GEAR: Training-Free Rule Distillation for Advanced and Efficient Tool-Augmented Reasoning

Anonymous ACL submission

Abstract

Tool use enables AI agents to interact with external systems and accomplish real-world tasks beyond text generation. Large reasoning models (LRMs) excel at tool use but rely on test-time compute that dramatically increases inference latency and cost. Distilling LRM outputs to smaller models has emerged as an efficient alternative. However, most researchers and developers face resource constraints with only API access and no training compute, precluding distillation. We introduce **GEAR**, a training-free framework that extracts compositional rules from tool-use failures through meta-reasoning and distills them into inference prompts to guide model reasoning. We hypothesize that combining explicit rules with examples provides richer guidance than either alone. Across 78 API domains, **GEAR** achieves comparable performance to 3-shot prompting, while **GEAR** with only 2 examples outperforms 3-shot by 9.7% on the leading proprietary LRM (Claude-4.5-Sonnet), validating our complementarity hypothesis. Remarkably, **GEAR** enables Qwen3-30B to surpass the best Claude configuration by 15.7% while being 5× faster, demonstrating training-free democratization of tool-use capabilities. *Code publicly released upon publication.*

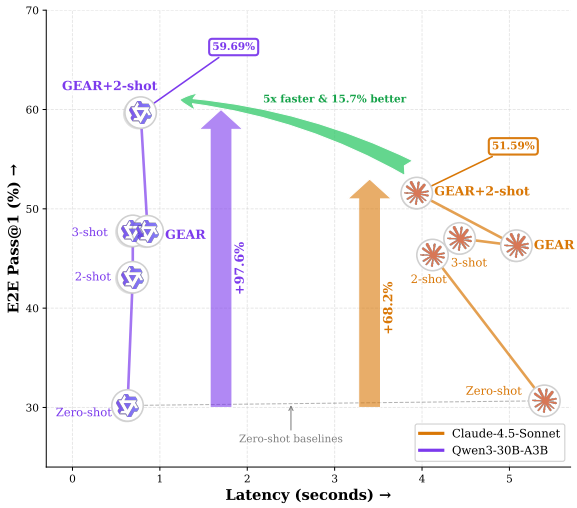

Figure 1: **GEAR** performance on Claude-4.5-Sonnet and Qwen3-30B. Qwen3-30B with **GEAR**+2-shot achieves the highest E2E Pass@1, surpassing Claude's best by 15.7% while being 5× faster, demonstrating training-free democratization of tool-use capabilities.

1 Introduction

Tool use has become essential for AI agents, enabling interaction with external systems for real-world tasks. Large reasoning models (LRMs) excel at in-distribution tool use but struggle with out-of-distribution (OOD) scenarios. Test-time compute techniques such as self-reflection (Yao et al., 2023b) and multi-path exploration (Yao et al., 2023a) address performance gaps but substantially increase latency and cost. Additionally, the proprietary (OpenAI, 2025; Anthropic, 2025; Google, 2025) or resource-intensive open-source (Guo et al., 2025; Yang et al., 2025; Team et al., 2025) nature of LRMs makes fine-tuning infeasible. Distillation approaches (Schick et al., 2023; Qin et al., 2024; Patil et al., 2024; Chen et al., 2024; Ning et al., 2024) train smaller models on LRM outputs but still require substantial training overhead, hindering democratization for resource-constrained practitioners. This creates an urgent need for training-free methods that improve performance without expensive test-time compute. Specifically, we target two objectives: (1) reducing real-time errors during tool execution, and (2) enabling generalization to out-of-distribution tool-use scenarios.

Leveraging in-context learning (ICL), LRMs can generalize to OOD tool-use scenarios in a training-free manner. Few-shot prompting (Brown et al., 2020), the most common ICL method, employs inductive learning through task-specific demonstrations. However, this approach is limited by combinatorially large failure spaces and context window constraints. Similarity-based retrieval (Zhang et al.,

2025) addresses coverage but introduces additional latency. We argue that tool-use failures exhibit recurring patterns revealing systematic reasoning gaps that can be captured more effectively through explicit rules. We hypothesize that combining explicit rules with fewer examples provides richer guidance than either alone or larger few-shot setup.

We introduce **GEAR** (**G**ather, **E**xtract, Rule-**A**ugmented **R**easoning), a training-free framework that operationalizes this hypothesis through three stages: (1) *gather* baseline predictions capturing both successes and failures, (2) *extract* compositional rules through meta-reasoning over failure patterns, and (3) *provide* rule-augmented reasoning by distilling rules into inference prompts. Central to **GEAR** is the rule extraction prompt, optimized through iterative LRM self-critique against a model-derived rubric. Unlike existing automated prompt optimization (APO) methods (Zhou et al., 2023; Yang et al., 2024), this process does not require performance feedback and runs for only single-digit iterations. Additionally, this optimization is domain-agnostic, meaning it needs to be run only once to generate the rule extraction prompt that can be used across all domains to extract and distill rules. This distillation is iteration-free and doesn't require performance feedback. This setup makes **GEAR** both time- and cost-efficient in every aspect from optimization to distillation, unlike the costly iterative optimization of APO methods.

We evaluate **GEAR** across 78 API domains. **GEAR** alone achieves comparable performance to 3-shot prompting, while **GEAR** with only 2 examples outperforms 3-shot by 9.7% on the best zero-shot proprietary LRM (Claude-4.5-Sonnet), validating our complementarity hypothesis. Remarkably, **GEAR** enables Qwen3-30B to surpass the best Claude configuration by 15.7% while being 5× faster, achieving nearly 2× the performance of both Qwen3-30B zero-shot and Claude zero-shot baselines (as shown in Figure 1), demonstrating training-free democratization of advanced tool-use capabilities.

2 Related Work

GEAR occupies a unique position in the landscape of tool-use improvement methods, combining training-free operation with cross-instance learning focused on failures.

Training-based approaches such as ToolFormer (Schick et al., 2023), ToolLLM (Qin et al., 2024), Gorilla (Patil et al., 2024), and others (Li et al., 2023; Chen et al., 2024; Ning et al., 2024) fine-tune models or distill LRM capabilities into smaller models. While effective, these methods require model weights and training compute unavailable to most practitioners. **GEAR** operates entirely through prompting with only API access.

Test-time reasoning methods including self-reflection (Yao et al., 2023b), multi-path exploration (Yao et al., 2023a), chain-of-thought (Wei et al., 2022), and self-consistency (Wang et al., 2023) improve per-query performance but increase latency. Per-instance reflection methods (Shinn et al., 2023; Madaan et al., 2023; Bai et al., 2022) generate trajectory-specific feedback requiring replay. In contrast, **GEAR** performs cross-instance learning, extracting reusable rules from historical failures that transfer across queries without replay.

To enhance ICL, existing work (Zheng et al., 2025; Stengel-Eskin et al., 2024; Wang et al., 2024; Murty et al., 2024; Zhang et al., 2025; Qian et al., 2023; Shang et al., 2025) acquires reusable artifacts like code, skills, and APIs through exploration. As these repositories grow, retrieval mechanisms (Lewis et al., 2020; Zhang et al., 2025) become necessary to select relevant components per query, introducing additional latency. **GEAR** instead extracts compact behavioral constraint rules from failures that are statically included in prompts without retrieval, complementing few-shot ICL through error prevention than capability expansion.

APO methods (Zhou et al., 2023; Yang et al., 2024; Fernando et al., 2024; Opsahl-Ong et al., 2024) refine instructions and demonstrations through expensive, time-consuming meta-prompting, evolutionary search, or reflective evolution, with GEPA (Agrawal et al., 2025) learning high-level rules from trajectories. However, all require performance feedback from enormous rollouts and embed learned knowledge into monolithic prompts. **GEAR** differs fundamentally: it extracts explicit, compositional rules without any performance feedback during both meta-prompt optimization (single-digit iterations, run once across all domains) and rule extraction (rollout-free), targeting real-time error reduction and OOD generalization through cross-instance failure analysis. Additionally, **GEAR** outputs interpretable, modular rules that complement few-shot examples, enabling democratization for resource-constrained practitioners with a training-free, cost-efficient approach.

3 Problem Formulation

We formalize tool-augmented reasoning as a sequential decision-making task where a language agent orchestrates external API calls to answer user queries.

3.1 Tool-Use as a Markov Decision Process

We model tool-use as an episodic Markov Decision Process (MDP), comprising state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $\mathcal{P}$, and reward function r. At each timestep t, the agent observes state $s_t = (\mathcal{I}, \mathcal{D}_{\text{API}}, q, \mathcal{H}_t)$, comprising system instructions $\mathcal{I}$, API documentation $\mathcal{D}_{\text{API}}$, user query q, and trajectory history $\mathcal{H}_t = \langle(\alpha_0, o_0), \ldots, (\alpha_{t-1}, o_{t-1})\rangle$ of previous action-observation pairs. The agent generates an action $\alpha_t = (a_t, \theta_t)$ consisting of an API identifier a_t and corresponding parameters θ_t, following a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maps states to actions. This process continues until a termination action provides the final answer.

While the deployment goal is producing correct final answers through full trajectory execution, this work focuses on **per-step action prediction accuracy**: given a state s_t with ground-truth action α_t^*, we seek policies that maximize $\mathbb{P}[\pi(s_t) = \alpha_t^*]$. This paradigm, inspired by step-wise preference learning (Chen et al., 2024), provides fine-grained assessment that better aligns with how LLMs reason over sequential decisions. See Appendix A for complete formalization.

3.2 Rule-Augmented Formulation

Contemporary approaches implement the policy π through in-context learning (ICL), employing inductive reasoning over demonstration examples to generalize to new queries. We hypothesize that combining inductive learning with deductive reasoning over explicit constraints provides richer guidance than either alone. While few-shot learning requires models to infer patterns from examples, explicit rules enable direct constraint application. We formalize baseline approaches in Appendix B.

GEAR augments the state with explicit domain-specific behavioral rules $\mathcal{R} = \{r_1, r_2, \ldots, r_{|\mathcal{R}|}\}$, yielding $s_t = (\mathcal{I}, \mathcal{D}_{\text{API}}, \mathcal{R}, q, \mathcal{H}_t)$. These rules remain static across all queries within the same domain. Each rule $r_i = (\text{Directive}_i, \text{Pedagogical}_i)$ comprises two components: the directive specifies when the rule applies and the required behavior (WHEN-THEN structure), while the pedagogical component illustrates correct and incorrect behaviors along with test criteria for verification. This structure enables deductive constraint application through both explicit directives and concrete behavioral guidance. **GEAR** is complementary to few-shot learning: combining rules with demonstration examples $\mathcal{E}$ yields $s_t = (\mathcal{I}, \mathcal{D}_{\text{API}}, \mathcal{R}, \mathcal{E}, q, \mathcal{H}_t)$, leveraging both deductive and inductive reasoning as validated empirically in Section 6.

