Learning to Ask May Be Better Than Learning to Answer

Asmi Kumar
MIT
MIT-IBM Watson AI Lab
asmi@mit.edu

Chuang Gan
UMass Amherst
MIT-IBM Watson AI Lab
chuangg@cs.umass.edu

Zhang-Wei Hong
MIT
MIT-IBM Watson AI Lab
zwhong@mit.edu

Abstract

Prompt optimization is increasingly important for compound LLM systems, where instructions shape not only final answers but also multi-step tool use, retrieval decisions, and stopping behavior. We study prompt-side reinforcement learning: instead of training the downstream solver, we use GRPO to train a policy that generates task-conditioned prompts for a frozen ReAct agent. Across HoVer, HotPotQA, and GSM8K, prompt-side GRPO is most effective on retrieval-intensive tasks, improving top-5 supporting-title recall on HoVer by 7.33 points and answer F1 on HotPotQA by 10.98 points, while providing little benefit for direct mathematical reasoning. It scales more favorably with rollout budget than reflective search-based optimization and can generate structurally distinct retrieval strategies for different question types without explicit supervision. These results suggest that learning to ask is a viable and sometimes preferable alternative to learning to answer in retrieval-heavy compound LLM systems, and that rollout budget allocation should be treated as an explicit design choice.

1 Introduction

Large language models (LLMs) have changed how intelligent systems are built [2]. Many systems now obtain useful behavior through natural-language instructions, making the prompt a central interface between a model and the task it must solve [30, 45]. Prompts can define the task, constrain outputs, shape intermediate reasoning, and guide the use of external information [35, 34]. Their role is especially important in *compound LLM systems*, which are composed of one or more LLM calls, potentially interleaved with tools and organized by control flows [1]. In these systems, prompt quality affects the behavior of an entire system rather than a single response [39, 15, 12, 25].

This effect is particularly pronounced for tool-using agents [27, 31]. A prompt can influence how an agent decomposes a problem, retrieves information, calls tools, uses prior observations, and decides when to stop. A weak prompt can lead to broad searches, repeated actions, premature termination, or poor multi-hop reasoning, while a strong prompt can improve the same frozen solver without changing its weights, tools, or inference infrastructure.

We study prompt optimization in this agentic setting. Rather than optimizing the downstream solver, we train a prompt-generation policy that produces task-conditioned instructions for a frozen tool-using agent. Given a task instance, the policy writes a full natural-language prompt under which the agent acts, calls tools, accumulates a trajectory, and returns an answer or retrieved evidence. The policy is updated from reward computed on the resulting trajectory. Thus, the learned model does not solve the task directly; it learns how to ask another model to solve it.

This makes reinforcement learning (RL) a natural fit [5, 43]. The effect of a prompt is only observed after the downstream agent has completed a trajectory, so supervision is delayed and behavioral rather than token-level [32]. We can sample a prompt, run the frozen agent under it, score the resulting

43rd International Conference on Machine Learning (ICML 2026). RLxF: Reinforcement Learning from World Feedback.

behavior, and update the prompt-generation policy toward prompts that induce higher reward. We therefore treat prompting as an outer-loop control problem.

Let $x \sim \mathcal{D}$ denote a task instance, let $\pi_\theta(p \mid x)$ be a prompt-generation policy that produces a prompt p, let A denote a frozen downstream agent, and let $\tau \sim A(\cdot \mid x, p)$ be the resulting trajectory. With reward $R(\tau)$ defined by downstream task performance, the objective is

$$J(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\, \mathbb{E}_{p\sim\pi_\theta(\cdot|x)}\, \mathbb{E}_{\tau\sim A(\cdot|x,p)} \left[R(\tau)\right]. \tag{1}$$

This objective separates the learned component from the evaluated behavior: the policy generates prompts, while reward is computed from the behavior those prompts induce in a fixed downstream system.

We refer to this setting as learning on the **asking** side of a compound system. This contrasts with learning on the **answering** side, where the model that directly solves the task is updated. The distinction is useful when the downstream solver is fixed because of access restrictions, deployment stability, or inference infrastructure. In such cases, prompt-level adaptation provides a practical route for improving system behavior while leaving the deployed agent unchanged.

