdot \mid x, \mathcal{B}), \tag{15}$$

highlighting that the realized distribution over rollout groups depends not only on π_θ but also on what has been retained and retrieved as shown in Figure 7.

Replay can be viewed as inducing an *off-policy* data source. Let π_{θ^-} denote the behavior policy under which a stored artifact was generated (e.g., a prior checkpoint). Reusing data collected under π_{θ^-} improves sample efficiency but introduces two core risks under policy drift: (i) *off-policy bias* when learning signals are computed on mismatched distributions, and (ii) *evaluator drift* when verifiers/judges (or their calibration) change relative to the current policy outputs. We write each stored entry as

$$e = (\tau, \phi, S, c(\tau), \theta^-, t_{\text{store}}), \tag{16}$$

where ϕ and S are the Filter signals and derived training signal, $c(\tau)$ is cost, θ^- is the policy version (or identifier) that produced τ, and t_{store} is a timestamp or age.

Formally, after generating a rollout group $\mathcal{T}(x)$, Replay applies a retention rule that decides what to store:

$$\mathcal{B} \leftarrow R_{\text{store}}\big(\mathcal{B}, \mathcal{T}(x), \{\phi(\tau^{(i)}; \mathcal{T}(x))\}_{i=1}^{K}\big), \tag{17}$$

and a corresponding policy for eviction, prioritization, or refresh. We denote by $\rho(e \mid x')$ a retrieval score (or priority) used to select entries relevant to a new prompt x', which may depend on similarity, verified correctness, diversity, cost, and freshness:

$$\mathcal{Z}(x') \sim R_{\text{retrieve}}(\mathcal{B} \mid x') \quad \text{with} \quad R_{\text{retrieve}} \text{ favoring high } \rho(e \mid x'). \tag{18}$$

Retrieved artifacts $\mathcal{Z}(x')$ can condition **Generate** (e.g., cached candidates, exemplars, verified sub-traces), influence **Control** (e.g., warm-starting the frontier or allocating fewer samples when strong cached solutions exist), and stabilize **Filter** (e.g., by maintaining reward variance in group-based objectives).

This abstraction unifies three reuse granularities surveyed in this section. First, *response resampling and retention* treats full rollouts as reusable units, enabling replay buffers, policy mixing, and mechanisms that stabilize group-based objectives under policy collapse (e.g., injecting cached correct responses to prevent advantages from vanishing). Second, *recomposition* treats trajectories as compositional, storing and recombining verified segments or prefixes $\tau_{\leq t}$ to amortize shared computation and target efficiency goals such as reduced verbosity or fewer tool calls. Third, *self-evolution* goes beyond the reuse of existing data: rollouts generate new tasks, solutions, or agents that feed back into training, forming self-evolving curricula that effectively expand the task distribution.

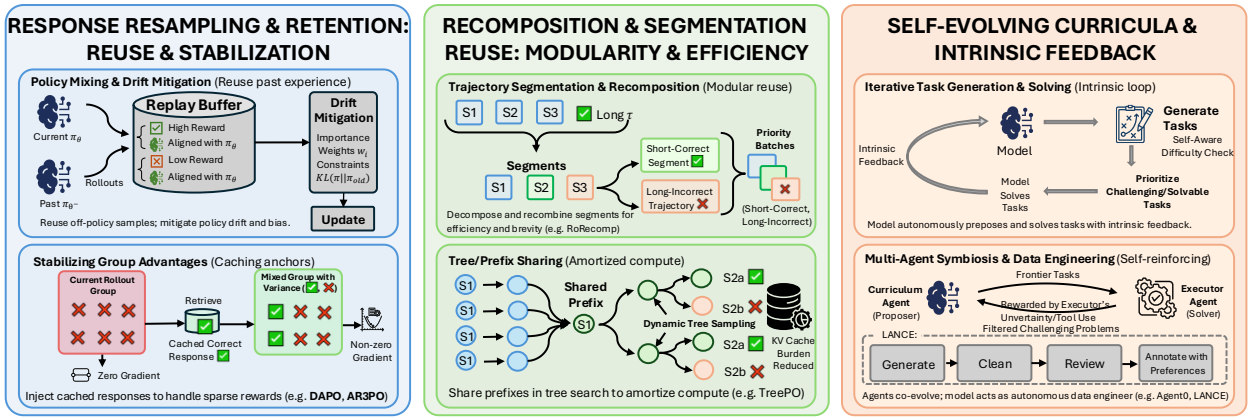

Figure 7: This figure summarizes the Replay layer of rollout pipelines, which governs how past rollouts are retained and reused without updating model parameters. Section 7 includes three components: (1) Response Resampling & Retention, which reuses full trajectories to stabilize learning; (2) Recomposition & Segmentation Reuse, which reuses segments or shared prefixes for modular efficiency; and (3) Self-Evolving Curricula & Intrinsic Feedback, which generates new tasks and data through intrinsic feedback. Together, these mechanisms improve data efficiency and training stability.

7.2 Response Resampling and Retention

Empirical study has demonstrated the effectiveness of replay strategies, as 1-shot RLVR (Wang et al., 2025e) shows that training on a single example with replay can match the performance of training on a 1.2k-example math subset (DeepScaleR) that contains that example. However, on-policy RL methods like GRPO do not reuse samples from previous policy iterations, limiting data efficiency. A direct response is to reuse cached trajectories and responses from earlier policies π_{θ^-}, but this introduces distribution shift under policy drift. RePO (Li et al., 2025c) maintains a replay buffer storing past rollouts and retrieves off-policy samples using diverse strategies, including prioritizing samples that are high-reward or closely aligned with the current policy. ReMix (Liang et al., 2025) proposes three components to mitigate instability under policy drift: mix-policy proximal policy gradient, a KL-convex policy constraint, and policy reincarnation. More generally, replay can be paired with principled off-policy objectives, e.g., Trajectory Balance with Asynchrony (Bartoldson et al., 2025) and Soft Policy Optimization (Cohen et al., 2025). Note that the importance ratio $\pi_\theta(u_t \mid s_t)/\pi_{\theta^-}(u_t \mid s_t)$ used by mix-policy updates can become extreme if policies diverge; AR3PO (Zhang et al., 2025l) addresses this by recomputing token probabilities under the current π_θ for reused responses.

Beyond replay buffers, recent work studies how to prioritize and reuse *valuable* experiences. ExGRPO (Zhan et al., 2025) organizes and prioritizes experiences using correctness and entropy as value indicators, then employs a mixed-policy objective to balance exploration with experience exploitation. Sample-centric progressive optimization (Chen et al., 2025f) also emphasizes per-sample prioritization via prefix-guided sampling and learning-progress weighting. For data efficiency, Sun et al. (2025c) combine difficulty-targeted online data selection with rollout replay, reducing RL fine-tuning time while matching GRPO performance.

Caching can also be used not only to reuse experience across iterations, but to stabilize learning signals *within* group-based objectives. In particular, group-normalized advantages in GRPO vanish when all rollouts for a prompt yield identical rewards, producing zero gradient. To address this, DAPO (Yu et al., 2025a) introduces a dynamic sampling strategy, which continues to sample new prompts and responses until the batch is fully filled with samples whose accuracy is neither all-zero nor all-one. This over-sampling inherently incurs higher inference cost, as observed in AR3PO (Zhang et al., 2025l) where they instead propose to retain correct responses from earlier training steps in the replay buffer. When current rollouts for a sample are all incorrect, a cached correct response will be injected from the buffer to retain the variance of rewards. Thus, incorrect rollouts receive negative advantages instead of zero without the need for response re-generation, offering efficiency enhancement. However, open questions remain regarding whether cached responses from π_{θ^-} remain valid anchors under the policy drift.