3.3 Rule Extraction Problem

The rule extraction problem seeks a compact rule set $\mathcal{R}^*$ that maximizes per-step accuracy on a held-out test set $\mathcal{D}_{\text{test}}$: $\mathcal{R}^* = \arg\max_{\mathcal{R}} \text{Accuracy}(\pi_{\mathcal{R}} \mid \mathcal{D}_{\text{test}})$ subject to three constraints: interpretability (natural language rules), generalization (class-level patterns rather than instance memorization), and compactness (minimal rule count). This is a discrete combinatorial optimization over natural language rule sets, precluding gradient-based methods. Section 4 describes our single-shot meta-reasoning approach to this problem.

3.4 Meta-Prompt Creation Problem

The quality of extracted rules depends critically on the meta-prompt $\mathcal{P}_{\text{meta}}$ used for rule extraction. We formulate meta-prompt optimization as finding an optimal meta-prompt $\mathcal{P}^*_{\text{meta}}$ that maximizes rule effectiveness: $\mathcal{P}^*_{\text{meta}} = \arg\max_{\mathcal{P}} \mathbb{E}_{\mathcal{R}\sim\text{Extract}(\mathcal{P},\mathcal{D}_{\text{analysis}})}[\text{Accuracy}(\pi_{\mathcal{R}} \mid \mathcal{D}_{\text{test}})]$ where $\mathcal{D}_{\text{analysis}}$ contains baseline predictions for failure analysis. Evaluating this objective requires expensive rule extraction and inference for each candidate prompt, making exhaustive search intractable. Section 4.1 describes our efficient LRM-based self-critique approach that optimizes without external feedback.

4 GEAR Framework

GEAR operates through two stages as illustrated in Figure 2: an offline meta-prompt creation stage performed once, followed by a per-domain stage that extracts and deploys rules for each API domain.

Offline Stage (Once). We create a domain-agnostic meta-prompt $\mathcal{P}^*_{\text{meta}}$ through iterative self-critique (Section 4.1). This meta-prompt guides rule extraction and is reused across all domains without modification.

Per-Domain Stage. For each API domain, **GEAR** executes three phases using the fixed

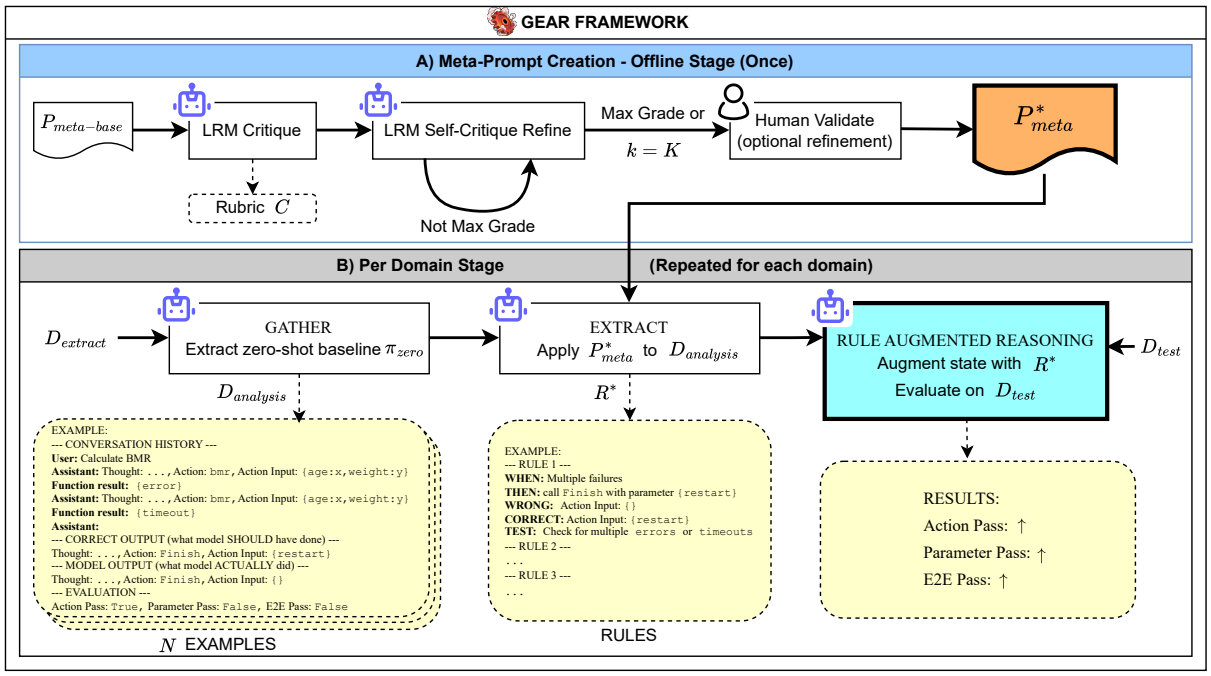

Figure 2: **GEAR** framework overview. The **offline stage** creates a domain-agnostic meta-prompt $\mathcal{P}^*_{\text{meta}}$ through iterative LRM self-critique with optional human validation. The **per-domain stage** executes three phases: Gather (collect baseline predictions), Extract (synthesize rules via meta-reasoning), and Rule-Augmented Reasoning (evaluate with rule-augmented state). The meta-prompt is created once and reused across all domains.

Algorithm 1 Meta-Prompt Creation

Input: Base meta-prompt $\mathcal{P}_{\text{meta-base}}$, iteration limit K
Output: Optimized meta-prompt $\mathcal{P}^*_{\text{meta}}$
// Stage 1: Rubric Generation and Initial Critique
$(\mathcal{C}, \{g_i^{(0)}\}) \leftarrow$ LRM-Critique($\mathcal{P}_{\text{meta-base}}$)
$c_0 \leftarrow (\mathcal{C}, \{g_i^{(0)}\})$; $\mathcal{P}_0 \leftarrow \mathcal{P}_{\text{meta-base}}$
// Stage 2: Iterative Refinement
for $k = 1$ to K **do**
  $\mathcal{P}_k \leftarrow$ LRM-Refine($\mathcal{P}_{k-1}, c_{k-1}$)
  $\{g_i^{(k)}\} \leftarrow$ LRM-SelfCritique($\mathcal{P}_k, \mathcal{C}$)
  $c_k \leftarrow (\mathcal{C}, \{g_i^{(k)}\})$
  if MaxGrade($\{g_i^{(k)}\}$) **then**
    break
  end if
end for
// Human Validation (optional)
$\mathcal{P}^*_{\text{meta}} \leftarrow$ HumanValidate($\mathcal{P}_K$)
return $\mathcal{P}^*_{\text{meta}}$

meta-prompt $\mathcal{P}^*_{\text{meta}}$: Gather, Extract, and Rule-Augmented Reasoning (Sections 4.2–4.4). Figure 2 illustrates the complete pipeline.

4.1 Meta-Prompt Creation

Rule extraction quality depends critically on the meta-prompt design. Traditional prompt optimization approaches evaluate performance on train data, requiring expensive inference for each candidate. **GEAR** instead uses LRM-based self-critique for efficient refinement without external feedback.

Algorithm 1 outlines our two-stage process. In Stage 1, an LRM generates an evaluation rubric $\mathcal{C}$ and critiques a base meta-prompt $\mathcal{P}_{\text{meta-base}}$. In Stage 2, the LRM iteratively refines this prompt using self-critique against the fixed rubric $\mathcal{C}$ until reaching the maximum grade or iteration limit K. This process operates without feedback from actual rule extraction or train data performance, making it highly efficient.

A human expert (here authors) performs final validation, a sanity check, as an optional but recommended step, verifying structural correctness (input placeholders, output schema) since LRMs can be overconfident in self-assessment. This lightweight validation maintains the training-free nature of **GEAR** while ensuring the resulting domain-agnostic meta-prompt $\mathcal{P}^*_{\text{meta}}$ is properly structured for reuse across all domains. The $\mathcal{P}_{\text{meta-base}}$, $\mathcal{P}^*_{\text{meta}}$, and self-critique prompts are provided in Appendix C.

4.2 Phase 1: Gather

We partition data into an extraction set $\mathcal{D}_{\text{extract}}$ for rule extraction and a held-out test set $\mathcal{D}_{\text{test}}$ for evaluation. We execute a zero-shot baseline policy π_{zero} on $\mathcal{D}_{\text{extract}}$ to generate the analysis dataset: $\mathcal{D}_{\text{analysis}} = \{(s_t^{(i)}, \alpha_{\text{pred}}^{(i)}, \alpha_*^{(i)}, e^{(i)})\}_{i=1}^{N}$ where $\alpha_{\text{pred}}^{(i)}$

is the predicted action, $\alpha_*^{(i)}$ is the ground-truth action, and $e^{(i)}$ is a binary correctness indicator. Importantly, $\mathcal{D}_{\text{analysis}}$ includes both successful and failed predictions, enabling contrastive analysis and preventing regression (as discussed in Section 8) in the Extract phase.

4.3 Phase 2: Extract

The Extract phase applies the optimized meta-prompt $\mathcal{P}^*_{\text{meta}}$ to $\mathcal{D}_{\text{analysis}}$ in a single inference call to synthesize domain-specific rules through automated meta-reasoning. The meta-prompt guides the LRM to: (1) categorize failures by type, (2) identify recurring patterns across multiple examples (frequency threshold $\geq \tau$), (3) determine root causes through comparative analysis of correct versus incorrect predictions, and (4) synthesize rules following the two-tuple structure $r_i = (\text{Directive}_i, \text{Pedagogical}_i)$ defined in Section 3.2.

The extraction produces domain-specific rules $\mathcal{R}^* = \{r_1, \ldots, r_{|\mathcal{R}^*|}\}$, typically 5–15 rules per domain depending on failure pattern diversity. Critically, the analysis process and rule structure requirements are specified within $\mathcal{P}^*_{\text{meta}}$ itself, demonstrating **GEAR**'s ability to leverage LRM metacognitive capabilities for end-to-end automated knowledge extraction. Example extracted rules and complete schema are provided in Appendix D.