Our experiments address whether prompt-side RL can improve a frozen solver, how asking-side optimization compares to answering-side optimization under matched rollout budgets, and how sensitive each approach is to the available rollout budget. We study these questions across retrieval-heavy and direct-reasoning tasks.

1.1 Contributions

This paper makes four main contributions.

1. We formulate prompt optimization for compound LLM systems as an outer-loop RL problem, where a trainable policy generates task-conditioned prompts for a fixed downstream agent and receives reward from the resulting trajectory.
2. We define an agentic prompt-optimization setting in which prompts shape tool use, retrieval strategy, evidence tracking, redundancy avoidance, and stopping behavior.
3. We apply GRPO to the asking side of the system, training a prompt-generation policy while keeping the downstream solver fixed.
4. We compare asking-side optimization with answering-side optimization and prior prompt-optimization methods under matched rollout budgets, reduced rollout budgets, and combined-optimization settings.

2 Related Work

Prompt optimization. Prompt optimization methods differ in what they optimize and what feedback they use. Early approaches learn continuous prompts for frozen models, including prefix-tuning, prompt tuning, and P-Tuning v2 [16, 14, 18, 19, 17]. These methods recover much of the benefit of full fine-tuning with a small learned parameterization, but the optimized prompt is a continuous embedding rather than an interpretable instruction.

A second line of work keeps optimization in language space. AutoPrompt uses gradient-guided search over discrete trigger tokens [30]. APE treats instruction generation as black-box search over natural-language candidates [45, 37]. ProTeGi uses minibatch errors to form textual critiques for prompt edits [26], and PromptBreeder evolves prompts through self-referential mutations [7, 9, 33]. These methods optimize language prompts, usually by searching for a fixed prompt or a candidate population. By contrast, the optimized object here is a task-conditioned policy that generates a prompt for each input instance.

Prompt optimization for compound systems. As language models have become components of larger pipelines, prompt optimization has expanded from single prompts to language-model programs. DSPy formulates LM systems as modular programs optimized against downstream metrics [12, 6], and MIPRO optimizes instructions and demonstrations for multi-stage programs under end-to-end evaluation [23]. This paper follows the compound-system view, with a separate prompt-generation model whose output instructs a frozen downstream agent.

Language feedback, reflection, and evolution. Several methods use language feedback to improve prompts or compound systems. TextGrad propagates textual critiques through variables in a computation graph [40, 21, 31]. GEPA optimizes prompts for compound AI systems through reflective mutation and Pareto-based candidate selection while keeping model weights fixed [1]. GEPA is closely related because it studies prompt optimization for compound systems and compares prompt evolution with GRPO-based answer-model adaptation.

The key difference is where learning is applied. Here, a trainable model generates instructions for a frozen agent, and reward is assigned from the downstream trajectories produced under those instructions. This placement matters for open-ended tool use, where prompts must guide behavior across a sequence of decisions. Many prompt-optimization settings assume fixed pipelines with prescribed module boundaries, such as a query writer, summarizer, or answer generator. In the agentic setting studied here, the downstream agent chooses tools over a trajectory and decides when to stop, making the prompt responsible for search strategy, evidence tracking, redundancy avoidance, and termination [39, 27, 3, 36, 42].

Reinforcement learning for prompt optimization. Reinforcement learning is useful when prompt quality is observed only after a downstream interaction. Prior RL methods for prompt optimization often use constrained action spaces: RLPrompt learns discrete prompt tokens for frozen models [5], while TEMPERA performs test-time prompt editing through structured actions over instructions, exemplars, and verbalizers [43]. Recent GRPO-based work applies RL to the answer-generating model itself [28, 8, 24, 22, 41, 1].In this paper, GRPO is applied at the prompting interface.

3 Experimental Framework

We instantiate prompt optimization in benchmark environments with a shared episode interface. A task instance $x \sim \mathcal{D}$ is given to a fixed downstream solver together with a prompt p. The solver interacts with the environment for at most T steps, producing a trajectory τ, and the completed trajectory is scored by a scalar reward $R(\tau)$. The optimization methods used to improve prompts are described in Section 4.