7.3 Recomposition and segment reuse

Rather than treating rollouts as atomic units, recomposition techniques decompose verified trajectories into segments or steps and recombine them into novel candidates, exploiting the modular structure of reasoning processes. RoRecomp (Li et al., 2025b) addresses RLVR's tendency toward verbosity by strategically recomposing training data into priority batches (pairing short-correct with long-incorrect responses to provide clear gradient signals for brevity) and compensation batches (replaying remaining responses from a buffer to prevent model collapse), achieving substantial reductions in reasoning length and unnecessary tool calls with minimal performance degradation. TreePO (Li et al., 2025d) introduces a self-guided rollout algorithm that views sequence generation as tree-structured search with dynamic tree sampling and fixed-length segment decoding, leveraging local uncertainty to spawn branches while amortizing computation across common prefixes; segment-wise sampling alleviates KV cache burden, while tree-based segment-level advantage estimation considers both global and local signals, reducing per-update compute while preserving exploration diversity. Related designs reuse strong prefixes or verified anchors as reusable starting points for branching and recomposition, e.g., BREAD (Zhang et al., 2025k) constructs branched rollouts from expert anchors to bridge supervised traces and RL updates. Both approaches reveal that reasoning trajectories exhibit compositional structure that can be exploited through recomposition, with the key insight being that granularity matters: response-level versus segment-level replay unlocks different efficiency gains depending on whether the goal is compressing verbose outputs or sharing redundant prefixes.

7.4 Self-evolving curricula and intrinsic feedback

Beyond caching and recomposing existing rollouts, self-evolution frameworks enable models to autonomously generate new training data through iterative loops where rollouts produce tasks, solutions, or agents that feed back into training. Efforts (Zhang et al., 2025c) have been made to propose a self-aware RL approach where the LLM alternates between proposing tasks and attempting to solve them. The framework introduces self-aware difficulty prediction to assess task difficulty relative to the model's current abilities and prioritize challenging yet solvable tasks, alongside self-aware limit breaking where the model recognizes capability boundaries and proactively requests minimal external data. Agent0 (Xia et al., 2025) establishes symbiotic competition between two agents initialized from the same base LLM: a curriculum agent that proposes increasingly challenging frontier tasks and an executor agent that learns to solve them. The curriculum agent is rewarded by the executor's uncertainty and tool-use frequency, while the executor is trained via RL on filtered challenging problems. External tools enhance the executor's problem-solving capacity, which in turn pressures the curriculum agent to construct more complex tool-aware tasks, establishing a self-reinforcing cycle without external human-curated data. LANCE (Wang et al., 2025b) enables LLMs to autonomously generate, clean, review, and annotate data with preference information. The model reviews seed data using constitutional principles to identify deficiencies, then generates new instruction-response pairs to address these gaps. Iterative cycles of data construction and preference-driven fine-tuning maintain high-quality generation across mathematical reasoning and general tasks. Complementary self-improvement loops also co-evolve reward modeling and policy learning (e.g., SPARK (Liu et al., 2025c)) or use self-play to continuously generate new training signals in interactive agentic domains (e.g., SWE-RL (Wei et al., 2025)). These approaches demonstrate that rollouts can actively shape the training distribution itself rather than serving as passive artifacts.

8 Domains & Case Studies

We view a benchmark as specifying the *rollout interface* on which a model policy is executed. A task instance is sampled as $x \sim \mathcal{D}$, and a rollout produces a trajectory $\tau = (x, u_{1:T}, o_{1:T})$ with implicit state $s_t = (x, u_{1:t-1}, o_{1:t-1})$. At each step, the model samples an output or action $u_t \sim \pi_\theta(\cdot \mid s_t)$, after which the interface returns an observation $o_t \sim P(\cdot \mid s_t, u_t)$, where $o_{1:T} = \emptyset$ in text-only settings. This induces a trajectory distribution

$$p_\theta(\tau \mid x) \;=\; \prod_{t=1}^{T} \pi_\theta(u_t \mid s_t)\, P(o_t \mid s_t, u_t), \tag{19}$$

where T is a stopping time (EOS, max length, success, or environment termination). Under this view, the benchmark determines the structure of rollouts and the form of feedback available for scoring and filtering.

We organize benchmarks into four categories based on the rollout interfaces they define:

1. **Verifiable language interfaces (math, code, SQL).** Text-in and text-out tasks with programmatic verification, such as normalized final-answer checks for math or execution-based checks for code and SQL.

2. **Multimodal reasoning interfaces.** Tasks with non-text inputs (vision, audio, video) where supervision relies on modality-aware verifiers, often implemented via structured answer extraction and rule-based checks.

3. **Agentic interactive interfaces.** Multi-step environments with tool or state feedback, where observations are non-empty and success is defined over trajectories and terminal environment states rather than a reference string.

4. **Agentic skill interfaces.** Environments that evaluate skill induction, library management, and cross-task transfer of reusable procedures, rather than independent episode solving alone.

8.1 Verifiable language interfaces (math, code, SQL)

Math benchmarks provide a canonical verifiable interface: rollouts are typically text-only so $o_{1:T} = \emptyset$ and $\tau = (x, u_{1:T})$ reduces to a completion $y = u_{1:T}$. A deterministic final-answer verifier $V(x, y)$ yields a binary correctness signal that can be used for evaluation and as a filtering primitive in rollout-based training. A representative dataset is MATH, which contains 12,500 competition mathematics problems with step-by-step solutions and supports exact match evaluation after answer normalization (Hendrycks et al., 2021). Standard evaluation suites for this interface include MATH500 and contest-style sets such as AIME/AMC, often alongside harder curated benchmarks such as OlympiadBench (He et al., 2024) (e.g., as reported in TreeRL (Hou et al., 2025)).

Math case study. In modern math reasoning post-training, this verifiable interface is often paired with RLVR-style objectives and exact-answer rewards, where rollout design (grouping/tree topology and sampling policy) materially affects both stability and cost. Representative systems scale verifiable training with math-specific data pipelines and RLVR-style objectives (e.g., DeepSeekMath (Shao et al., 2024); DeepSeek-R1 (Guo et al., 2025b); SEED-GRPO (Chen et al., 2025b)), and increasingly rely on tree/group rollouts (TreeRL (Hou et al., 2025); TreeRPO (Yang et al., 2025c)) and variance/uncertainty-aware curriculum or sampling (VCRL (Jiang et al., 2025b)) to improve reliability and efficiency. When process supervision is available, step-wise verifiers can further densify the interface beyond terminal answer checks (e.g., Math-Shepherd (Wang et al., 2023)), shifting the rollout design emphasis toward prefix/segment-level filtering and control.

Code and SQL benchmarks define a closely related but execution-grounded interface. The model outputs $u_{1:T}$ specify a program, an edit sequence, or a query, while verification is implemented via compilation, runtime execution, and unit tests, or via database execution for Text-to-SQL. LiveCodeBench (Jain et al., 2024) is a representative benchmark suite that evaluates multiple settings including direct generation and execution-aware variants. Text-to-SQL benchmarks instantiate the same execution-grounded interface with a database engine as the verifier, enabling correctness checks based on query execution outcomes (e.g., BIRD (Li et al., 2023)).

Code/SQL case study. Execution verifiers naturally induce multi-stage rollouts (generate $\rightarrow$ compile/execute $\rightarrow$ observe failures $\rightarrow$ retry/repair), making filtering and replay particularly concrete. CodeRL (Le et al., 2022) and RLTF (Liu et al., 2023) explicitly use compilation/tests as feedback signals to shape rollouts and learning. For Text-to-SQL, recent work demonstrates that execution outcomes can be used as scalar rewards under GRPO-style updates (Kulkarni & Srikumar, 2025), and RLVR-style systems can be built with simple rewards and strong reasoning (Yao et al., 2026; Ali et al., 2025). Recomposition and segment reuse are also natural in this domain, since verified patches, tests, or partial traces can be cached and recombined (Li et al., 2025b).

8.2 Multimodal reasoning interfaces

Multimodal benchmarks extend task instances beyond text, for example $x = (v, q) \sim \mathcal{D}_{\text{mm}}$ where v is an image or video and q is a textual query. A rollout produces a trajectory $\tau = (x, u_{1:T}, o_{1:T})$ and the primary object scored by the benchmark is the generated completion $y = u_{1:T}$. The defining component of the interface is the verifier, which extracts a structured answer from y and compares it to annotations or rule-based checks. Recent benchmarks and data pipelines emphasize making this verification pathway deterministic and scalable, including spatially grounded video reasoning with verifiable evaluation (Ouyang et al., 2025), verifiable synthesis pipelines for constructing checkable multimodal supervision (Wu et al., 2025d), and multimodal post-training resources that formalize reward construction and evaluation under verifiable or semi-verifiable signals (Zhang et al., 2025f; Leng et al., 2025). Additional multimodal reasoning resources follow the same interface while varying input modalities and answer formats (Huang et al., 2025b; Shen et al., 2025).