4.4 Phase 3: Rule-Augmented Reasoning

The Rule-Augmented Reasoning phase deploys extracted rules during inference on $\mathcal{D}_{\text{test}}$ by augmenting the agent's state to $s_t = (\mathcal{I}, \mathcal{D}_{\text{API}}, \mathcal{R}^*, q, \mathcal{H}_t)$ as defined in Section 3.2. Rules guide both action selection and parameter population, requiring no additional computation beyond the standard forward pass. Since rules are modular, they can be combined with other techniques such as few-shot learning or deployed compositionally by activating subsets based on priority assigned according to estimated error coverage during extraction.

5 Experimental Setup

5.1 Dataset

We use the ToolPreference dataset (Chen et al., 2024), which augments ToolBench (Qin et al., 2024) trajectories with preference pairs for tool-use evaluation. This dataset is well-suited for our goals: it provides preference signals necessary for per-step evaluation, includes failed API calls with recovery steps reflecting real-world error conditions, and spans diverse API domains for out-of-distribution evaluation—unlike the original ToolBench or Toucan (Xu et al., 2025) which lack explicit action-level preferences. We apply LLM-as-judge filtering to remove noisy preference pairs where neither action represents an optimal next step (details in Appendix E). After filtering, we retain 1,949 examples across 78 API domains, with 70:30 split into $\mathcal{D}_{\text{extract}}$ (n=1,397) for rule extraction and $\mathcal{D}_{\text{test}}$ (n=552) for evaluation. Dataset statistics are provided in Appendix F.

5.2 Models and Configuration

We evaluate **GEAR** across both proprietary and open-source models. **Proprietary models:** Claude-4.5-Sonnet, GPT-5.1, and Gemini-2.5-Flash represent state-of-the-art commercial LLMs. **Open-source models:** Qwen3-4B, Qwen3-30B and Ministral3-14B represent efficient alternatives for resource-constrained deployments. Proprietary models are evaluated in two modes: *standard mode* (temperature T=0 for greedy decoding) and *thinking mode* enabling test-time reasoning (temperature T=1 as recommended by providers). Since thinking mode incurs substantial latency overhead and we optimize for efficiency, open-source models are evaluated in standard mode only. Model details are provided in Appendix G.

Rule extraction (Section 4.3) uses Claude-4.5-Sonnet with frequency threshold τ=3 (extracting rules only for patterns appearing in $\geq$3 examples), producing 6–12 rules per domain depending on failure diversity. To compare which mode produces higher-quality rules, we perform extraction using both standard and thinking modes. To account for minor non-determinism in LLM-based extraction despite greedy decoding, we conduct three runs per mode and select the best-performing rule set among all six runs based on performance on $\mathcal{D}_{\text{extract}}$.

5.3 Baselines

All prompts follow the ReAct output format (Yao et al., 2023b), where models generate reasoning (Thought) before tool invocation (Action). Models observe previous Thought-Action-Observation tuples in the history $\mathcal{H}_t$. We compare **GEAR** against five baselines: (1) **Zero-shot**: standard ReAct prompt without examples or rules, (2) **Zero-shot-CoT**: adds an explicit step-by-step reasoning instruction, and (3–5) **1/2/3-shot**: one, two, or three randomly sampled examples from $\mathcal{D}_{\text{extract}}$ to measure scaling behavior of example-

Method	Pass@1 (%)			Gain (%)	Efficiency			
	Action	Parameter	E2E	ΔE2E (%)	Latency (s)	ΔLat. (%)	In Tok.	Out Tok.
Claude-4.5-Sonnet *(Standard | Thinking)*								
Zero-shot	62.76 | 64.49	40.04 | 38.63	30.67 | 30.36	– | −1.0	5.40 | 7.57	– | −40.2	1,445 | 1,474	179 | 416
Zero-shot-CoT	62.52 | 64.52	38.14 | 38.23	28.70 | 30.58	−6.4 | −0.3	5.69 | 9.64	−5.4 | −78.5	1,459 | 1,488	246 | 504
1-shot	67.64 | 67.19	49.90 | 52.18	39.72 | 42.33	+29.5 | +38.0	4.05 | **7.01**	+24.9 | **−29.8**	2,041 | 2,070	136 | 356
2-shot	71.07 | 70.80	54.93 | 56.36	45.35 | 48.13	+47.9 | +56.9	4.12 | 7.04	+23.7 | −30.4	2,665 | 2,693	122 | 359
3-shot	73.30 | 73.96	55.74 | 58.38	47.02 | 50.04	+53.3 | +63.2	4.43 | 7.29	+17.9 | −35.1	3,340 | 3,368	123 | 364
GEAR (ours)	73.14 | 73.15	53.20 | 54.03	46.29 | 46.89	+50.9 | +52.9	5.08 | 8.81	+5.9 | −63.2	3,131 | 3,184	179 | 503
Critical-**GEAR** (ours)	72.41 | 65.72	52.79 | 46.51	45.52 | 36.13	+48.4 | +17.8	4.18 | 8.35	+22.6 | −54.8	2,052 | 2,107	176 | 465
GEAR+2-shot (ours)	**75.78** | **79.31**	**58.54** | **62.04**	**51.59** | **56.75**	**+68.2** | **+85.0**	3.94 | 7.74	+27.0 | −43.4	4,336 | 4,924	145 | 489
2-shot+**GEAR** (ours)	74.94 | 72.04	58.38 | 57.21	50.46 | 47.95	+64.5 | +56.3	3.91 | 8.40	+27.6 | −55.6	4,320 | 4,397	147 | 472
DirectiveOnly-**GEAR**	68.90 | 67.92	49.62 | 48.19	41.67 | 39.63	+35.9 | +29.2	4.72 | 8.61	+12.5 | −59.4	2,331 | 2,378	201 | 523
PedagogyOnly-**GEAR**	69.80 | 66.26	48.61 | 50.43	40.66 | 39.97	+32.6 | +30.3	**3.82** | 7.84	**+29.2** | −45.3	2,279 | 2,315	170 | 483
Qwen3-30B-A3B *(Standard only)*								
Zero-shot	57.53	42.72	30.20	−1.5†	**0.63**	**+88.4**†	1,299	83
2-shot	66.43	54.05	43.11	+40.6†	0.69	+87.2†	2,366	89
3-shot	70.97	58.00	47.78	+55.8†	0.69	+87.3†	2,961	90
GEAR (ours)	72.81	56.25	47.74	+55.7†	0.85	+84.2†	2,850	114
GEAR+2-shot (ours)	**82.25**⋆	**65.24**⋆	**59.69**⋆	**+94.6**⋆†	0.78	+85.6†	4,446	101

Table 1: **GEAR** results on $\mathcal{D}_{\text{test}}$. E2E Pass@1 (%) is the primary metric. For Claude-4.5-Sonnet, values shown as Standard | Thinking. ΔE2E = relative improvement over Zero-shot; ΔLat. = speedup (positive = faster). Best results per model per mode in **bold**; ⋆overall best across all configurations. †Relative to Claude-4.5-Sonnet Zero-shot (Standard). Row colors: **GEAR** variants; ablation studies. Full results in Appendix I.

based learning.[1] We exclude sampling-based and reflection methods as they significantly increase latency, and APO methods as they require performance feedback and iterative optimization incompatible with our efficiency-focused, feedback-free setup. All prompts are provided in Appendix H.

5.4 GEAR Variants

Beyond core **GEAR**, we evaluate two variants testing deployment flexibility:

Complementarity. Rules enable deductive constraint application while examples enable inductive pattern inference. We hypothesize that these distinct reasoning mechanisms offer complementary benefits. Some failures may be better addressed by explicit constraints, others by pattern demonstration, and for shared failures, both mechanisms may reinforce each other. To test this, we evaluate **GEAR**+2-shot, which combines extracted rules with few-shot examples in the prompt. If rules and examples address distinct failure modes, their combination should yield improvements beyond either approach alone. Additionally, we test whether *ordering* affects performance by comparing **GEAR**+2-shot (rules followed by examples) against 2-shot+**GEAR** (examples followed by rules). We hypothesize that placing rules first establishes explicit constraints that frame the action space, allowing subsequent examples to demonstrate *how* to satisfy those constraints, a "constrain-then-demonstrate" ordering.

Compositionality. For efficiency-sensitive applications, rules can be deployed compositionally by activating subsets based on priority. During extraction, rules are categorized into priority buckets (critical, high, medium, low) based on estimated error coverage. We evaluate Critical-**GEAR**, which deploys only critical-priority rules, as one instantiation of this approach. Other threshold choices (e.g., all except low-priority) may yield different trade-offs. We hypothesize that Critical-**GEAR** will reduce latency while retaining most of **GEAR**'s effectiveness, as high-priority rules capture the majority of the error distribution.

5.5 Evaluation Metrics

We measure correctness using three Pass@1 metrics: **Action** (correct API), **Parameter** (exact match of all parameter names and values, order-independent), and **E2E** (complete action correctness, our primary metric). Efficiency is measured via **latency** (seconds from input to final token). We report single run mean across 78 domains in $\mathcal{D}_{\text{test}}$.

6 Results

We evaluate **GEAR** on $\mathcal{D}_{\text{test}}$, comparing against baselines on both effectiveness (Pass@1) and efficiency (latency). Table 1 presents our results with

[1] We limit to 3-shot as 55% of domains contain only 6 test examples, and higher counts risk demonstration-test overlap.

E2E as the primary metric, crucial for real-world AI-agent deployment where partial correctness still constitutes failure. Due to space constraints, we focus on Claude-4.5-Sonnet and Qwen3-30B for standard mode. Additional model results showing similar trends are provided in Appendix I.

Supporting our complementarity hypothesis, **GEAR** combined with few-shot examples achieves the best performance across both models, confirming that deductive guidance (rules) and inductive learning (examples) provide complementary benefits, whether by addressing different failures or reinforcing each other on shared ones. On Claude, **GEAR**+2-shot achieves 51.59% E2E (68.2% over zero-shot), yielding 11.4% relative improvement over GEAR alone and 13.8% over 2-shot alone.

On Qwen, the complementarity is even stronger: **GEAR**+2-shot achieves 59.69% E2E, the highest performance across all models and configurations tested, outperforming 3-shot by 24.92% and surpassing the best proprietary result (Claude **GEAR**+2-shot) by 15.7% while being 5× faster (0.78s vs 3.94s). While latency gains primarily reflect model size (30B vs presumably much larger), the performance advantage demonstrates that **GEAR**'s explicit guidance effectively amplifies smaller models' instruction-following capabilities, enabling open-source models to surpass proprietary alternatives. Ordering also matters: **GEAR**+2-shot outperforms 2-shot+**GEAR** by 2.24% on Claude (51.59% vs 50.46%), supporting our "constrain-then-demonstrate" hypothesis, where rules frame the problem space and examples demonstrate solutions within it.