We evaluate on three benchmarks that test complementary forms of structured language-model behavior. HoVer and HotPotQA require multi-hop retrieval and tool use, while GSM8K tests direct mathematical reasoning in a single model call. Each benchmark consists of a dataset of instances $\mathcal{D} = \{x_i\}_{i=1}^{N}$, a fixed solver interface, a bounded interaction budget, and a task-specific reward computed only after the episode terminates.

3.1 Tasks

The two retrieval benchmarks use the same ReAct-style episode structure (Algorithm 1). At each step, the solver emits a structured action $a_t = (\texttt{thought}_t, \texttt{tool}_t, \texttt{args}_t)$ from the tool set

$$\{\texttt{search_wikipedia(query)},\ \texttt{lookup_wikipedia(title)},\ \texttt{finish()}\}.$$

The environment executes the selected tool, appends the resulting observation to the trajectory, and returns the updated trajectory to the solver. The episode terminates when the solver calls `finish`, repeats a previously executed non-finish tool call, exhausts the step budget T, or produces invalid JSON or an unknown tool. After termination, a separate extraction prompt asks the solver to return its final prediction as strict JSON. This extraction step is used only to parse the completed trajectory into the benchmark-specific output format.

HoVer. HoVer [11] is a retrieval-augmented verification benchmark over Wikipedia. Each instance x consists of a natural-language claim and a hidden gold set $G(x)$ of supporting Wikipedia article titles. We evaluate the agent's ability to recover the titles needed to verify or refute the claim, rather than the final verification label. After the episode, the solver extracts predicted titles as strict JSON. Duplicates and empty strings are removed, and only the first five predicted titles are scored. The reward and reported metric are top-5 supporting-title recall:

$$r_{\text{base}}^{\text{HoVer}}(x, y) = \text{Acc}_{\text{HoVer}}(x, y) = \frac{1}{|G(x)|} \sum_{g \in G(x)} \mathbf{1}\{g \in \text{Top5}(y)\}.$$

Algorithm 1 ReAct episode structure

Require: Input x, prompt p, solver S, environment E, max steps T

$\tau \leftarrow \emptyset$; $\mathcal{C} \leftarrow \emptyset$
for $t = 1, \ldots, T$ **do**
  Sample $a_t = (\textit{thought}, \textit{tool}, \textit{args})$ from S under (p, x, τ)
  if invalid JSON **or** unknown tool **then**
    force `finish`; **break**
  end if
  if $(\textit{tool}, \textit{args}) \in \mathcal{C}$ **and** $\textit{tool} \neq$ `finish` **then**
    append loop warning; **break**
  end if
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{(\textit{tool}, \textit{args})\}$
  $o_t \leftarrow E.\textsc{Execute}(\textit{tool}, \textit{args})$
  append (a_t, o_t) to τ
  if $\textit{tool} =$ `finish` **then**
    break
  end if
end for
Extract final prediction y from τ using a JSON extraction prompt
return τ, y, and $r_{\text{base}}(x, y)$

HotPotQA. HotPotQA [38] is a multi-hop question-answering benchmark with bridge and comparison questions. Bridge questions require following entity chains across articles, while comparison questions require reasoning across related entities. Each instance contains a question, a gold answer a^*, and a gold set of supporting titles G. After the episode, the solver must return strict JSON with

$$\{\texttt{"answer"} : \hat{a},\ \texttt{"titles"} : [\hat{g}_1, \ldots]\}.$$

We report normalized answer F1, $F_1^{\text{ans}}(\hat{a}, a^*)$, and top-5 supporting-title recall, $R_{@5}^{\text{title}}(\hat{G}, G)$, separately. During training, we use their average as the base reward:

$$r_{\text{base}}^{\text{HotPotQA}}(x, y) = \tfrac{1}{2}\, F_1^{\text{ans}}(\hat{a},\ a^*) + \tfrac{1}{2}\, R_{@5}^{\text{title}}(\hat{G},\ G).$$

All HotPotQA experiments use hard-level questions.