8.3 Agentic interactive benchmarks

Agentic benchmarks define an interactive rollout interface where the model alternates between actions and environment feedback. A task instance $x \sim \mathcal{D}_{\text{ag}}$ specifies an initial context and an action space over tools or environment operations. A rollout yields $\tau = (x, u_{1:T}, o_{1:T})$ with actions $u_t \sim \pi_\theta(\cdot \mid s_t)$ and observations $o_t \sim P(\cdot \mid s_t, u_t)$. Progress and correctness are mediated by tool outputs, execution results, or simulator responses, and cost is often dominated by long horizons and expensive interactions.

Software engineering agents instantiate this interface through codebase editing and test execution. Agent-RLVR provides repository and issue environments where the agent proposes patches and validates them via unit tests, producing observations such as build errors and test failures that define filtering and rewards (Da et al., 2025). Related evaluations commonly use SWE-Bench (Jimenez et al., 2023) and agent-oriented wrappers such as SWE-agent (Yang et al., 2024) and SWE-Gym (Pan et al., 2024).

Web agents expose a tool-mediated loop over webpages and external content. BrowserGym (Chezelles et al., 2024) and AgentDojo (Debenedetti et al., 2024) define tasks with actions such as clicking, typing, and navigation, and observations that reflect page state and tool outputs. ARLAS (Wang et al., 2025g) complements these interfaces with robustness-focused evaluation and adversarial training, including vulnerability to indirect prompt injection.

Dialogue simulators provide a text-based interactive interface with structured feedback. RLVER (Wang et al., 2025c) uses a simulated user that updates an internal emotion score e_t after each turn, inducing a verifiable trajectory-level reward from the emotion trajectory (e.g., the terminal score e_T) under a stopping condition defined by goal completion or a turn limit, and evaluates within the SAGE simulator framework (Zhang et al., 2025a).

More broadly, agentic post-training increasingly couples tool-logged trajectories with multi-turn control policies and self-generated curricula, e.g., Agent0 (Xia et al., 2025) alternates between curriculum construction and execution with tool use, producing interactive rollouts whose value depends on both terminal success and intermediate tool feedback.

Across these interfaces, the benchmark determines where feedback enters the trajectory, how verification is computed, and how rollout budgets should be allocated across depth, branching, and replay.

8.4 Agentic skills benchmarks

Agentic skill benchmarks evaluate whether an agent can induce reusable procedures from trajectories, store them, and transfer them to new tasks. Unlike the general agentic benchmarks above, which reward *capability reuse* only through independent episode solving, these interfaces reward *capability reuse* through skill induction, library management, and cross-task generalization.

Web-skill suites centre on WebArena and Mind2Web (typically instantiated through BrowserGym (Chezelles et al., 2024)). Agent Workflow Memory (Wang et al., 2024b) abstracts sub-routines into parameterized natural-language workflows retrieved in future episodes, measuring cross-task and cross-domain transfer. Subsequent work replaces text workflows with executable programs: Agent Skill Induction (Wang et al., 2025h) represents skills as Python functions verified via re-execution, and SkillWeaver (Zheng et al., 2025a) has agents autonomously discover and refine skills into reusable APIs that transfer across models, while ReUseIt (Liu et al., 2025b) synthesizes reusable workflows with execution guards from both successful and failed attempts. In tool-calling environments, Agent World Model (Wang et al., 2026a) trains agents via GRPO on ∼1,000 synthetic MCP-server environments and evaluates out-of-distribution on BFCLv3, τ^2-bench, and MCP-Universe, and Trajectory2Task (Wang et al., 2026b) synthesizes verifiable trajectories for ambiguous, changing, and infeasible user intents through its Retail-3I benchmark. For long-horizon app workflows, SAGE (Wang et al., 2025a) integrates a programmatic skill library into GRPO-based RL with sequential rollouts on AppWorld, where the scenario-level metric directly measures within-chain skill transfer.

Memory-centric and embodied suites test whether skill representations generalize across datasets and models. MemSkill (Zhang et al., 2026a) treats memory operations as learnable, evolvable skills via PPO-trained selection, evaluating on LoCoMo, LongMemEval, ALFWorld, and HotpotQA. SkillRL (Xia et al., 2026)

Table 7: Benchmark families viewed as rollout interfaces, with representative case studies and verifiers.

Interface family	Case study	Interface and feedback	Representative benchmarks / resources
Verifiable Language			
Verifiable language	Math	Text-only rollout ($o_{1:T} = \emptyset$); deterministic final-answer verification $V(x, y)$ after normalization.	MATH (Hendrycks et al., 2021); OlympiadBench (He et al., 2024); TreeRL/TreeRPO (Hou et al., 2025; Yang et al., 2025c).
Verifiable language	Code	Program/patch/query outputs; verification via compilation, execution, and unit tests.	LiveCodeBench (Jain et al., 2024); CodeRL/RLTF (Le et al., 2022; Liu et al., 2023).
Verifiable language	SQL	Query execution with a database engine; correctness by execution result.	BIRD (Li et al., 2023); execution-rewarded Text-to-SQL RL (Kulkarni & Srikumar, 2025); Arctic-Text2SQL-R1 (Yao et al., 2026).
Multimodal Reasoning			
Multimodal reasoning	General VLM post-training	Modality-aware verification via structured answer extraction + label/rule checks.	Multimodal RLVR resources (R1-VL; MMR1) (Zhang et al., 2025f; Leng et al., 2025).
Multimodal reasoning	Space / spatial-video	Rule-based or grounded evaluation to keep supervision deterministic and scalable.	SpaceR (Ouyang et al., 2025);SPACEVISTA (Sun et al., 2025b); InternSpatial (Deng et al., 2025); SPAR (Zhang et al., 2025d); VSI-Bench (Yang et al., 2025a).
Agentic Interactive			
Agentic interactive	Software engineering	Tool loop with non-empty observations; success measured by terminal repo/test state.	Agent-RLVR (Da et al., 2025); SWE-Bench / SWE-agent / SWE-Gym (Jimenez et al., 2023; Yang et al., 2024; Pan et al., 2024).
Agentic interactive	Web agents	Click/type/navigate actions; observations from page state and tool outputs.	BrowserGym; AgentDojo; AR-LAS (Chezelles et al., 2024; Debenedetti et al., 2024; Wang et al., 2025g).
Agentic interactive	Dialogue simulators	Turn-based interaction; reward from simulator state and terminal goal conditions.	RLVER; SAGE (Wang et al., 2025c; Zhang et al., 2025a).
Agentic Skills			
Agentic skills	Web-skill suites	Skill induction from trajectories; reuse via memory retrieval or programmatic re-execution.	AWM (Wang et al., 2024b); ASI (Wang et al., 2025h); Skill-Weaver (Zheng et al., 2025a); ReUseIt (Liu et al., 2025b).
Agentic skills	Tool-calling suites	Synthetic environment generation; verifiable trajectories for non-idealized intents.	Agent World Model (Wang et al., 2026a); Trajectory2Task (Wang et al., 2026b).
Agentic skills	Long-horizon & memory-centric	Skill-library integration with RL; procedural memory with cross-task/cross-model transfer.	SAGE (Wang et al., 2025a); MemSkill (Zhang et al., 2026a); SkillRL (Xia et al., 2026); MACLA (Forouzandeh et al., 2025); Mem^{P} (Fang et al., 2026); EXIF (Yang et al., 2025b).

co-evolves a hierarchical skill bank with the policy during GRPO on ALFWorld, WebShop, and search-augmented QA tasks. Concurrent work explores complementary procedural-memory mechanisms across overlapping benchmarks: hierarchical memory with Bayesian reliability tracking (Forouzandeh et al., 2025), step-level and script-level procedural abstractions with cross-model transfer (Fang et al., 2026), dual-form reusable expertise with continuous scoring and pruning (Qiu et al., 2026), exploration-driven skill discovery with iterative feedback (Yang et al., 2025b), trajectory distillation into reusable strategic principles (Wu et al., 2025c), and modular memory units for multi-agent workflow automation (Han et al., 2025). Across these settings, success is measured not only by terminal task completion but also by efficiency gains from skill reuse, robustness under changing intents, and cross-task transfer of learned procedures.