GEAR alone achieves 50.92% relative improvement over zero-shot on Claude (46.29% vs 30.67%) and 58.1% on Qwen (47.74% vs 30.20%). Gains are concentrated in Parameter Pass@1 (32.9% on Claude, 31.7% on Qwen) compared to Action Pass@1, suggesting explicit rules are particularly effective at addressing parameter formatting errors. While 3-shot slightly outperforms **GEAR** on Claude (47.02% vs 46.29%), they are effectively tied on Qwen (47.78% vs 47.74%), indicating smaller models benefit as much from rules as from additional examples.

Few-shot scaling exhibits diminishing returns: on Claude, the third example adds only 1.67pp compared to 5.63pp for the second, while increasing latency by 7.5% (4.12s to 4.43s). **GEAR** offers efficient alternatives at multiple operating points: Critical-**GEAR** matches 2-shot performance with 23% fewer tokens, while **GEAR**+2-shot outperforms 3-shot on both models (4.57pp on Claude, 11.91pp on Qwen) with lower latency, confirming that augmenting with rules is more effective than scaling with examples. Interestingly, **GEAR**+2-shot achieves lower latency than **GEAR** alone despite higher input tokens on both Claude (3.94s vs 5.08s) and Qwen (0.78s vs 0.86s). We hypothesize this reflects increased generation confidence from combined guidance, as evidenced by the aforementioned accuracy gains alongside fewer output tokens (145 vs 180 on Claude, 101 vs 114 on Qwen).

7 Analysis

We conduct additional analyses to understand when and why **GEAR** works, examining the impact of thinking mode across extraction and evaluation, and how performance varies with task complexity.

7.1 Standard vs Thinking Mode

During the Extract phase (Section 4.3), thinking-extracted rules consistently outperform standard-extracted across all models, achieving 41.64% average E2E versus 36.99% for standard extraction and winning in 63.8% of domain-model combinations (444/696). This advantage persists even when rules are evaluated in standard mode, suggesting extended reasoning enables more thorough failure analysis. Interestingly, quantitative analysis reveals that thinking- and standard-extracted rules are similar in count (9.1 vs 9.3 average) and length (749 vs 735 average characters), indicating the performance gap stems from rule quality rather than quantity. Additionally, extractor model capability matters: using Qwen3-30B for extraction yields 41.23% E2E compared to 47.74% with Claude extraction (both evaluated on Qwen3-30B), confirming that more capable models produce higher-quality rules even when smaller models are used for inference. Investigating what constitutes higher-quality rules is beyond the scope of this work and remains an avenue for future research. Complete extraction mode analysis is provided in Appendix J.

For evaluation, thinking mode shows model-dependent results. On GPT-5.1, thinking substantially improves zero-shot (35.68% vs 27.24%, 31.0% relative) but incurs 7.2× latency overhead (18.47s vs 2.56s). On Gemini-2.5-Flash, gains are modest across all methods, with few-shot benefiting more (3-shot: 5.8% relative) than zero-shot

Statement-style ($\mathcal{P}_{\text{meta-base}}$) – No improvement When an API call fails (timeout, 500 error, service unavailable), try alternative endpoints or different parameter values before giving up. If multiple functions fail consistently, proceed to Finish with an explanation.
Structured ($\mathcal{P}^*_{\text{meta}}$) – Consistent gains **WHEN** API response contains 'status_code=503' or 'timeout' **THEN** Try one alternative function if available; if alternative also fails, call Finish. Don't retry same function. **Wrong:** Retried same function after 503. **Correct**: Tried alternative once, then called Finish.

Table 2: Rule format comparison.

(2.0%) or **GEAR** (4.7%). On Claude, thinking improves few-shot methods (2-shot: 6.1% relative) but shows no gains for zero-shot. Across all models, **GEAR**'s gains from thinking are modest compared to few-shot methods, suggesting that explicit rule guidance already provides much of the benefit that extended reasoning offers. Combined with substantial latency overhead, standard-mode **GEAR** remains the optimal choice for latency-sensitive deployments (full results in Appendix I).

7.2 Complexity Analysis

We analyze how performance varies with task complexity on Claude-4.5-Sonnet, operationalized as the number of previous API calls (hops) in the trajectory history. As expected, all methods exhibit performance degradation with increasing complexity, with E2E Pass@1 declining from 39-71% at hop 0 to 12-38% at hop 5, while latency and cost increase approximately linearly due to longer context. The relative ordering of methods remains consistent across complexity levels, with **GEAR**+2-shot achieving the best tradeoff between performance, latency, and cost throughout.

We also examine action space complexity, measured by the number of available APIs per domain (ranging from 3 to 11). Unlike trajectory complexity, performance does not exhibit a clear degradation pattern with increasing action space size $|\mathcal{A}|$, suggesting that API documentation provides sufficient guidance for action selection regardless of the number of alternatives. **GEAR**+2-shot maintains consistently strong performance across $|\mathcal{A}|$ while achieving competitive latency. Complete analysis with visualizations is provided in Appendix K.

8 Ablation Studies

We validate **GEAR**'s design choices through three ablations on Claude-4.5-Sonnet (Standard mode).

Rule Components. Each rule comprises a directive and pedagogical component (Section 3.2). DirectiveOnly-**GEAR** achieves 41.67% E2E (35.9% relative over zero-shot), capturing 70.5% of full **GEAR**'s improvement. PedagogyOnly-**GEAR** achieves 40.35% E2E (31.6% relative), capturing 62.1% of the improvement (see Table 1). Neither component alone matches full **GEAR** (46.29%), confirming their complementary roles: directives specify *what* to do and *when*, while the pedagogical component clarifies *how* through correct/incorrect behaviors.

Rule Format. In preliminary experiments, we compared statement-style rules from $\mathcal{P}_{\text{meta-base}}$ against structured heuristics from $\mathcal{P}^*_{\text{meta}}$ with explicit WHEN-THEN conditions, concrete examples, and verification criteria (Table 2). Statement-style rules yielded no improvement over zero-shot, while structured rules produced consistent gains, motivating our iterative meta-prompt refinement.

Contrastive Extraction. In preliminary experiments, we tested extracting rules from failed predictions only versus both successes and failures. Failure-only extraction caused 1–2% regression below zero-shot, as rules became overly restrictive and prohibited valid behaviors. Including successful predictions enables discriminative pattern identification, preventing such overfitting. This motivated including both outcomes in $\mathcal{D}_{\text{analysis}}$.

9 Conclusion

We introduced **GEAR**, a training-free framework that extracts compositional rules from tool-use failures through meta-reasoning and distills them into inference prompts. Evaluated across 78 API domains, **GEAR** matches 3-shot baselines, while **GEAR** combined with few-shot outperforms them, achieving the highest results across all models and configurations, validating our key hypothesis that deductive guidance (rules) and inductive learning (examples) provide complementary benefits. Most remarkably, Qwen3-30B with **GEAR**+2-shot achieves the best overall performance, surpassing all proprietary LRMs while being 5× faster, demonstrating that explicit guidance can amplify smaller models' capabilities beyond larger proprietary alternatives. We present a principled approach to reduce real-time tool-use errors and latency, democratizing advanced tool-use capabilities for resource-constrained practitioners.

10 Limitations

GEAR's design prioritizes resource efficiency, which involves deliberate trade-offs. It neither uses feedback from data to create the rule extraction prompt nor to refine extracted rules. Unlike existing automatic prompt optimization methods that iterate with performance feedback, **GEAR**'s single-shot approach may miss opportunities for refinement. While this significantly reduces computational overhead, incorporating lightweight feedback mechanisms could improve rule quality and remains an avenue for future work.

Our evaluation uses per-step action prediction rather than full trajectory execution. We take inspiration from the ToolPreference dataset design (Chen et al., 2024), where step-wise preference pairs were deliberately chosen over path-wise alternatives for training. Chen et al. (2024) argue that step-wise supervision provides fine-grained process feedback that better matches how LLMs reason: inferring the next API call based on previous execution responses. Path-wise supervision, in contrast, only differentiates correct from incorrect final responses, limiting generalization to unseen instructions. Extending this reasoning to evaluation, per-step assessment captures the quality of individual decisions that compound into trajectory success. Furthermore, each evaluation instance includes complete trajectory history $\mathcal{H}_t$ across varying hop lengths, capturing decision-making at different trajectory depths. While per-step accuracy may not perfectly correlate with end-to-end success, our consistent improvements across 78 domains suggest robust findings.

11 AI Assistants in Research or Writing

We used Claude 4.5 Sonnet and Claude 4.5 Opus throughout this work for paired programming, proofreading and refining manuscript text, brainstorming ideas for figures and tables, and generating visualization code. All AI-generated content was reviewed, verified, and edited by the authors, who retain full responsibility for the final manuscript.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2025. Gepa: Reflective prompt evolution can outperform reinforcement learning. *Preprint*, arXiv:2507.19457.

Anthropic. 2025. Introducing claude sonnet 4.5. Published: 2025-09-25.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, and 32 others. 2022. Constitutional ai: Harmlessness from ai feedback. *Preprint*, arXiv:2212.08073.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Sijia Chen, Yibo Wang, Yi-Feng Wu, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. 2024. Advancing tool-augmented large language models: Integrating insights from errors in inference trees. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. Promptbreeder: self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Google. 2025. Gemini 2.5: Our most intelligent ai model. Published: 2025-05-25.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA. Association for Computing Machinery.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: iterative refinement with self-feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Mistral. 2025. Introducing mistral 3. Published: 2025-12-2.

Shikhar Murty, Christopher D. Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. 2024. Bagel: bootstrapping agents by guiding exploration with language. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Xuefei Ning, Zifu Wang, Shiyao Li, Zinan Lin, Peiran Yao, Tianyu Fu, Matthew B. Blaschko, Guohao Dai, Huazhong Yang, and Yu Wang. 2024. Can llms learn by teaching for better reasoning? a preliminary study. In *Advances in Neural Information Processing Systems*, volume 37, pages 71188–71239. Curran Associates, Inc.