GSM8K. GSM8K [4] is a grade-school arithmetic benchmark. Unlike the retrieval tasks, GSM8K is evaluated with a single frozen-model call and does not use the ReAct tool loop. The environment sends [`system prompt`, problem] to the solver and records the completion as the episode trajectory. The prompt asks the solver to provide step-by-step reasoning and a final numeric answer. The reward and reported metric are exact-match accuracy after answer normalization:

$$r_{\text{base}}^{\text{GSM8K}}(x, y) = \text{Acc}_{\text{GSM8K}}(x, y) = \mathbf{1}\{\text{norm}(y) = \text{norm}(y^*)\}.$$

3.2 Prompt-Policy Parameterization

Let p_{seed} be a benchmark-specific seed prompt and $c(x)$ the control context for instance x (task description, required tool names, and lightweight instance metadata). We use **Qwen-3-8B** with LoRA adapters as the prompt-generation policy:

$$\pi_\theta^{\text{prompt}}(p \mid p_{\text{seed}},\ c(x)).$$

Conditioning on p_{seed} keeps exploration within prompts that define a valid solver interface, allowing optimization to focus on controller behavior such as search-vs-lookup decisions, redundancy avoidance, evidence tracking, and stopping criteria.

3.3 Optimization Profiles

To encourage exploration, each prompt-generation request includes a short optimization profile. A profile is a natural-language directive that biases the generated prompt toward a particular axis of controller behavior (e.g., aggressive early search, deferred `finish` until a coverage threshold is met, or prioritizing `lookup` over repeated `search` calls).

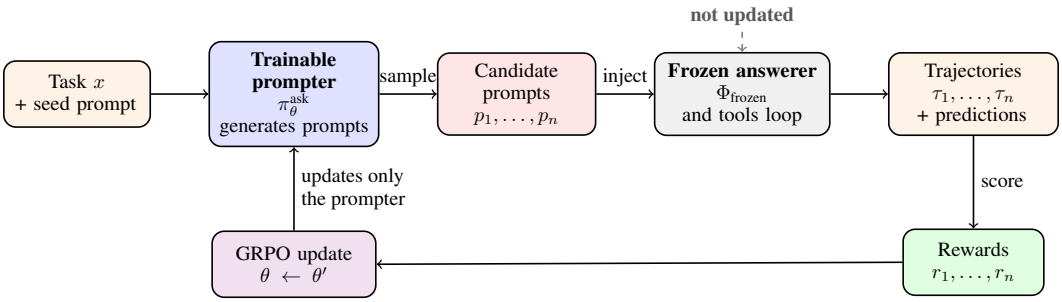


Figure 1: Prompt-side reinforcement learning pipeline. A trainable prompter π_θ^{ask} receives the task and seed prompt, samples a group of candidate prompts, and injects each prompt into the same frozen downstream answerer Φ_{frozen}. The answerer runs the tool-using episode and produces trajectories and predictions, which are scored by the task reward. GRPO uses these group rewards to update only the prompter parameters; the downstream answerer and tools remain fixed throughout training.

4 RL Methods

We compare two reinforcement learning regimes that use the same benchmarks, task splits, and GRPO-style group sampling procedure. They differ only in where the policy is placed. In *prompt-side RL*, a trainable model generates system prompts for a frozen downstream solver; the solver's weights remain fixed. In *answer-side RL*, the downstream solver is the trainable model, and the prompt is fixed throughout training.

4.1 Prompt-Side RL

For each training instance x_i, the prompt-generation policy $\pi_\theta^{\text{prompt}}$ receives a benchmark-specific generation request and samples a group of G candidate prompts:

$$\hat{p}_{ij} \sim \pi_\theta^{\text{prompt}}(\cdot \mid p_{\text{seed}},\, c(x_i),\, f_j)\,, \quad j = 1, \ldots, G,$$

where p_{seed} is the benchmark-specific seed prompt, $c(x_i)$ is the control context for instance x_i, and f_j is an optimization focus. Each sampled prompt is sanitized before execution: Markdown fences are removed, and when `<prompt>...</prompt>` tags are present, only the tagged content is retained. The sanitized prompt is then injected into the frozen solver Φ_{frozen}, which runs the full benchmark episode and returns a scalar reward.

Seed prompts. Each benchmark uses a seed prompt p_{seed} defined in Section 3.2; prompt-side RL generates candidate prompts conditioned on this seed and the instance control context.