9 Failure Modes and Open Problems

Rollout pipelines tend to fail in recurring, diagnosable ways, but their root causes can span multiple stages of the Generate–Filter–Control–Replay (GFCR) lifecycle. Table 8 provides a troubleshooting index: each row

names a symptom, suggests a *primary* module to inspect first, and links to the most relevant subsections for concrete diagnostics and mitigation levers. Rows are grouped and color-coded by module for fast scanning. The *Primary module* column is intentionally a starting point rather than a unique attribution—when a local fix does not resolve the symptom, follow the section pointers to audit neighboring modules and shared measurement assumptions (e.g., verifier calibration, budget accounting, and provenance).

Table 8: **Troubleshooting index for rollout pathologies.** Each row maps an observed failure mode to the GFCR module that is typically the best first place to intervene, and provides diagnosis/mitigation pointers.

Issue	Primary module	Diagnosis and mitigation pointers
Filter: scoring, verification, supervision		
Spurious signals and bias	**Filter**	**Diagnosis.** Scores correlate with superficial properties (e.g., length, style, formatting); models exploit judge/verifier artifacts. **Mitigation.** Strengthen structural validity and safety gates (§5.2); use calibrated comparative assessment with abstention where appropriate (§5.5); design supervision to reduce proxy optimization (§5.7).
Verifier brittleness and metric mismatch	**Filter**	**Diagnosis.** High sensitivity to normalization and parsing; false positives/negatives under small output perturbations. **Mitigation.** Prefer deterministic, execution-based verification when available (§5.3); add redundant checks or consistency constraints, and incorporate process-level signals when terminal verification is noisy (§5.4).
Reward hacking and over-optimization	**Filter**	**Diagnosis.** Policies exploit proxy rewards (including judge idiosyncrasies) rather than improving task success; gains may not transfer and can regress under distribution shift. **Mitigation.** Prefer verifiable checks when possible (§5.3); improve reward/judge robustness and calibration (§5.5); use supervision mappings that reduce proxy exploitation and length/style bias (§5.7).
Generate: exploration and diversity		
Coverage collapse and near-duplicate rollouts	**Generate**	**Diagnosis.** Candidate sets have low diversity, high duplication, and limited disagreement, reducing effective supervision. **Mitigation.** Increase exploration and diversity controls (§4.4); diversify guidance and scaffolding (§4.3); allocate budget explicitly to exploration when needed (§6.3).
Control: budgeting, stopping, interaction policies		
Compute efficiency and tail performance regressions	**Control**	**Diagnosis.** Early stopping and partial rollouts improve average efficiency but reduce accuracy on difficult instances. **Mitigation.** Report per-difficulty token accounting and tail metrics; use tail-aware budgeting and scheduling (§6.3); apply conservative continuation/stopping rules (§6.5); adjust rollout configuration, including length control (§6.4).
Vanishing advantages in group-relative updates	**Control**	**Diagnosis.** Within-group reward variance is low, causing relative advantages to collapse and updates to weaken. **Mitigation.** Prioritize prompts with higher variance or uncertainty (§6.2); adapt group size and sampling budget (§6.3); emphasize learning-value signals and group-aware weighting (§5.6, §5.7); consider replay-based variance-restoration strategies (§7.2).
Instability in interactive search and tool use	**Control**	**Diagnosis.** Over-pruning, repeated tool failures, or looping behavior; success depends strongly on branching and retry policies. **Mitigation.** Use principled branch/prune control (§6.6); choose interaction topology that matches feedback latency (§4.2); enforce executability constraints and tool safety gates (§5.2).
Replay: reuse, drift, self-evolution		
Replay drift and missing provenance	**Replay**	**Diagnosis.** Reused trajectories become stale as policies and environments change; recomposition can propagate unverified fragments. **Mitigation.** Apply retention policies that track recency and versioning (§7.2); constrain recomposition to verified segments and re-verify upon reuse (§7.3, §5.3); control on-policy/off-policy mixing when replay is used for training (§6.7).
Unsafe self-evolution and self-amplified errors	**Replay**	**Diagnosis.** Self-generated tasks or data induce distribution shift, contamination, or escalating invalid and unsafe outputs. **Mitigation.** Require strict acceptance gates and verifier-backed inclusion (§5.2, §5.3); limit the share of self-generated data and enforce periodic evaluation on fixed, external benchmarks (§7.4).

Open problems. Several recurrent gaps cut across modules:

- **Verifier/judge evaluation and calibration.** We lack standardized, domain-spanning protocols for measuring verifier error rates, robustness to formatting/normalization, and how verifier calibra-

tion drifts over time (Yan et al., 2025; Huang et al., 2025c). This impacts both filtering reliability (§5.3, §5.5) and control decisions that depend on early feedback (§6.5).

- **Reward misalignment and transfer.** Even with verifiable rewards, RLVR systems can learn brittle heuristics or exploit reward artifacts, raising questions about what forms of supervision best transfer across tasks and domains (Shao et al., 2025a; Gao et al., 2023). This is tightly coupled to training-signal construction (§5.7) and generation diversity (§4.4).
- **Compute accounting and measurement.** Many methods report tokens but omit comparable accounting of tool calls, verifier runtime, branch/prune overhead, and replay refresh, making it difficult to assess true cost/benefit trade-offs (Tu et al., 2025). This affects Control and Systems choices (§6.3, §6.8).
- **Safe reuse and self-evolution.** Replay and self-evolution raise unresolved questions about provenance tracking, contamination, and how to bound the influence of self-generated data while still enabling autonomous curricula (Jain et al., 2024). This spans Replay and Filter (§7.4, §5.2).

Reading guide. In practice, many failures arise from interactions across modules. We therefore recommend starting with the primary module indicated in Table 8, then checking the referenced subsections for specific controls, safeguards, and measurement guidance. The visual icons appearing in the figures were created using OpenAI's ChatGPT and are intended purely as illustrations.

10 Conclusion

In this survey, we argued that rollout design is the missing link between optimization objectives and the experience that drives post-training for reasoning-oriented LLMs. We formalized rollout pipelines with unified notation and introduced the Generate–Filter–Control–Replay (GFCR) framework to decompose them into four composable modules, and used it to synthesize recent methods across verifiable language interfaces (math, code, SQL), multimodal reasoning, interactive agentic settings, and agentic skill benchmarks that evaluate skill induction, reuse, and cross-task transfer. Together with our criterion taxonomy (Table 3), GFCR makes rollout choices explicit and comparable, and supports a diagnostic view of recurring failure modes (Table 8) and their mitigation levers. Looking ahead, key open problems include robust verifier/judge evaluation and calibration, principled compute accounting beyond token counts, and safe reuse and self-evolution with provenance tracking (§9). We hope this perspective helps practitioners build and debug rollout pipelines, and encourages transparent reporting of rollout details that materially affect outcomes (e.g., verifier/judge configurations, budget policies, branching/pruning overheads, and replay refresh rules).

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

Alnur Ali, Ashutosh Baheti, Jonathan Chang, Ta-Chung Chi, Brandon Cui, Andrew Drozdov, Jonathan Frankle, Abhay Gupta, Pallavi Koppol, Sean Kulinski, Jonathan Li, Dipendra Misra, Krista Opsahl-Ong, Jose Javier Gonzalez Ortiz, Matei Zaharia, and Yue Zhang. A state-of-the-art sql reasoning model using rlvr, 2025. URL https://arxiv.org/abs/2509.21459.

Udbhav Bamba, Minghao Fang, Yifan Yu, Haizhong Zheng, and Fan Lai. Xrpo: Pushing the limits of grpo with targeted exploration and exploitation. *arXiv preprint arXiv:2510.06672*, 2025.

Brian Bartoldson, Siddarth Venkatraman, James Diffenderfer, Moksh Jain, Tal Ben-Nun, Seanie Lee, Minsu Kim, Johan Obando-Ceron, Yoshua Bengio, and Bhavya Kailkhura. Trajectory balance with asynchrony: Decoupling exploration and learning for fast, scalable llm post-training, 2025. URL https://arxiv.org/abs/2503.18929.

Philipp Becker, Niklas Freymuth, Serge Thilges, Fabian Otto, and Gerhard Neumann. Troll: Trust regions improve reinforcement learning for large language models, 2025. URL https://arxiv.org/abs/2510.03817.