OpenAI. 2025. Gpt-5.1: A smarter, more conversational chatgpt. Published: 2025-11-12.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. 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, Miami, Florida, USA. Association for Computational Linguistics.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: large language model connected with massive apis. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

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

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. 2025. Agentsquare: Automatic LLM agent search in modular design space. In *The Thirteenth International Conference on Learning Representations*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Elias Stengel-Eskin, Archiki Prasad, and Mohit Bansal. 2024. Regal: refactoring programs to discover generalizable abstractions. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, and 150 others. 2025. Kimi k2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. 2025. Toucan: Synthesizing 1.5m tool-agentic data from real-world mcp environments. *Preprint*, arXiv:2510.01179.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: deliberate problem solving with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Qizheng Zhang, Michael Wornow, and Kunle Olukotun. 2025. Agentic plan caching: Test-time memory for fast and cost-efficient LLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

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

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*.

A Complete MDP Formalization

A.1 State Space

At each timestep t, the agent's state $s_t \in \mathcal{S}$ is defined as:

$$s_t = (\mathcal{I}, \mathcal{D}_{\text{API}}, q, \mathcal{H}_t) \tag{1}$$

System Instructions. The system instructions $\mathcal{I}$ specify the agent's role, output format requirements, and behavioral guidelines. This component is globally fixed across all queries and domains.

API Documentation. The API documentation $\mathcal{D}_{\text{API}} = \{(a_i, \text{doc}_i, \Theta_i)\}_{i=1}^{|\mathcal{A}|}$ provides structured information about available APIs, where a_i is the API identifier, doc_i is a natural language description, and Θ_i is the parameter space. This component is fixed per domain.

User Query. The user query $q \in \mathcal{Q}$ is a natural language input specifying the task to be accomplished. This component remains fixed throughout an episode.

Trajectory History. The trajectory history $\mathcal{H}_t = \langle(\alpha_0, o_0), \ldots, (\alpha_{t-1}, o_{t-1})\rangle$ records all previous action-observation pairs, where α_k denotes the action at timestep k and o_k denotes the corresponding observation. At the initial timestep, the history is empty: $\mathcal{H}_0 = \langle\rangle$.

A.2 Action Space

At each timestep, the agent generates an action $\alpha_t \in \mathcal{A}$, where $\alpha_t = (a_t, \theta_t)$ consists of an API identifier $a_t \in \{a_1, \ldots, a_{|\mathcal{A}|}\}$ and parameters $\theta_t \in \Theta_{a_t}$. The parameter space Θ_i for each API a_i may contain discrete, continuous, or structured parameters.

A.3 Objective Functions

Deployment Objective. The agent follows a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maps states to actions. The deployment objective is to maximize expected task completion over the query distribution $\mathcal{Q}$:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{q\sim\mathcal{Q}} \mathbb{E}_{\tau\sim p(\cdot|q,\pi)}[r(\tau)] \tag{2}$$

where τ denotes a complete trajectory and $r(\tau)$ is a binary reward based on task completion.

Evaluation Objective. We evaluate policies using per-step action accuracy on a dataset of optimal trajectories $\mathcal{D}$:

$$\text{Accuracy}(\pi) = \mathbb{E}_{(s,\alpha^*)\sim\mathcal{D}}[\mathbb{1}[\pi(s) = \alpha^*]] \tag{3}$$

Two actions are considered equal if both components match: $(a, \theta) = (a', \theta') \iff (a = a') \wedge (\theta = \theta')$.

B Baseline Policy Formulations

Contemporary Large Reasoning Model (LRM) policies implement π through in-context learning, varying in how they augment the agent's state with guidance information.

B.1 Policy Formulations

Zero-Shot Policy. The zero-shot policy operates on the standard state representation:

$$s_t = (\mathcal{I}, \mathcal{D}_{\text{API}}, q, \mathcal{H}_t) \tag{4}$$

relying entirely on the model's pre-trained knowledge and reasoning capabilities.

Few-Shot Policy. The few-shot policy augments the state with demonstration examples:

$$s_t = (\mathcal{I}, \mathcal{D}_{\text{API}}, \mathcal{E}, q, \mathcal{H}_t) \tag{5}$$

where the example set $\mathcal{E} = \{(q^{(i)}, \mathcal{H}^{(i)})\}_{i=1}^{k}$ contains k query-trajectory pairs demonstrating successful interactions. This approach requires inductive reasoning to generalize patterns from specific examples.

Chain-of-Thought Policy. Chain-of-Thought (CoT) augments the system instructions $\mathcal{I}$ with explicit reasoning directives, encouraging step-by-step problem decomposition at the cost of increased output tokens.

Rule-Augmented Policy (GEAR). **GEAR** augments the state with explicit behavioral rules:

$$s_t = (\mathcal{I}, \mathcal{D}_{\text{API}}, \mathcal{R}, q, \mathcal{H}_t) \tag{6}$$

where the rule set $\mathcal{R} = \{r_1, \ldots, r_{|\mathcal{R}|}\}$ contains domain-specific constraints extracted through meta-reasoning. Each rule $r_i = (\text{Directive}_i, \text{Pedagogical}_i)$ comprises: (1) a directive specifying the condition and required behavior (WHEN-THEN structure), and (2) a pedagogical component illustrating correct behavior, incorrect behavior, and test criteria for verification. Unlike few-shot learning which requires inductive pattern inference, both rule components enable deductive constraint application.

Combined Policy (GEAR+Few-Shot). **GEAR** can be combined with few-shot examples:

$$s_t = (\mathcal{I}, \mathcal{D}_{\text{API}}, \mathcal{R}, \mathcal{E}, q, \mathcal{H}_t) \tag{7}$$

This formulation leverages both deductive reasoning (via rules $\mathcal{R}$) and inductive reasoning (via examples $\mathcal{E}$), enabling complementary guidance mechanisms.

C Meta-Prompt Materials

This appendix provides the prompts used in **GEAR**'s meta-prompt creation stage (Section 4.1).

C.1 Base Meta-Prompt

The base meta-prompt $\mathcal{P}_{\text{meta-base}}$ (Figure 3) serves as the starting point for iterative refinement. This initial version produces statement-style rules that lack structured WHEN-THEN format.

C.2 Stage 1: Rubric Generation and Critique Prompt

In Stage 1, an LRM generates an evaluation rubric $\mathcal{C}$ and critiques $\mathcal{P}_{\text{meta-base}}$ against this rubric. The critique prompt (Figures 4, 5) instructs the LRM to create comprehensive evaluation criteria and perform detailed analysis.

C.3 Stage 2: Self-Critique Prompt

In Stage 2, the same LRM iteratively refines the prompt using self-critique. At each iteration k, the LRM generates an improved prompt $\mathcal{P}_k$ based on previous critique c_{k-1}, then evaluates $\mathcal{P}_k$ against the fixed rubric $\mathcal{C}$.

Refinement Prompt. The refinement prompt (Figures 6, 7) takes the previous prompt $\mathcal{P}_{k-1}$ and critique c_{k-1} as input:

Self-Critique Prompt. After generating $\mathcal{P}_k$, the LRM evaluates it against rubric $\mathcal{C}$. The self-critique prompt (Figures 8, 9) explicitly instructs the LRM to be "harsh" and avoid "grade inflation," as we observed LRMs tend toward overconfidence in self-assessment. The prompt requires specific evidence from the generated prompt for each criterion and a detailed explanation of any grade below A+.

C.4 Optimized Meta-Prompt

The optimized meta-prompt $\mathcal{P}^*_{\text{meta}}$ (Figure 10) is the final output after self-critique refinement and human validation. This prompt produces structured rules with explicit WHEN-THEN directives and pedagogical components.

The meta-prompt comprises eight structured sections that guide the LRM through: (1) role and task definition, (2) task context and baseline metrics, (3) data structure explanation, (4) systematic analysis steps including failure categorization, pattern identification, and root cause analysis, (5) rule extraction with structured output format, (6) validation criteria, (7) meta-analysis of coverage and expected improvement, and (8) final output format specification.

C.4.1 Key Design Principles

The meta-prompt incorporates several design principles that emerged from the self-critique process described in Section 4.1. While the complete prompt will be released upon publication for full reproducibility, we describe its key structural components and design rationale here.

Structured Multi-Step Analysis Framework. The prompt enforces a systematic eight-step pro-

cess that prevents premature rule extraction. The LRM must first complete comprehensive failure analysis (Steps 1-4) before synthesizing rules (Steps 5-6), followed by validation and meta-analysis (Steps 7-8). Each step produces structured output that serves as input to subsequent steps, ensuring thorough coverage. For instance, Step 1 requires frequency counts for each failure type (e.g., "Wrong function called: X examples"), which Step 4 then analyzes for root causes.

Explicit Categorization Schema. The prompt defines three primary failure categories: action selection errors, parameter population errors, and API error handling failures, with multiple subcategories under each. This taxonomy guides the LRM to systematically organize failures rather than analyzing them in an ad-hoc manner. For example, parameter errors are subdivided into format mismatches, missing values, wrong data types, and documentation misinterpretation, each requiring different rule types.

Frequency-Based Filtering. The prompt explicitly instructs the LRM to extract rules only for patterns occurring in $\geq \tau$ examples, preventing overfitting to rare edge cases. The LRM must track occurrence counts during analysis and justify each rule with its frequency. This threshold-based approach balances coverage (addressing common issues) with generalization (avoiding instance-specific rules).

Contrastive Learning Guidance. Step 5 requires analyzing successful predictions alongside failures to identify what distinguishes correct from incorrect behaviors. The prompt provides a structured template for this comparison answering what failed, what successful examples did right and Actionable mitigation strategy. This contrastive analysis prevents rules that would regress correct predictions, addressing the issue identified in Section 8.

Generalization Through Examples. The prompt includes just a couple of explicit good vs. bad rule examples to guide abstraction. These examples demonstrate the desired level of abstraction—rules should capture structural patterns rather than surface features.

Structured Output Schema. The prompt fully specifies the JSON schema (detailed in Table 3) for extracted rules, including all required fields and their formats. This structured output ensures rules are parseable, categorizable, and contain sufficient information for validation and deployment decisions.

Multi-Level Validation Framework. Step 7 provides explicit validation criteria organized as yes/no checks:

- *Coverage*: Does this rule prevent failures in $\geq \tau$ examples?
- *Regression*: Could this rule break currently correct predictions?
- *Atomicity*: Does this rule address exactly one failure pattern?
- *Implementability*: Can this be added to a prompt as a clear instruction?
- *Testability*: Is the test criterion measurable and unambiguous?
- *Generalizability*: Does this apply across different APIs, not just one?
- *Consistency*: Does this conflict with other rules?

The prompt requires the LRM to evaluate each extracted rule against all criteria and revise or remove rules that fail any check.