Shaped training reward. Let r_{base} be the downstream reward returned by the frozen solver. For HoVer, r_{base} is top-5 supporting-title recall. For HotPotQA, r_{base} is the combined training reward $r_{\text{base}}^{\text{HotPotQA}}$, which averages normalized answer F1 and top-5 supporting-title recall. Let $\nu = 1 - \text{sim}(p_{\text{seed}}, \hat{p})$ be a string-similarity novelty score between the seed prompt and the sampled prompt. The shaped reward used during prompt-side training is

$$r_{\text{train}} = r_{\text{base}} + \lambda_{\text{novel}} \cdot \min(\nu, \nu_{\max}) \cdot r_{\text{base}}.$$

keeping downstream task performance as the primary signal while simultaneously encouraging prompt exploration.

4.2 Answer-Side RL

In answer-side RL the prompt is fixed to p_{seed} and the trainable model is the solver. For retrieval tasks, the policy acts within the same ReAct-style episode interface described in Algorithm 1. At each step, it observes the fixed prompt, task input, tool schema, and accumulated trajectory, then emits a structured JSON action:

$$a_t = \{\texttt{next_thought},\ \texttt{next_tool_name},\ \texttt{next_tool_args}\}.$$

Invalid tool requests force termination, and repeated non-`finish` calls with the same arguments trigger loop detection. The policy is optimized with GRPO using the benchmark reward from the completed episode.

4.3 GRPO Training Objective

We use Group Relative Policy Optimization (GRPO) for training [28]. For each task x_i, the policy samples G completions with rewards $\{r_{ij}\}_{j=1}^{G}$ and computes

$$\hat{A}_{ij} = \frac{r_{ij} - \mu_i}{\sigma_i + \epsilon}, \qquad \mu_i = \frac{1}{G}\sum_{j=1}^{G} r_{ij}, \quad \sigma_i^2 = \frac{1}{G}\sum_{j=1}^{G}(r_{ij} - \mu_i)^2.$$

Implementation. We use Agent Lightning and vERL for GRPO training [8, 29, 20], vLLM for rollouts [13], and FSDP with LoRA for actor updates [44, 10]. We use bf16 weights, gradient checkpointing, and offloading as needed.

5 Results

We compare optimization strategies under a fixed rollout budget. Let $\mathcal{M}$ denote an optimization method, B denote the maximum number of rollouts, and $J(\mathcal{M}; B)$ denote task performance after training method $\mathcal{M}$ using at most B rollouts. We write the budget-constrained objective as

$$\mathcal{M}^*(B) = \arg\max_{\mathcal{M}} J(\mathcal{M}; B) \quad \text{s.t.} \quad C(\mathcal{M}) \leq B,$$

where $C(\mathcal{M})$ is the number of downstream rollouts used during optimization.

We use standard train/validation/test splits for each benchmark. Gold supervision is never visible to the solver during rollout and is used only after episode termination to compute reward. We report mean top-5 title recall on HoVer, mean answer F1 on HotPotQA, and mean exact-match accuracy on GSM8K, each averaged over the held-out test split. We also use greedy decoding for prompt generation during evaluation.

Finding 1: Prompt-side GRPO improves retrieval-intensive tasks but not math reasoning. Table 1 shows that GRPO Asking improves performance on the two retrieval-intensive benchmarks relative to the baseline. On HoVer, it increases top-5 supporting-title recall by 7.33 points, and on HotPotQA, it increases answer F1 by 10.98 points and supporting-title recall by 10.55 points. HoVer and HotPotQA require the agent to decompose the input, retrieve evidence across multiple steps, and decide when to stop, all of which can be shaped by the generated prompt. In contrast, on GSM8K, GRPO Asking reduces exact-match accuracy, suggesting that prompt-side optimization is less useful when the task is solved through a single model call rather than a tool-using retrieval trajectory. In this setting, optimization appears more effective when applied to the answering model directly.

Qwen3 8B Performance Across Benchmarks			
Method	**HoVer**	**HotPotQA**	**GSM8K**
Task performance			
Baseline	33.67	44.75, 62.11	0.94
GRPO Asking	**41.00**	**55.73, 72.66**	0.89
GRPO Answering	40.33	43.77, 62.89	**0.95**
Total optimization budget (# rollouts)			
GRPO Asking	4800	8192	2000
GRPO Answering	4800	8192	2000

Table 1: Performance of prompt optimization methods on HoVer, HotPotQA, and GSM8K. HoVer is reported as top-5 supporting-title recall, HotPotQA as answer F1 and supporting-title recall, and GSM8K as exact-match accuracy.