Shreyas Chaudhari, Pranjal Aggarwal, Vishvak Murahari, Tanmay Rajpurohit, Ashwin Kalyan, Karthik Narasimhan, Ameet Deshpande, and Bruno Castro da Silva. Rlhf deciphered: A critical analysis of reinforcement learning from human feedback for llms. *ACM Comput. Surv.*, 58(2), September 2025. ISSN 0360-0300. doi: 10.1145/3743127. URL https://doi.org/10.1145/3743127.

Ding Chen, Qingchen Yu, Pengyuan Wang, Wentao Zhang, Bo Tang, Feiyu Xiong, Xinchi Li, Minchuan Yang, and Zhiyu Li. xverify: Efficient answer verifier for reasoning model evaluations. *arXiv preprint arXiv:2504.10481*, 2025a. URL https://arxiv.org/abs/2504.10481.

Minghan Chen, Guikun Chen, Wenguan Wang, and Yi Yang. Seed-grpo: Semantic entropy enhanced grpo for uncertainty-aware policy optimization, 2025b. URL https://arxiv.org/abs/2505.12346.

Peter Chen, Xiaopeng Li, Ziniu Li, Xi Chen, and Tianyi Lin. Stepwise guided policy optimization: Coloring your incorrect reasoning in grpo. *arXiv preprint arXiv:2505.11595*, 2025c. URL https://arxiv.org/abs/2505.11595.

Qiaoling Chen, Zijun Liu, Peng Sun, Shenggui Li, Guoteng Wang, Ziming Liu, Yonggang Wen, Siyuan Feng, and Tianwei Zhang. Respec: Towards optimizing speculative decoding in reinforcement learning systems. *arXiv preprint arXiv:2510.26475*, 2025d.

Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. Self-evolving curriculum for llm reasoning, 2025e. URL https://arxiv.org/abs/2505.14970.

Xinjie Chen, Minpeng Liao, Guoxin Chen, Chengxi Li, Biao Fu, Kai Fan, and Xinggao Liu. From data-centric to sample-centric: Enhancing llm reasoning via progressive optimization. *arXiv preprint arXiv:2507.06573*, 2025f.

De Chezelles, Thibault Le Sellier, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F Xu, Siva Reddy, Quentin Cappart, et al. The browsergym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*, 2024.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Taco Cohen, David W Zhang, Kunhao Zheng, Yunhao Tang, Remi Munos, and Gabriel Synnaeve. Soft policy optimization: Online off-policy rl for sequence models. *arXiv preprint arXiv:2503.05453*, 2025.

Jeff Da, Clinton Wang, Xiang Deng, Yuntao Ma, Nikhil Barhate, and Sean Hendryx. Agent-rlvr: Training software engineering agents via guidance and environment rewards, 2025. URL https://arxiv.org/abs/2506.11425.

Muzhi Dai, Chenxu Yang, and Qingyi Si. S-grpo: Early exit via reinforcement learning in reasoning models. *arXiv preprint arXiv:2505.07686*, 2025.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *Advances in Neural Information Processing Systems*, 37:82895–82920, 2024.

Nianchen Deng, Lixin Gu, Shenglong Ye, Yinan He, Zhe Chen, Songze Li, Haomin Wang, Xingguang Wei, Tianshuo Yang, Min Dou, et al. Internspatial: A comprehensive dataset for spatial reasoning in vision-language models. *arXiv preprint arXiv:2506.18385*, 2025.

Zheng Ding and Weirui Ye. Treegrpo: Tree-advantage grpo for online rl post-training of diffusion models. *arXiv preprint arXiv:2512.08153*, 2025.

Zhihao Dou, Qinjian Zhao, Zhongwei Wan, Dinggen Zhang, Weida Wang, Towsif Raiyan, Benteng Chen, Qingtao Pan, Yang Ouyang, Zhiqiang Gao, Shufei Zhang, and Sumon Biswas. Plan then action:high-level planning guidance reinforcement learning for llm reasoning, 2025. URL https://arxiv.org/abs/2510.01833.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory, 2026. URL https://arxiv.org/abs/2508.06433.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

Saman Forouzandeh, Wei Peng, Parham Moradi, Xinghuo Yu, and Mahdi Jalili. Learning hierarchical procedural memory for llm agents through bayesian selection and contrastive refinement. *arXiv preprint arXiv:2512.18950*, 2025.

Xiaolong Fu, Lichen Ma, Zipeng Guo, Gaojing Zhou, Chongxiao Wang, ShiPing Dong, Shizhe Zhou, Ximan Liu, Jingling Fu, Tan Lit Sin, et al. Dynamic-treerpo: Breaking the independent trajectory bottleneck with structured sampling. *arXiv preprint arXiv:2509.23352*, 2025.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pp. 10835–10866. PMLR, 2023.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024. URL https://arxiv.org/abs/2411.15594.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang

Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, September 2025a. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025b.

Yongxin Guo, Wenbo Deng, Zhenglin Cheng, and Xiaoying Tang. G$\hat{2}$rpo-a: Guided group relative policy optimization with adaptive guidance. *arXiv preprint arXiv:2508.13023*, 2025c.

Zichuan Guo and Hao Wang. A survey of reinforcement learning in large language models: From data generation to test-time inference. *Available at SSRN 5128927*, 2025.

Hasan Abed Al Kader Hammoud, Kumail Alhamoud, Abed Hammoud, Elie Bou-Zeid, Marzyeh Ghassemi, and Bernard Ghanem. Train long, think short: Curriculum learning for efficient reasoning. *arXiv preprint arXiv:2508.08940*, 2025.

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

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems, 2024. URL https://arxiv.org/abs/2402.14008.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. TreeRL: LLM reinforcement learning with on-policy tree search. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12355–12369, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.604. URL https://aclanthology.org/2025.acl-long.604/.

Qinghao Hu, Shang Yang, Junxian Guo, Xiaozhe Yao, Yujun Lin, Yuxian Gu, Han Cai, Chuang Gan, Ana Klimovic, and Song Han. Taming the long-tail: Efficient reasoning rl training with adaptive drafter. *arXiv preprint arXiv:2511.16665*, 2025a.

Zengjie Hu, Jiantao Qiu, Tianyi Bai, Haojin Yang, Binhang Yuan, Qi Jing, Conghui He, and Wentao Zhang. Vade: Variance-aware dynamic sampling via online sample-level difficulty estimation for multimodal rl, 2025b. URL https://arxiv.org/abs/2511.18902.

Bingning Huang, Tu Nguyen, and Matthieu Zimmer. Tree-opo: Off-policy monte carlo tree-guided advantage optimization for multistep reasoning. *arXiv preprint arXiv:2509.09284*, 2025a.

Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Yao Hu, and Shaohui Lin. Vision-r1: Incentivizing reasoning capability in multimodal large language models, 2025b. URL https://arxiv.org/abs/2503.06749.

Yuzhen Huang, Weihao Zeng, Xingshan Zeng, Qi Zhu, and Junxian He. From accuracy to robustness: A study of rule- and model-based verifiers in mathematical reasoning, 2025c. URL https://arxiv.org/abs/2505.22203.

Hugging Face. Math-verify: A robust mathematical expression evaluation system. https://github.com/huggingface/Math-Verify, 2025. Accessed: 2026-02-03.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Vaibhav Jain and Gerrit Grossmann. Guiding exploration in reinforcement learning through llm-augmented observations, 2025. URL https://arxiv.org/abs/2510.08779.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for llm agent reinforcement learning. *arXiv preprint arXiv:2509.21240*, 2025.

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

Guochao Jiang, Wenfeng Feng, Guofeng Quan, Chuzhan Hao, Yuewei Zhang, Guohua Liu, and Hao Wang. Vcrl: Variance-based curriculum reinforcement learning for large language models. *arXiv preprint arXiv:2509.19803*, 2025b.

Lingjie Jiang, Xun Wu, Shaohan Huang, Qingxiu Dong, Zewen Chi, Li Dong, Xingxing Zhang, Tengchao Lv, Lei Cui, and Furu Wei. Think only when you need with large hybrid-reasoning models. *arXiv preprint arXiv:2505.14631*, 2025c.