Meta-Analysis and Coverage Estimation. Step 8 requires the LRM to estimate the expected improvement from extracted rules by computing: (1) total failures in dataset, (2) failures addressed by rules (sum of estimated impacts), (3) coverage percentage, and (4) predicted performance gains. This meta-analysis serves as a self-consistency check—if coverage is low (<60%) or predicted improvement is implausibly high (>50%), the LRM must identify gaps or reconsider rule quality.

Domain-Agnostic Parameterization. The prompt uses placeholders for domain-specific information: `{api_documentation}`, `{system_instruction}`, `{baseline_metrics}`, and `{examples}`. All instructions and analysis frameworks are written generically to work across diverse API domains without modification. This design enables the single meta-prompt to be reused across all 78 evaluation domains.

D Extract Phase Details

This section provides details on the Extract phase (Section 4.3), including the complete JSON schema Table 3 for extracted rules, an extracted rule in JSON (Figure 11) and then serialized (Figure 12) format for usage in next phase, and representative examples from BMR/TMR (Table 4) and finance (Table 5) domains.

Field	Type	Required	Description
`rule_id`	integer	Yes	Unique identifier for the rule (1, 2, 3, ...)
`name`	string	Yes	Short descriptive name (e.g., "HTML Error Detection")
`priority`	string	Yes	Priority level: "High", "Medium", or "Low". Determines deployment order.
`category`	string	Yes	Failure category: "Action Selection", "Parameter Population", or "Error Handling"
`addresses`	string	Yes	Specific failure type and frequency (e.g., "Repeated failed API calls (12 examples)")
`condition`	string	Yes	WHEN clause: boolean predicate specifying when rule applies (e.g., "When API response starts with '<!DOCTYPE'")
`action`	string	Yes	THEN clause: prescribed behavior (e.g., "Recognize service unavailability, try alternative once, then call Finish")
`example_wrong`	string	Yes	Concrete example of incorrect behavior from data (e.g., "Retried bmr_index 3 times after HTML error")
`example_correct`	string	Yes	Concrete example of correct behavior (e.g., "After HTML error, tried tmr_index once, then called Finish")
`test_criterion`	string	Yes	Measurable verification method (e.g., "Check if model stops after HTML error and tries alternative")
`estimated_impact`	string	Yes	Expected improvement as percentage (e.g., "12% of action failures")

Table 3: Complete JSON schema for extracted rules. Each rule object must contain all required fields.

Rule ID	Category	Summary
1	Error Handling	**HTML Error Detection**: When response contains HTML tags, recognize as service unavailability. Do not retry same function. Try alternative once or call Finish.
2	Parameter Population	**Boolean String Format**: When parameter name suggests boolean (e.g., "inimperial"), check API docs for exact format. Use string "true"/"false", not synonyms like "metric"/"imperial".
3	Action Selection	**Direct Calculation**: When user provides all required parameters (age, weight, height, sex), call calculation API directly. Do not call helper APIs to retrieve values.

Table 4: Example rules extracted from BMR/TMR domain.

E Data Filtering

The original ToolPreference dataset contains approximately 33,000 single-turn examples constructed by tracing successful paths in depth-first search decision trees. Specifically, we use `dpo_preferencepairs_train.json`, which contains user queries, action histories, and preferred/dispreferred next-action tuples. This construction process resulted in preference pairs where some "preferred" actions were suboptimal recovery steps rather than truly optimal next actions.

To address this noise, we employed LLM-as-judge filtering using Claude-4.5-Sonnet to verify whether either action in each preference tuple represents the optimal next action given the current state. Specifically, the judge evaluates each tuple and classifies it as: (1) preferred action is optimal, (2) dispreferred action is optimal, or (3) neither action is optimal. Tuples classified as (3) were discarded, while tuples classified as (2) had their preference labels swapped. This filtering was critical for extracting meaningful behavioral rules rather than surface-level patterns from noisy supervision.

After filtering, the dataset was reduced from approximately 33,000 to 1,949 high-quality examples suitable for rule extraction. The complete filtering prompt is shown in Figure 13.

F Dataset Statistics

From the filtered dataset, we selected domains containing $\geq$20 valid examples to ensure sufficient data for both rule extraction and evaluation. This threshold yielded 78 domains with example counts

Rule ID	Category	Summary
1	Error Handling	**Empty Response Handling**: When API returns empty list/dict, check if this is valid result (no data found) vs error. If unclear, call alternative data source or Finish with explanation.
2	Parameter Population	**Stock Symbol Format**: Extract exact ticker symbols from user query (e.g., "AAPL", "GOOGL"). Do not convert to full company names. Preserve uppercase format.
3	Action Selection	**Time Range Selection**: When query mentions time period (e.g., "last 2 weeks"), use time-range API accepting start/end dates rather than calling daily APIs repeatedly.

Table 5: Example rules extracted from Finance domain.

	$\mathcal{D}_{\text{extract}}$		$\mathcal{D}_{\text{test}}$	
$\lvert\mathcal{A}\rvert$	**#Domains**	**#Examples**	**#Domains**	**#Examples**
3	17	303	17	120
4	14	248	14	100
5	13	244	13	97
6	6	106	6	42
7	10	181	10	70
8	6	98	6	37
9	2	46	2	18
10	1	18	1	7
11	9	153	9	61
Total	**78**	**1,397**	**78**	**552**

Table 6: Dataset composition by action space size. $|\mathcal{A}|$ denotes the number of available APIs in $\mathcal{D}_{\text{API}}$ for each domain.

ranging from 20–45 per domain. Domains span diverse categories including health (BMR/TMR calculation), finance (stock data), search, and utilities. We split the 78 domains 70:30 into $\mathcal{D}_{\text{extract}}$ (n=1,397) for rule extraction and $\mathcal{D}_{\text{test}}$ (n=552) for evaluation. The per-domain distribution of examples across both subsets is visualized in Figure 15. Additionally, Table 6 presents the dataset distribution grouped by action space size $|\mathcal{A}|$, where each row aggregates domains sharing the same number of available APIs in their documentation $\mathcal{D}_{\text{API}}$.

G Model Specifications

Table 7 provides specifications for all models used in our experiments. We evaluate three proprietary models (Claude-4.5-Sonnet (Anthropic, 2025), GPT-5.1 (OpenAI, 2025), Gemini-2.5-Flash (Google, 2025)) and three open-source instruct-tuned models (Qwen3-30B (Yang et al., 2025), Qwen3-4B (Yang et al., 2025), Ministral3-14B (Mistral, 2025)).

Configuration Modes. All models are evaluated in *standard mode* with greedy decoding (temperature T=0). Proprietary models are additionally evaluated in *thinking mode* with temperature T=1 (as recommended by providers) and extended reasoning budgets. Open-source models are evaluated in standard mode only due to the latency overhead of thinking mode.

Model-Specific Notes. GPT-5.1 requires an additional system prompt to ensure proper JSON-formatted outputs:

```
"You must respond with ONLY valid JSON.
Return an object with keys: thought,
action, action_input. action_input must
be a flat JSON object."
```

This was not required for other models, which reliably followed the output format from the main user instructions.

Local Deployment. Open-source Qwen3 models and Ministral3-14B were served locally using vLLM (Kwon et al., 2023) with the following commands:

- **Qwen3-30B:**[2] `vllm serve Qwen3-30B-A3B-Instruct-2507 -dtype auto -tensor-parallel-size 4`
- **Qwen3-4B:**[3] `vllm serve Qwen3-4B-Instruct-2507 -dtype auto -max_model_len 10000 -tensor-parallel-size 1`
- **Ministral3-14B:**[4] `vllm serve Ministral-3-14B-Instruct-2512 -dtype auto -max_model_len 10000 -tensor-parallel-size 2`

[2] https://huggingface.co/Qwen/Qwen3-30B-A3B-Instruct-2507

[3] https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507

[4] https://huggingface.co/mistralai/Ministral-3-14B-Instruct-2512

Hardware. Local inference was conducted on a server with 8× NVIDIA A100-SXM4-40GB GPUs (320GB total VRAM), CUDA 12.8, and driver version 570.124.06. Qwen3-30B utilized 4 GPUs via tensor parallelism, while Qwen3-4B ran on a single GPU.

H Baseline Configurations

This section details the prompt configurations for all baselines and **GEAR** variants evaluated in our experiments.

Zero-shot. Uses the standard ReAct prompt containing system instructions $\mathcal{I}$, API documentation $\mathcal{D}_{\text{API}}$, user query q, and trajectory history $\mathcal{H}_t$. No additional instructions or examples are provided. See Figure 16.

Zero-shot-CoT. Extends the zero-shot prompt by adding an explicit reasoning instruction to the task section: *"Think step by step, examine previous steps and thoughts, then decide."* See Figure 17.

Few-shot (1/2/3-shot). Includes randomly sampled examples from $\mathcal{D}_{\text{extract}}$, where each example contains the full user query, trajectory history, and action specification. The same prompt template is used for all few-shot variants; only the number of examples varies. See Figure 18.

GEAR. Incorporates extracted rules as guidelines positioned before the user query and history, as described in Section 3.2 in zero-shot prompt. See Figure 19.

GEAR + 2-shot. Combines rules with two in-context examples. Examples are placed *between* the rules and the user query. See Figure 20.

2-shot + GEAR. An alternative ordering where two in-context examples are placed *before* the guidelines/rules section.

I Extended Results

Tables 8, 9, and 10 present complete results across all model configurations. We highlight key findings below. All results would be discussed in terms of E2E Pass@1.

Zero-shot-CoT Ineffectiveness Across all models, Zero-shot-CoT provides negligible or negative gains. Claude Standard shows −6.4%, Gemini Standard −6.5%, and GPT-5.1 Standard −21%. Decrease is observed for thinking mode as well. This confirms that generic reasoning prompts offer limited value for structured tool-calling compared to explicit formatting guidance specifically for proprietary LRMs. We observed a slight performance gain for instruction-tuned open-source models, suggesting that chain-of-thought remains relevant for these models, but not for proprietary models that are RL-tuned for reasoning through test-time compute scaling.