HoVer			
Method	**1600**	**3200**	**4800**
Prompt-side optimization only			
GEPA	**35.00**	37.33	38.33
GRPO Asking	34.33	**39.00**	**41.00**
Difference	-0.67	+1.67	+2.67
Prompt-side + answer-side optimization			
GEPA + GRPO Answering	**44.33**	46.67	45.33
GRPO Asking + GRPO Answering	44.67	46.67	**47.33**
Difference	+0.34	+0	+2.00
HotPotQA			
Method	**2048**	**4096**	**8192**
Prompt-side optimization only			
GEPA	**46.18, 64.84**	46.96, 61.72	48.83, 63.67
GRPO Asking	43.78, 59.38	**47.12, 64.84**	**55.73, 72.66**
Difference	-2.40, -5.46	+0.16, +3.12	+6.90, +8.99

Table 2: Numerical sample-efficiency comparison across rollout budgets. Column headers denote the number of downstream rollouts used during optimization. Bold values indicate the better method within each comparison block, and "Difference" reports GRPO Asking minus GEPA. HotPotQA entries are reported as answer F1 followed by supporting-title recall. Answer-side GRPO ablations are reported only for HoVer because answer-side GRPO did not improve HotPotQA in Table 1.

Finding 2: Prompt-side GRPO improves more with additional rollout budget than GEPA. Table 2 shows that GRPO Asking scales more strongly with rollout budget than GEPA. On HoVer, GEPA improves modestly from 35.00 to 38.33 (+3.33) as the budget increases from 1600 to 4800 rollouts, while GRPO Asking improves from 34.33 to 41.00 (+6.67) over the same range. At the mid-range budget, the two methods are nearly tied, suggesting that GEPA remains competitive under moderate rollout budgets but does not show the same scaling trend. A similar pattern appears on HotPotQA: GRPO Asking trails GEPA at 2048 rollouts, is roughly tied at 4096, and leads by 6.90 answer-F1 points and 8.99 title-recall points at 8192. The larger recall gain suggests that additional prompt-side GRPO samples are especially useful for improving supporting-document retrieval. Overall, these results indicate that gradient-based prompt optimization benefits more from increased rollout budget than reflective search in the retrieval-heavy settings studied here.

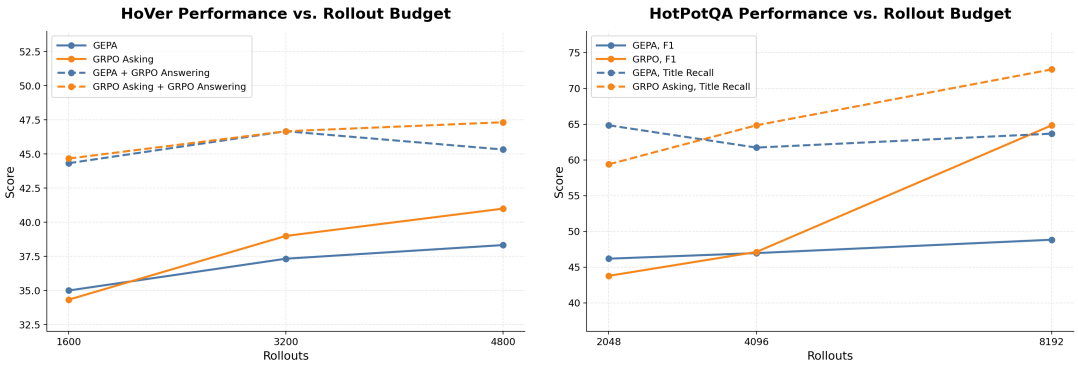


Figure 2: Rollout-budget scaling for prompt optimization on HoVer and HotPotQA. Each plot shows performance as the optimization budget increases, comparing GEPA with GRPO Asking under the prompt-side optimization setting.