Ruili Jiang, Kehai Chen, Xuefeng Bai, Zhixuan He, Juntao Li, Muyun Yang, Tiejun Zhao, Liqiang Nie, and Min Zhang. A survey on human preference learning for aligning large language models. *ACM Computing Surveys*, 58(6):1–39, 2025d.

Shuyang Jiang, Yusheng Liao, Ya Zhang, Yanfeng Wang, and Yu Wang. Overthinking reduction with decoupled rewards and curriculum data scheduling, 2025e. URL https://arxiv.org/abs/2509.25827.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Bowen Jin, TJ Collins, Donghan Yu, Mert Cemri, Shenao Zhang, Mengyu Li, Jay Tang, Tian Qin, Zhiyang Xu, Jiarui Lu, et al. Controlling performance and budget of a centralized multi-agent llm system with reinforcement learning. *arXiv preprint arXiv:2511.02755*, 2025a.

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

Aayush Karan and Yilun Du. Reasoning with sampling: Your base model is smarter than you think, 2025. URL https://arxiv.org/abs/2510.14901.

Timo Kaufmann, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. A Survey of Reinforcement Learning from Human Feedback. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856.

Polina Kirichenko, Mark Ibrahim, Kamalika Chaudhuri, and Samuel J. Bell. Abstentionbench: Reasoning llms fail on unanswerable questions, 2025. URL https://arxiv.org/abs/2506.09038.

Atharv Kulkarni and Vivek Srikumar. Reinforcing code generation: Improving text-to-sql with execution-based learning. *arXiv preprint arXiv:2506.06093*, 2025.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning, 2022. URL https://arxiv.org/abs/2207.01780.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Cărbune, Abhinav Rastogi, and Sushant Prakash. Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*, 2023. URL https://arxiv.org/abs/2309.00267.

Sicong Leng, Jing Wang, Jiaxi Li, Hao Zhang, Zhiqiang Hu, Boqiang Zhang, Yuming Jiang, Hang Zhang, Xin Li, Lidong Bing, Deli Zhao, Wei Lu, Yu Rong, Aixin Sun, and Shijian Lu. Mmr1: Enhancing multimodal reasoning with variance-aware sampling and open resources, 2025. URL https://arxiv.org/abs/2509.21268.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, et al. From generation to judgment: Opportunities and challenges of llm-as-a-judge. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 2757–2791, 2025a.

Gang Li, Yulei Qin, Xiaoyu Tan, Dingkang Yang, Yuchen Shi, Zihan Xu, Xiang Li, Xing Sun, and Ke Li. Rorecomp: Enhancing reasoning efficiency via rollout response recomposition in reinforcement learning. *arXiv preprint arXiv:2509.25958*, 2025b.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Siheng Li, Zhanhui Zhou, Wai Lam, Chao Yang, and Chaochao Lu. Repo: Replay-enhanced policy optimization, 2025c. URL https://arxiv.org/abs/2506.09340.

Yizhi Li, Qingshui Gu, Zhoufutu Wen, Ziniu Li, Tianshun Xing, Shuyue Guo, Tianyu Zheng, Xin Zhou, Xingwei Qu, Wangchunshu Zhou, et al. Treepo: Bridging the gap of policy optimization and efficacy and inference efficiency with heuristic tree-based modeling. *arXiv preprint arXiv:2508.17445*, 2025d.

Yuming Li, Yikai Wang, Yuying Zhu, Zhongyu Zhao, Ming Lu, Qi She, and Shanghang Zhang. Branchgrpo: Stable and efficient grpo with structured branching in diffusion models. *arXiv preprint arXiv:2509.06040*, 2025e.

Ziniu Li, Pengyuan Wang, Tian Xu, Tian Ding, Ruoyu Sun, and Yang Yu. Review of reinforcement learning for large language models: Formulations, algorithms, and opportunities, 2025f.

Jing Liang, Hongyao Tang, Yi Ma, Jinyi Liu, Yan Zheng, Shuyue Hu, Lei Bai, and Jianye Hao. Squeeze the soaked sponge: Efficient off-policy reinforcement finetuning for large language model. *arXiv preprint arXiv:2507.06892*, 2025.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Tianqianjin Lin, Xi Zhao, Xingyao Zhang, Rujiao Long, Yi Xu, Zhuoren Jiang, Wenbo Su, and Bo Zheng. Ravr: Reference-answer-guided variational reasoning for large language models. *arXiv preprint arXiv:2510.25206*, 2025. URL https://arxiv.org/abs/2510.25206.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, Qiang Fu, Xiao Han, Wei Yang, and Deheng Ye. Rltf: Reinforcement learning from unit test feedback, 2023. URL https://arxiv.org/abs/2307.04349.

Qiyuan Liu, Hao Xu, Xuhong Chen, Wei Chen, Yee Whye Teh, and Ning Miao. Enhancing large language model reasoning with reward models: An analytical survey. *arXiv preprint arXiv:2510.01925*, 2025a.

Yimeng Liu, Misha Sra, Jeevana Priya Inala, and Chenglong Wang. Reuseit: Synthesizing reusable ai agent workflows for web automation. *arXiv preprint arXiv:2510.14308*, 2025b.

Ziyu Liu, Yuhang Zang, Shengyuan Ding, Yuhang Cao, Xiaoyi Dong, Haodong Duan, Dahua Lin, and Jiaqi Wang. Spark: Synergistic policy and reward co-evolving framework. *arXiv preprint arXiv:2509.22624*, 2025c.

Runyu Ma, Jelle Luijkx, Zlatan Ajanovic, and Jens Kober. Explorllm: Guiding exploration in reinforcement learning with large language models, 2025. URL https://arxiv.org/abs/2403.09583.

Youssef Mroueh. Reinforcement learning with verifiable rewards: Grpo's effective loss, dynamics, and success amplification, 2025. URL https://arxiv.org/abs/2503.06639.

Hieu Trung Nguyen, Bao Nguyen, Wenao Ma, Yuzhi Zhao, Ruifeng She, and Viet Anh Nguyen. Adaptive rollout allocation for online reinforcement learning with verifiable rewards, 2026. URL https://arxiv.org/abs/2602.01601.

Kun Ouyang, Yuanxin Liu, Haoning Wu, Yi Liu, Hao Zhou, Jie Zhou, Fandong Meng, and Xu Sun. Spacer: Reinforcing mllms in video spatial reasoning. *arXiv preprint arXiv:2504.01805*, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

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

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

Ruoyu Qin, Weiran He, Weixiao Huang, Yangkun Zhang, Yikai Zhao, Bo Pang, Xinran Xu, Yingdi Shan, Yongwei Wu, and Mingxing Zhang. Seer: Online context learning for fast synchronous llm reinforcement learning. *arXiv preprint arXiv:2511.14617*, 2025.

Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. Autorefine: From trajectories to reusable expertise for continual llm agent refinement. *arXiv preprint arXiv:2601.22758*, 2026.

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

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

Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, et al. Spurious rewards: Rethinking training signals in rlvr. *arXiv preprint arXiv:2506.10947*, 2025a.

Zelei Shao, Vikranth Srivatsa, Sanjana Srivastava, Qingyang Wu, Alpay Ariyak, Xiaoxia Wu, Ameen Patel, Jue Wang, Percy Liang, Tri Dao, et al. Beat the long tail: Distribution-aware speculative decoding for rl training. *arXiv preprint arXiv:2511.13841*, 2025b.

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.

Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, Ruochen Xu, and Tiancheng Zhao. Vlm-r1: A stable and generalizable r1-style large vision-language model, 2025. URL https://arxiv.org/abs/2504.07615.

Lin Shi, Chiyu Ma, Wenhua Liang, Xingjian Diao, Weicheng Ma, and Soroush Vosoughi. Judging the judges: A systematic study of position bias in LLM-as-a-judge. In Kentaro Inui, Sakriani Sakti, Haofen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dhirendra Pratap Singh (eds.), *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 292–314, Mumbai, India, December 2025. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics. ISBN 979-8-89176-298-5. URL https://aclanthology.org/2025.ijcnlp-long.18/.

Vaishnavi Shrivastava, Ahmed Awadallah, Vidhisha Balachandran, Shivam Garg, Harkirat Behl, and Dimitris Papailiopoulos. Sample more to think less: Group filtered policy optimization for concise reasoning, 2025. URL https://arxiv.org/abs/2508.09726.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. PRMBench: A fine-grained and challenging benchmark for process-level reward models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25299–25346, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1230. URL https://aclanthology.org/2025.acl-long.1230/.