Consistent GEAR Effectiveness Across Models **GEAR** or its variants match or exceed the strongest few-shot baselines across nearly all configurations tested. On Claude, Qwen-30B, and Qwen-4B in standard mode, **GEAR**+2-shot substantially outperforms 3-shot (by 9.72%, 24.92%, and 27.39% respectively). Ministral achieves best results with **GEAR** alone, exceeding 3-shot by 10.7%. GPT-5.1 shows mode-dependent behavior: Standard mode is neck-to-neck (**GEAR**+2-shot 49.04% vs 3-shot 49.72%), while Thinking mode strongly favors 3-shot (**GEAR**+2-shot 51.84 % vs 3-shot 60.16%). The sole exception is Gemini-2.5-Flash, where 3-shot outperforms **GEAR**+2-shot by 10% (Standard) and 13.16% (Thinking). We attribute this to Gemini's exceptionally weak zero-shot baseline (17.57%), which is 43% lower than Claude's (30.67%). Despite this, rule extraction yields comparable rule counts to other models (see Table 11), suggesting the extracted rules may be less precisely calibrated to Gemini's failure modes. Additionally, Gemini may have limited instruction-following capacity where excessive explicit constraints interfere with its internal reasoning, whereas few-shot examples provide more digestible guidance.

Ministral3-14B: Reversed Complementarity Ministral uniquely achieves best E2E with **GEAR** alone (42.53%), outperforming both **GEAR**+2-shot (38.49%) and 3-shot (38.42%). The ordering effect is also reversed: 2-shot+**GEAR**outperforms **GEAR**+2-shot by 7.43%. We hypothesize that this smaller model experiences interference when processing both rules and examples simultaneously, with rules alone providing cleaner guidance.

Ordering Effect The "constrain-then-demonstrate" ordering (**GEAR**+2-shot) outperforms "demonstrate-then-constrain" (2-shot+**GEAR**) on Claude (2.24%), GPT 5.1 (7.4%), Qwen-30B (3.28%), and Qwen-4B

Model	Version / Endpoint	Access	Standard Mode			Thinking Mode		
			Temp	Max Tokens	Reasoning	Temp	Max Tokens	Reasoning
Claude-4.5-Sonnet	`claude-sonnet-4-5-20250929-v1:0`	Amazon Bedrock	0	500	`budget_tokens=0`	1	3000	`budget_tokens=1500`
GPT-5.1	`gpt-5.1-2025-11-13`	OpenAI API	0	500	`reasoning_effort="none"`	1	3000	`reasoning_effort="high"`
Gemini-2.5-Flash	`gemini-2.5-flash`	Google AI API	0	500	`thinking_budget=-1`	1	3000	`thinking_budget=1500`
Qwen3-30B	`Qwen3-30B-A3B-Instruct-2507`	Local (vLLM)	0	500	N/A			—
Qwen3-4B	`Qwen3-4B-Instruct-2507`	Local (vLLM)	0	500	N/A			—
Ministral3-4B	`Ministral-3-14B-Instruct-2512`	Local (vLLM)	0	500	N/A			—

Table 7: Model specifications and configurations. Standard mode uses greedy decoding (T=0) for deterministic outputs. Thinking mode enables extended reasoning with higher temperature (T=1) as recommended by providers. Open-source models (Qwen3) are evaluated in standard mode only. GPT-5.1 additionally uses `verbosity="low"` in both modes to minimize latency. All models accessed via official APIs except Qwen3 models and Ministral3-14B, which were hosted locally using vLLM (Kwon et al., 2023).

(21.14%), with amplified effects on smaller models, in standard mode. Thinking mode showed similar results. Exceptions include Ministral (reversed by 7.4%) and Gemini (negligible 0.6% difference), indicating model-specific sensitivity to prompt structure.

Small Model Competitiveness Despite reduced capacity, Qwen3-4B with **GEAR**+2-shot achieves 57.52% E2E, substantially exceeding Claude's 3-shot (47.02%). Similarly, Ministral3-14B with **GEAR** lone (42.53%) approaches Claude's 2-shot performance (45.35%) despite a much weaker zero-shot baseline (24.42% vs 30.67%), demonstrating that explicit guidance effectively compensates for limited model capacity.

Efficiency and Generation Confidence **GEAR**+2-shot consistently achieves lower latency than **GEAR** alone despite higher input token counts across most models: Claude (3.94s vs 5.08s), Qwen-30B (0.78s vs 0.85s), and Gemini (1.80s vs 1.86s). This counterintuitive result supports our hypothesis that combined rule-example guidance increases generation confidence, reducing the model's search space during decoding. This is evidenced by consistently fewer output tokens for **GEAR**+2-shot compared to **GEAR** alone (145 vs 179 on Claude, 101 vs 114 on Qwen-30B, 94 vs 98 on Gemini). The efficiency gains are particularly notable given the substantial accuracy improvements, suggesting **GEAR** variants offer superior performance-to-latency ratios compared to both zero-shot baselines and few-shot alternatives.

Future Directions The model-specific variations observed in **GEAR**'s effectiveness, particularly the reversed complementarity in Ministral and reduced gains on Gemini, raise intriguing questions about the internal reasoning mechanisms underlying tool-calling in LLMs. Understanding why certain models benefit more from deductive rules versus inductive examples, or why prompt ordering effects vary across architectures, could yield deeper insights into how these models process structured generation tasks. However, our API-only experimental setup, without access to model gradients or output logits, similar to constraints in automated prompt optimization research, limits the mechanistic analyses possible in this work. We leave such interpretability investigations, including attention pattern analysis and probing studies on open-weight models, for future research.

J Extraction Mode Analysis

We compare standard-mode extraction versus thinking-mode extraction across all evaluated models. Table 11 presents average E2E accuracy and win rates for each extractor-evaluator combination. Claude-4.5-Sonnet was the underlying LRM for extraction. The hyperparameters for standard-mode are: max_tokens = 32000, temperature = 0. The hyperparameters for thinking-mode are: max_tokens = 32000, temperature = 1, budget_tokens = 16000.

Key Findings. Thinking-mode extraction produces superior rules across nearly all configurations:

- Thinking extractor achieves 12.6% relative improvement in average E2E (41.64% vs 36.99%)
- Thinking extractor wins in 63.8% of domain-model combinations (444/696)
- Benefits are consistent across all model families, with largest gains on Gemini-2.5-Flash Standard (29.3% relative), GPT-5.1 Standard (22.8% relative), and Qwen3-4B (17.1% relative)
- Gains extend to newly evaluated models: Ministral3-14B shows 12.0% relative improvement with thinking-extracted rules

- Thinking- and standard-extracted rules are similar in count (9.1 vs 9.3 average) and length (749 vs 735 average characters), indicating the performance gap stems from rule quality rather than quantity

These results suggest that extended reasoning during extraction enables more thorough failure analysis, producing higher-quality rules that transfer across inference-time configurations. Investigating what specific characteristics constitute higher-quality rules remains an avenue for future research.

K Complexity Analysis

We analyze how performance varies with task complexity along two dimensions: trajectory complexity (number of hops) and action space complexity (number of available APIs).

K.1 Trajectory Complexity (Hops)

We operationalize trajectory complexity as the number of previous API calls (hops) in the trajectory history $\mathcal{H}_t$. Higher hop counts indicate more complex multi-step reasoning tasks. Figure 21 presents results on Claude-4.5-Sonnet.

Key Findings.

- All methods exhibit performance degradation as complexity increases, with E2E Pass@1 declining from 39-71% at hop 0 to 12-38% at hop 5
- **GEAR**+2-shot maintains highest performance across all complexity levels (70.71% at hop 0, 37.5% at hop 5)
- Relative ordering of methods remains consistent regardless of complexity, suggesting complementarity between rules and examples is robust to task difficulty
- Parameter prediction degrades more severely than action selection: zero-shot Parameter Pass@1 drops from 45.45% to 12.5%, while Action Pass@1 remains stable (74.75% to 75.0%)
- Latency and cost increase approximately linearly with hop count due to longer context windows
- Critical-**GEAR**offers favorable efficiency, maintaining competitive accuracy with lower cost than **GEAR** and **GEAR**+2-shot across all complexity levels

K.2 Action Space Complexity (Number of APIs)

We operationalize action space complexity as the number of available APIs per domain, ranging from 3 to 11 in our dataset. Larger action spaces present more alternatives for the model to choose from, potentially increasing decision difficulty. Figure 22 presents results on Claude-4.5-Sonnet.

Key Findings.

- Unlike trajectory complexity, performance does not exhibit a clear monotonic degradation pattern with increasing action space size
- Performance varies considerably across API counts but appears more influenced by domain-specific characteristics than action space size alone
- **GEAR**+2-shot maintains consistently strong performance across action space sizes, achieving highest E2E Pass@1 at most configurations (e.g., 61.86% at 5 APIs, 49.18% at 11 APIs)
- Latency remains relatively stable across action space sizes, with occasional outliers due to domain-specific factors
- Cost increases moderately with API count due to longer API documentation in the prompt
- Critical-**GEAR**achieves competitive performance with substantially lower cost across all action space sizes

These results suggest that action selection difficulty is less sensitive to the number of available alternatives than to trajectory length, likely because API documentation provides sufficient discriminative information regardless of action space size.

Base Meta-Prompt $\mathcal{P}_{\text{meta-base}}$

```
You are an expert prompt engineer and a brilliant systems analyst. I have run a baseline
experiment on tool-use tasks using a powerful LLM.

The results show a clear reasoning gap:
- **Action Pass@1 (Choosing the right tool)**: {action_pass_rate}%
  ({action_pass_count}/{total})
- **Parameter Pass@1 (Populating the tool's input)**: {param_pass_rate}%
  ({param_pass_count}/{total})
- **E2E Pass@1 (Both correct)**: {e2e_pass_rate}% ({e2e_pass_count}/{total})

The "Parameter" failures are *semantic*—the model picks the wrong *values* for the parameters,
even when the syntax is correct.

The model is failing at two levels:
1. **Action Reasoning:** It struggles to select the correct tool from the API list.
2. **Parameter Reasoning:** It struggles to populate parameters with correct values based on
  the user's query and conversation history.

Your task is to analyze {num_samples} examples and generate 5-7 general-purpose, high-impact
"Heuristic Reasoning Rules" that will help the model improve *both* action selection and
parameter population.

---
**GUIDELINES FOR RULES:**

* **Target Both Reasoning Failures:** Generate rules that guide *Action Selection* and rules
  that guide *Parameter Population*.
* **Go Deep, Not Surface-Level:** The rules must be generalizable. Do not create rules that
  only apply to a single example or task.
    * **BAD RULE (Surface-Level):** "If the user asks about 'NFLX' and 'AMZN', use the
      `ranges_2w_for_finance...` tool."
    * **GOOD RULE (Structural Action Rule):** "Before acting, check the conversation
      history. If a previous action failed, do not repeat the exact same action. Instead,
      select a different tool or modify the parameters."
    * **GOOD RULE (Structural Parameter Rule):** "When an API has multiple parameters, scan
      the user's query and history for specific values for each one. Do not use example
      values from the documentation."
---

Here are the {num_samples} examples:

{examples}

---

Based on your analysis of all cases, what are the 5-7 general-purpose reasoning rules that will
improve both the `Action Pass@1` and `Parameter Pass@1` scores?

**Distilled Heuristic Rules:**
Provide exactly 5-7 rules in the following format:
1. [Rule description]
2. [Rule description]
...
```

Figure 3: Base meta-prompt before refinement. This prompt lacked systematic analysis steps and explicit validation criteria, leading to less effective rule extraction.