Finding 3: Answer-side GRPO improves both prompt-optimization pipelines but does not remove the effect of the prompt optimizer. On HoVer, adding answer-side GRPO to either prompt optimizer consistently outperforms prompt-side optimization alone (Table 2). At a rollout budget of 4800, GEPA + GRPO Answering reaches 45.33 and GRPO Asking + GRPO Answering reaches 47.33, compared with 38.33 and 41.00 respectively without answer-side optimization. These gains are similar in magnitude (+7.00 and +6.33), suggesting that answer-side optimization provides a consistent additional benefit regardless of the prompt optimizer used. At the same time, the advantage of GRPO Asking over GEPA is preserved after adding answer-side optimization (+2.00 at budget 4800), indicating that prompt-side and answer-side optimization address partially distinct failure modes in the pipeline.

Finding 4: Task-conditioned prompts can adapt the retrieval strategy to the question type. As shown in Figure 3, GRPO Asking learns to generate qualitatively different retrieval strategies depending on the reasoning structure of the question, even though both prompt types share the same seed prompt and tool-use scaffold. For HotPotQA bridge questions, where answering requires chaining through an intermediate entity, the optimized prompt instructs the agent to perform sequential entity-chain retrieval: identify the starting entity, look it up, extract the implied bridge entity from the result, and then perform a second search. For comparison questions, where two entities must be evaluated against each other, the optimized prompt prescribes parallel evidence retrieval: identify both entities upfront, look each up independently, and collect evidence for both before finishing. The comparison-conditioned prompt also warns against starting with entity names not mentioned in the question, a constraint that would be harmful for bridge questions but helps prevent spurious retrievals in the comparison setting.

This task-conditioned specialization suggests that GRPO Asking does not simply add generic context to the prompt, but can induce different retrieval policies for different question types. This form of per-instance adaptation is difficult for fixed-prompt optimizers such as GEPA, which optimize one or more shared system prompts across tasks [1].

6 Conclusion

This work asks where a fixed rollout budget is best spent in a compound LLM system: optimizing the prompt generator, or the downstream solver. We formulate prompt-side optimization as an outer-loop RL problem in which a trainable policy generates task-conditioned prompts for a frozen ReAct agent. Using GRPO, we train the prompter while keeping the solver's weights fixed throughout.

On retrieval-heavy tasks where the solver must decompose problems, gather evidence across multiple steps, and decide when to stop, prompt-side RL can be a better use of a fixed compute budget than reflective search-based prompt optimization. On single-call reasoning tasks, the compute is better spent on the solver itself. Neither asking-side nor answering-side optimization dominates universally, so the choice should depend on the structure of the task.

More broadly, this work supports a modular view of adaptation in compound LLM systems. Capable behavior does not always require updating the model that directly answers the task. A trainable prompter paired with a frozen solver can improve system behavior under constrained rollout budgets, which matters in settings where solver retraining is expensive, inaccessible, or undesirable.

Limitations and future work. The current evaluation is limited to single-task training, a small benchmark suite, and a limited number of training runs. Future work should test cross-task generalization by training on multiple benchmarks and evaluating on held-out tasks, and should expand the benchmark suite to better separate retrieval-heavy tasks from single-call reasoning ones. The results may also depend on the choice of frozen solver, tool interface, seed prompt, and ReAct scaffold, so future experiments should test whether the same trends hold across different agent designs.

Comparisons with GEPA should be repeated in the modular, non-agentic setup used in the original GEPA evaluation. Additionally, retrieval metrics based on exact title matching may also undercount valid evidence titles, so future evaluations should include judge-based title validation. Finally, future work could ablate over GRPO hyperparameters, prompt initialization, optimization profiles, and hybrid prompt-generation methods such as retrieval augmentation (RAG) or evolutionary search.

Task-conditioned prompt specialization on HotPotQA

Bridge-conditioned generation

You are a multi-hop HotPotQA evidence agent. Your goal: retrieve the supporting paragraphs needed to answer the question, then finish.

Tools: search_wikipedia(query), lookup_wikipedia(title), finish(). Use exact titles returned by search. Do not repeat calls.

Steps:

1. For bridge questions, identify the chain of entities: the first entity is the starting point, often a known term or entity from the question.

2. Use the first entity to perform a search_wikipedia.

3. From the first lookup, identify the bridge entity or answer candidate (e.g., a person, organization, event) that is explicitly stated or implied in the paragraph.