Saksham Sahai Srivastava and Vaneet Aggarwal. A technical survey of reinforcement learning techniques for large language models. *arXiv preprint arXiv:2507.04136*, 2025.

Yi Su, Dian Yu, Linfeng Song, Juntao Li, Haitao Mi, Zhaopeng Tu, Min Zhang, and Dong Yu. Crossing the reward bridge: Expanding rl with verifiable rewards across diverse domains, 2025. URL https://arxiv.org/abs/2503.23829.

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

Peiwen Sun, Shiqiang Lang, Dongming Wu, Yi Ding, Kaituo Feng, Huadai Liu, Zhen Ye, Rui Liu, Yun-Hui Liu, Jianan Wang, et al. Spacevista: All-scale visual spatial reasoning from mm to km. *arXiv preprint arXiv:2510.09606*, 2025b.

Yifan Sun, Jingyan Shen, Yibin Wang, Tianyu Chen, Zhendong Wang, Mingyuan Zhou, and Huan Zhang. Improving data efficiency for llm reinforcement fine-tuning through difficulty-targeted online data selection and rollout replay, 2025c. URL https://arxiv.org/abs/2506.05316.

Zheyue Tan, Mustapha Abdullahi, Tuo Shi, Huining Yuan, Zelai Xu, Chao Yu, Boxun Li, and Bo Zhao. Earl: Efficient agentic reinforcement learning systems for large language models. *arXiv preprint arXiv:2510.05943*, 2025.

Hieu Tran, Zonghai Yao, and Hong Yu. Exploiting tree structure for credit assignment in rl training of llms. *arXiv preprint arXiv:2509.18314*, 2025.

Tuhina Tripathi, Manya Wadhwa, Greg Durrett, and Scott Niekum. Pairwise or pointwise? evaluating feedback protocols for bias in llm-based evaluation. *arXiv preprint arXiv:2504.14716*, 2025. URL https://arxiv.org/abs/2504.14716.

Aaron Tu, Weihao Xuan, Heli Qi, Xu Huang, Qingcheng Zeng, Shayan Talaei, Yijia Xiao, Peng Xia, Xiangru Tang, Yuchen Zhuang, Bing Hu, Hanqun Cao, Wenqi Shi, Tianang Leng, Rui Yang, Yingjian Chen, Ziqi Wang, Irene Li, Nan Liu, Huaxiu Yao, Li Erran Li, Ge Liu, Amin Saberi, Naoto Yokoya, Jure Leskovec, Yejin Choi, and Fang Wu. Position: The hidden costs and measurement gaps of reinforcement learning with verifiable rewards, 2025. URL https://arxiv.org/abs/2509.21882.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

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

Peidong Wang, Ming Wang, Zhiming Ma, Xiaocui Yang, Shi Feng, Daling Wang, Yifei Zhang, and Kaisong Song. Language models as continuous self-evolving data engineers. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18108–18127, 2025b.

Peisong Wang, Ruotian Ma, Bang Zhang, Xingyu Chen, Zhiwei He, Kang Luo, Qingsong Lv, Qingxuan Jiang, Zheng Xie, Shanyi Wang, Yuan Li, Fanghua Ye, Jian Li, Yifan Yang, Zhaopeng Tu, and Xiaolong Li. Rlver: Reinforcement learning with verifiable emotion rewards for empathetic agents, 2025c. URL https://arxiv.org/abs/2507.03112.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2023.

Shuhe Wang, Shengyu Zhang, Jie Zhang, Runyi Hu, Xiaoya Li, Tianwei Zhang, Jiwei Li, Fei Wu, Guoyin Wang, and Eduard Hovy. Reinforcement learning enhanced llms: A survey. *arXiv preprint arXiv:2412.10400*, 2024a.

Victor Wang, Michael JQ Zhang, and Eunsol Choi. Improving LLM-as-a-judge inference with the judgment distribution. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 23173–23199, Suzhou, China, November 2025d. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1259. URL https://aclanthology.org/2025.findings-emnlp.1259/.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. Reinforcement learning for reasoning in large language models with one training example, 2025e. URL https://arxiv.org/abs/2504.20571.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning, 2026a. URL https://arxiv.org/abs/2602.10090.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning, 2025f. URL https://arxiv.org/abs/2504.20073.

Ziyi Wang, Yuxuan Lu, Yimeng Zhang, Jing Huang, Jiri Gesi, Xianfeng Tang, Chen Luo, Yisi Sang, Hanqing Lu, Manling Li, et al. Trajectory2task: Training robust tool-calling agents with synthesized yet verifiable data for complex user intents. *arXiv preprint arXiv:2601.20144*, 2026b.

Zizhao Wang, Dingcheng Li, Vaishakh Keshava, Phillip Wallis, Ananth Balashankar, Peter Stone, and Lukas Rutishauser. Adversarial reinforcement learning for large language model agent safety. *arXiv preprint arXiv:2510.05442*, 2025g.

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

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

Yuxiang Wei, Zhiqing Sun, Emily McMilin, Jonas Gehring, David Zhang, Gabriel Synnaeve, Daniel Fried, Lingming Zhang, and Sida Wang. Toward training superintelligent software agents through self-play swe-rl. *arXiv preprint arXiv:2512.18552*, 2025.

Fang Wu, Weihao Xuan, Heli Qi, Ximing Lu, Aaron Tu, Li Erran Li, and Yejin ChoiRetry. Deepsearch: Overcome the bottleneck of reinforcement learning with verifiable rewards via monte carlo tree search. *arXiv preprint arXiv:2509.25454*, 2025a.

Feijie Wu, Weiwu Zhu, Yuxiang Zhang, Soumya Chatterjee, Jiarong Zhu, Fan Mo, Rodin Luo, and Jing Gao. Portool: Tool-use llm training with rewarded tree. *arXiv preprint arXiv:2510.26020*, 2025b.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, et al. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025c.

Zijian Wu, Jinjie Ni, Xiangyan Liu, Zichen Liu, Hang Yan, and Michael Qizhe Shieh. Synthrl: Scaling visual reasoning with verifiable data synthesis. *arXiv preprint arXiv:2506.02096*, 2025d.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL `https://arxiv.org/abs/2602.08234`.

Shangyu Xing, Siyuan Wang, Chenyuan Yang, Xinyu Dai, and Xiang Ren. Lookahead tree-based rollouts for enhanced trajectory-level exploration in reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2510.24302*, 2025.

Yixuan Even Xu, Yash Savani, Fei Fang, and J. Zico Kolter. Not all rollouts are useful: Down-sampling rollouts in llm reinforcement learning, 2025. URL `https://arxiv.org/abs/2504.13818`.

Yuchen Yan, Jin Jiang, Zhenbang Ren, Yijun Li, Xudong Cai, Yang Liu, Xin Xu, Mengdi Zhang, Jian Shao, Yongliang Shen, et al. Verifybench: Benchmarking reference-based reward systems for large language models. *arXiv preprint arXiv:2505.15801*, 2025.

Jihan Yang, Shusheng Yang, Anjali W Gupta, Rilyn Han, Li Fei-Fei, and Saining Xie. Thinking in space: How multimodal large language models see, remember, and recall spaces. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 10632–10643, 2025a.

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

Yongjin Yang, Sinjae Kang, Juyong Lee, Dongjun Lee, Se-Young Yun, and Kimin Lee. Automated skill discovery for language agents through exploration and iterative feedback. *arXiv preprint arXiv:2506.04287*, 2025b.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Xiaodan Liang, Yiwei Wang, and Jing Tang. Treerpo: Tree relative policy optimization. *arXiv preprint arXiv:2506.05183*, 2025c.

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

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Gaurav Nuti, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-sql, 2026. URL `https://arxiv.org/abs/2505.20315`.

Jingyang Yi, Jiazheng Wang, and Sida Li. Shorterbetter: Guiding reasoning models to find optimal inference length for efficient reasoning. *arXiv preprint arXiv:2504.21370*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL `https://openreview.net/forum?id=2a36EMSSTp`.