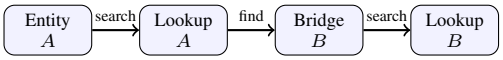
4. Use the bridge entity to perform another search_wikipedia.

5. Use the second lookup to find the relevant supporting paragraph (e.g., the final answer is in the paragraph).

6. finish(). Do not finish until both supporting titles are retrieved.

For comparison questions, retrieve the supporting paragraphs for each entity mentioned in the question and finish once both are retrieved.

For questions involving a chain of entities (e.g., "What is the relation between X and Y?" or "What is the birthplace of the director of Z?"), the chain must be followed step-by-step.



Sequential entity-chain retrieval

Comparison-conditioned generation

You are a multi-hop HotPotQA evidence agent. Your goal: retrieve the supporting paragraphs needed to answer the question, then finish.

Tools: search_wikipedia(query), lookup_wikipedia(title), finish(). Use exact titles returned by search. Do not repeat calls.

Steps:

1. Identify the main entities in the question.

2. If the question is a comparison, search_wikipedia for the first entity.

3. lookup_wikipedia to find supporting details.

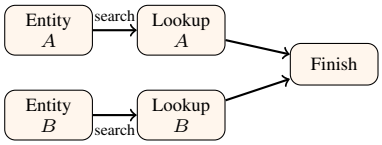
4. search_wikipedia for the second entity in the comparison.

5. lookup_wikipedia to find supporting details.

6. finish().

Do not start with entity names not mentioned in the question. For comparison questions, start only with the two entities being compared and retrieve evidence for both before deciding.

For bridge questions, when a new entity is discovered from a lookup, use that as the next search query.



Parallel evidence retrieval for compared entities

Figure 3: Task-conditioned prompt specialization on HotPotQA. The generated prompts from the GRPO Asking method share the same seed prompt and tool-use scaffold, shown in gray, but specialize different parts of the retrieval policy depending on the input task. Highlighted text is reproduced verbatim from the generated prompts. The bridge-conditioned prompt emphasizes sequential entity-chain traversal, while the comparison-conditioned prompt emphasizes collecting evidence for both compared entities before finishing.

References

[1] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] Yongchao Chen, Jacob Arkin, Yilun Hao, Yang Zhang, Nicholas Roy, and Chuchu Fan. Prompt optimization in multi-step tasks (promst): Integrating human feedback and heuristic-based sampling. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3859–3920, 2024.

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

[5] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3369–3391, 2022.

[6] David Dohan, Winnie Xu, Aitor Lewkowycz, Jacob Austin, David Bieber, Raphael Gontijo Lopes, Yuhuai Wu, Henryk Michalewski, Rif A Saurous, Jascha Sohl-Dickstein, et al. Language model cascades. *arXiv preprint arXiv:2207.10342*, 2022.

[7] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

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

[9] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

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

[11] Yichen Jiang, Shikha Bordia, Zheng Zhong, Charles Dognin, Maneesh Singh, and Mohit Bansal. Hover: A dataset for many-hop fact extraction and claim verification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3441–3460, 2020.

[12] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

[13] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[14] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 3045–3059, 2021.

[15] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[16] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, 2021.

[17] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys*, 55(9):1–35, 2023.

[18] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, 2022.

[19] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too. *AI open*, 5:208–215, 2024.

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

[21] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

[22] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[23] Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, 2024.

[24] 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.

[25] Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. Grips: Gradient-free, edit-based instruction search for prompting large language models. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3845–3864, 2023.

[26] Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 7957–7968, 2023.

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

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

[29] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297, 2025.

[30] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pages 4222–4235, 2020.

[31] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[32] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[33] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*, 2023.

[34] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

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

[36] Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. *Advances in Neural Information Processing Systems*, 37: 25981–26010, 2024.

[37] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2023.

[38] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018.

[39] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[40] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

[41] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077, 2024.

[42] Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. Offline training of language model agents with functions as learnable weights. In *Forty-first International Conference on Machine Learning*, 2024.

[43] Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E Gonzalez. Tempera: Test-time prompting via reinforcement learning. *arXiv preprint arXiv:2211.11890*, 2022.

[44] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

[45] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The eleventh international conference on learning representations*, 2022.