Tianyu Yu, Bo Ji, Shouli Wang, Shu Yao, Zefan Wang, Ganqu Cui, Lifan Yuan, Ning Ding, Yuan Yao, Zhiyuan Liu, et al. Rlpr: Extrapolating rlvr to general domains without verifiers. *arXiv preprint arXiv:2506.18254*, 2025b.

Yue Yu, Zhengxing Chen, Aston Zhang, Liang Tan, Chenguang Zhu, Richard Yuanzhe Pang, Yundi Qian, Xuewei Wang, Suchin Gururangan, Chao Zhang, Melanie Kambadur, Dhruv Mahajan, and Rui Hou. Self-generated critiques boost reward modeling for language models. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11499–11514, Albuquerque, New Mexico, April 2025c. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.573. URL `https://aclanthology.org/2025.naacl-long.573/`.

Yuyuan Zeng, Yufei Huang, Can Xu, Qingfeng Sun, Jianfeng Yan, Guanghui Xu, Tao Yang, and Fengzong Lian. Zero reinforcement learning towards general domains. *arXiv preprint arXiv:2510.25528*, 2025.

Runzhe Zhan, Yafu Li, Zhi Wang, Xiaoye Qu, Dongrui Liu, Jing Shao, Derek F. Wong, and Yu Cheng. Exgrpo: Learning to reason from experience, 2025. URL `https://arxiv.org/abs/2510.02245`.

Bang Zhang, Ruotian Ma, Qingxuan Jiang, Peisong Wang, Jiaqi Chen, Zheng Xie, Xingyu Chen, Yue Wang, Fanghua Ye, Jian Li, Yifan Yang, Zhaopeng Tu, and Xiaolong Li. Sentient agent as a judge: Evaluating higher-order social cognition in large language models, 2025a. URL `https://arxiv.org/abs/2505.02847`.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, et al. The landscape of agentic reinforcement learning for llms: A survey. *arXiv preprint arXiv:2509.02547*, 2025b.

Hangfan Zhang, Siyuan Xu, Zhimeng Guo, Huaisheng Zhu, Shicheng Liu, Xinrun Wang, Qiaosheng Zhang, Yang Chen, Peng Ye, Lei Bai, et al. The path of self-evolving large language models: Achieving data-efficient learning via intrinsic feedback. *arXiv preprint arXiv:2510.02752*, 2025c.

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

Jiahui Zhang, Yurui Chen, Yanpeng Zhou, Yueming Xu, Ze Huang, Jilin Mei, Junhui Chen, Yu-Jie Yuan, Xinyue Cai, Guowei Huang, et al. From flatland to space: Teaching vision-language models to perceive and reason in 3d. *arXiv preprint arXiv:2503.22976*, 2025d.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. AdaptThink: Reasoning models can learn when to think. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3716–3730, Suzhou, China, November 2025e. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.184. URL https://aclanthology.org/2025.emnlp-main.184/.

Jingyi Zhang, Jiaxing Huang, Huanjin Yao, Shunyu Liu, Xikun Zhang, Shijian Lu, and Dacheng Tao. R1-vl: Learning to reason with multimodal large language models via step-wise group relative policy optimization, 2025f. URL https://arxiv.org/abs/2503.12937.

Jipeng Zhang, Haolin Yang, Kehao Miao, Ruiyuan Zhang, Renjie Pi, Jiahui Gao, and Xiaofang Zhou. ExeSQL: Self-taught text-to-SQL models with execution-driven bootstrapping for SQL dialects. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 24305–24326, Suzhou, China, November 2025g. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1320. URL https://aclanthology.org/2025.findings-emnlp.1320/.

Jixiao Zhang and Chunsheng Zuo. Grpo-lead: A difficulty-aware reinforcement learning approach for concise mathematical reasoning in language models. *arXiv preprint arXiv:2504.09696*, 2025.

Junjie Zhang, Jingyi Xi, Zhuoyang Song, Junyu Lu, Yuhua Ke, Ting Sun, Yukun Yang, Jiaxing Zhang, Songxin Zhang, and Zejian Xie. L0: Reinforcement learning to become general agents, 2025h. URL https://arxiv.org/abs/2506.23667.

Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, et al. A survey of reinforcement learning for large reasoning models. *arXiv preprint arXiv:2509.08827*, 2025i.

Xichen Zhang, Sitong Wu, Yinghao Zhu, Haoru Tan, Shaozuo Yu, Ziyi He, and Jiaya Jia. Scaf-grpo: Scaffolded group relative policy optimization for enhancing llm reasoning, 2025j. URL https://arxiv.org/abs/2510.19807.

Xuechen Zhang, Zijian Huang, Yingcong Li, Chenshun Ni, Jiasi Chen, and Samet Oymak. Bread: Branched rollouts from expert anchors bridge sft & rl for reasoning, 2025k. URL https://arxiv.org/abs/2506.17211.

Yuheng Zhang, Wenlin Yao, Changlong Yu, Yao Liu, Qingyu Yin, Bing Yin, Hyokun Yun, and Lihong Li. Improving sampling efficiency in rlvr through adaptive rollout and response reuse, 2025l. URL https://arxiv.org/abs/2509.25808.

Zhaoxi Zhang, Yitong Duan, Yanzhi Zhang, Yiming Xu, Zhixiang Wang, Kun Liang, Yang Li, Jiahui Liang, Deguo Xia, Jizhou Huang, Jiyan He, and Yunfang Wu. One tool is enough: Reinforcement learning for repository-level llm agents, 2026b. URL https://arxiv.org/abs/2512.20957.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516, Vienna, Austria, July 2025m. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.547. URL https://aclanthology.org/2025.findings-acl.547/.

Jian Zhao, Runze Liu, Kaiyan Zhang, Zhimu Zhou, Junqi Gao, Dong Li, Jiafei Lyu, Zhouyi Qian, Biqing Qi, Xiu Li, and Bowen Zhou. Genprm: Scaling test-time compute of process reward models via generative reasoning. *arXiv preprint arXiv:2504.00891*, 2025a. URL https://arxiv.org/abs/2504.00891.

Weikang Zhao, Xili Wang, Chengdi Ma, Lingbin Kong, Zhaohua Yang, Mingxiang Tuo, Xiaowei Shi, Yitao Zhai, and Xunliang Cai. Mua-rl: Multi-turn user-interacting agent reinforcement learning for agentic tool use. *arXiv preprint arXiv:2508.18669*, 2025b.

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

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning, 2025b. URL https://arxiv.org/abs/2412.06559.

Haizhong Zheng, Yang Zhou, Brian R. Bartoldson, Bhavya Kailkhura, Fan Lai, Jiawei Zhao, and Beidi Chen. Act only when it pays: Efficient reinforcement learning for llm reasoning via selective rollouts, 2025c. URL https://arxiv.org/abs/2506.02177.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

Xin Zheng, Jie Lou, Boxi Cao, Xueru Wen, Yuqiu Ji, Hongyu Lin, Yaojie Lu, Xianpei Han, Debing Zhang, and Le Sun. Critic-CoT: Boosting the reasoning abilities of large language model via chain-of-thought critic. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 1768–1806, Vienna, Austria, July 2025d. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.89. URL https://aclanthology.org/2025.findings-acl.89/.

Xiangxin Zhou, Zichen Liu, Anya Sims, Haonan Wang, Tianyu Pang, Chongxuan Li, Liang Wang, Min Lin, and Chao Du. Reinforcing general reasoning without verifiers. *arXiv preprint arXiv:2505.21493*, 2025a.

Yuzhen Zhou, Jiajun Li, Yusheng Su, Gowtham Ramesh, Zilin Zhu, Xiang Long, Chenyang Zhao, Jin Pan, Xiaodong Yu, Ze Wang, Kangrui Du, Jialian Wu, Ximeng Sun, Jiang Liu, Qiaolin Yu, Hao Chen, Zicheng Liu, and Emad Barsoum. April: Active partial rollouts in reinforcement learning to tame long-tail generation, 2025b. URL https://arxiv.org/abs/2509.18521.

Jiaru Zou, Ling Yang, Jingwen Gu, Jiahao Qiu, Ke Shen, Jingrui He, and Mengdi Wang. Reasonflux-prm: Trajectory-aware prms for long chain-of-thought reasoning in llms, 2025. URL https://arxiv.org/abs/2506.18896.