Stage 1: Rubric Generation and Critique Prompt

```
You are an expert prompt engineer and AI systems analyst. Your task is to perform a brutal,
honest critique of a prompt designed to extract heuristic rules to improve model performance.

### TASK
The base prompt is designed to analyze evaluation results and extract heuristic rules to
improve LLM performance on the following task:

{task_instructions}

### BASELINE METRICS
Current performance on this task:
- **Action Pass@1**: {action_pass_rate}% - Model selects the correct tool
- **Parameter Pass@1**: {param_pass_rate}% - Model populates correct parameter values
- **E2E Pass@1**: {e2e_pass_rate}% - Both action and parameters correct
- **Total Examples**: {total}

### PROBLEM
Rules extracted by the base prompt either don't improve these metrics or actively decrease
performance.

### BASE PROMPT TO CRITIQUE
{base_prompt}

The base prompt must work across different API domains (e.g., BMR/TMR, Finance, Search).
Example API:

{api_docs}

### EXAMPLE FAILURE CASE
Below is one failure example to help you assess if the base prompt's analysis approach would
work:

{failure_example}

### YOUR TASK
1. Create a critique rubric that evaluates:
   **STRUCTURAL REQUIREMENTS (Must-Have):**
   a) Role & Task Definition - Does it clearly state the role? Does it require analyzing BOTH
successes and failures? Does it explain the task context (what model was doing, with what API)
BEFORE presenting baseline metrics? Are actionable heuristics explicitly required?
   b) Problem Statement - Does it provide clear placeholders/sections for: (i) the system
instruction given to the model, (ii) the API documentation the model used, (iii) baseline
zero-shot performance metrics? Are these properly labeled and positioned after task context?
   c) Optimization Goals - Does it specify: (i) improve performance metrics, (ii) reduce token
generation for lower latency?
   d) Data Structure - Does it explain the example format (input, preferred_output,
non_preferred_output)?
   e) Example Walkthrough - Does it demonstrate analysis on one example following its analysis
steps?
   f) Analysis Steps - Are the detailed analysis steps clear, systematic, and effective?
   g) Examples Placeholder - Is there a clear section to provide all examples?
   h) Analysis Execution - Does it guide how to conduct the analysis across all examples?
   i) Output Format - Does it specify a structured output format (e.g., JSON)? Is the schema
clearly defined with required fields? Is it parseable by standard tools? Does it include all
necessary information for extracting actionable heuristics (condition, action, examples, test
criteria)?

   **ANALYSIS METHODOLOGY (Your Focus):**
   Design 5-8 additional criteria specifically evaluating the analysis steps (item f above).
   Consider: Does it analyze successes vs only failures? Root cause identification?
Generalizability? Actionability?
```

Figure 4: Stage 1: Rubric Generation and Critique Prompt. This prompt guides the LRM to generate a comprehensive rubric and critique the base meta-prompt.

Stage 1: Rubric Generation and Critique Prompt (Cont.)

```
2. Analyze the base prompt against your complete rubric
3. For EACH rubric criterion (a-i and any additional), provide:
   - Detailed critique with specific examples
   - Letter grade (A+, A, A-, B+, B, B-, C+, C, C-, D+, D, D-, F)
   - Reason for the grade
4. Provide an overall letter grade based on all criteria

### OUTPUT FORMAT
You MUST respond with ONLY a valid JSON object with exactly these keys:

```json
{
  "rubric": "Your critique rubric with all criteria",
  "critiques": {
    "criterion_1_key": {"grade": "[letter grade]", "reason": "[why this grade]", "details":
"[specific examples and analysis]"},
    "criterion_2_key": {"grade": "[letter grade]", "reason": "[why this grade]", "details":
"[specific examples and analysis]"},
    "criterion_N_key": {"grade": "[letter grade]", "reason": "[why this grade]", "details":
"[specific examples and analysis]"}
  },
  "overall_grade": "[letter grade]",
  "overall_summary": "Brief summary of strengths and critical weaknesses"
}
```

Note: Use descriptive keys for each criterion (e.g., 'role_task_definition',
'analysis_methodology', etc.). Include all criteria from the structural requirements (a-i) plus
any additional criteria you design.

Be brutally honest. If the prompt is fundamentally flawed, say so explicitly.
```

Figure 5: Stage 1: Rubric Generation and Critique Prompt (Cont.). This prompt guides the LRM to generate a comprehensive rubric and critique the base meta-prompt.

Stage 2: Refinement Prompt

```
You are an expert prompt engineer. Your task is to generate an improved prompt for extracting
effective heuristic reasoning rules from model failures.

### CONTEXT
The goal is to create a prompt that:
- Analyzes LLM failures on tool-use tasks
- Extracts actionable, general-purpose heuristics
- Generates rules that ACTUALLY improve performance when appended to task prompts

### CRITIQUE OF PREVIOUS PROMPT
{critique_analysis}

### CRITIQUE RUBRIC
{critique_rubric}

### EXAMPLE TASK CONTEXT
Below is ONE example of the task domain. Your prompt must work across different APIs/tools.

Example Task Instructions: {task_instructions}

Example API Documentation: {api_docs}
```

Figure 6: Stage 2: Refinement prompt. This guides the LRM to generate improved prompts addressing identified weaknesses.

Stage 2: Refinement Prompt (Cont.)

```
### TASK DOMAIN
This prompt will be used to extract heuristics from LLM failures on tool-use tasks.
The task involves (across DIFFERENT APIs/tools):
- Selecting the correct API/tool from documentation
- Populating parameters with correct values from user queries
- Handling conversation history and context

NOTE: The example below is just ONE instance. Your prompt must generalize to other APIs/tools.

### YOUR TASK
Generate an improved prompt that addresses ALL the flaws identified in the critique.

### REQUIREMENTS
1. The prompt must guide the LLM to extract heuristics that:
   - Are GENERAL-PURPOSE and transferable across ANY API/tool domain
   - Anticipate common failure modes (action selection, parameter population, context
     handling)
   - Are actionable and specific enough to change behavior
   - Avoid overfitting to the specific example API shown above

2. Output format:
   - No special tokens (no <thinking>, <output>, etc.)
   - Use simple delimiters (---, ###)
   - Rules should be numbered and easily extractable

3. The prompt should:
   - Provide clear examples of good vs bad rules
   - Guide deep analysis of failure patterns
   - Encourage structural/systematic thinking
   - Prevent overfitting to specific examples

### OUTPUT FORMAT
Provide your improved prompt as a JSON-compatible array of strings.
Each string is one line of the prompt.

--- IMPROVED PROMPT ---
[Your improved prompt here, formatted as JSON array]
```

Figure 7: Stage 2: Refinement prompt (Cont.). This guides the LRM to generate improved prompts addressing identified weaknesses.

Stage 2: Self-Critique Prompt

```
You are a harsh, uncompromising prompt evaluation expert. Your task is to critique a prompt
designed to extract heuristic reasoning rules.

### CRITIQUE RUBRIC
{critique_rubric}

### PROMPT TO EVALUATE
{generated_prompt}

### YOUR TASK
Evaluate the prompt against EACH criterion in the rubric.

For each criterion:
1. Score it (Excellent / Good / Adequate / Poor / Failing)
2. Provide specific evidence from the prompt
3. Identify concrete weaknesses
```

Figure 8: Stage 2: Self-critique prompt. This evaluates generated prompts against the fixed rubric.

Stage 2: Self-Critique Prompt (Cont.)

```
### OUTPUT FORMAT

--- CRITERION-BY-CRITERION EVALUATION ---
[For each criterion, provide assessment]

--- OVERALL ASSESSMENT ---
Strengths:
- [List 2-3 key strengths]

Critical Weaknesses:
- [List all significant flaws]

Grade: [A+, A, A-, B+, B, B-, C+, C, C-, D, or F]

Grade Justification:
[Explain why this specific grade, not higher or lower]

--- WHAT PREVENTS A+ GRADE ---
[If grade is not A+, be BRUTALLY specific about what is missing or wrong]
[List concrete, actionable gaps that prevent the highest grade]

Be harsh. Grade inflation helps no one. If it's not truly excellent, don't give it an A+.
```

Figure 9: Stage 2: Self-critique prompt (Cont.). This evaluates generated prompts against the fixed rubric.

Meta-Prompt $\mathcal{P}^*_{\text{meta}}$

```
[The complete meta-prompt will be publicly released upon publication to ensure
full reproducibility. For review purposes, the key structural components and
design principles are detailed in Appendix C.4.1.]

This should include:
- Role definition: "You are an expert at analyzing LLM tool-use behavior..."
- Task context: Description of what model was doing
- Baseline metrics placeholders: {action_pass_rate}, {param_pass_rate}, etc.
- API documentation placeholder: {api_documentation}
- System instruction placeholder: {system_instruction}
- Analysis execution instructions
- Step 1: Failure frequency analysis
- Step 2: Parameter format deep analysis
- Step 3: API error pattern analysis
- Step 4: Root cause identification
- Step 5: Success pattern analysis
- Step 6: Extract rules (with JSON format specification)
- Step 7: Validate rules
- Step 8: Meta-analysis
- Examples placeholder: {examples}
- Final output format specification
```

Figure 10: Complete meta-prompt $\mathcal{P}^*_{\text{meta}}$ for rule extraction. The prompt guides systematic analysis through eight structured steps.

Example Rule in JSON Format

```
{
  "rule_id": 1,
  "name": "HTML Error Detection",
  "priority": "High",
  "category": "Error Handling",
  "addresses": "Repeated failed API calls after service errors (12 examples)",
  "condition": "When API response is a string starting with '<!DOCTYPE' or
               '<html>' or contains 'Application Error'",
  "action": "Immediately recognize this as service unavailability (NOT a
            parameter error). Try alternative function once if available.
            If no alternative or alternative fails, call Finish with
            explanation. Never retry same function after HTML error.",
  "example_wrong": "Received HTML error from bmr_index_for_bmr_and_tmr,
                   then retried bmr_index_for_bmr_and_tmr with same
                   parameters 3 times",
  "example_correct": "Received HTML error from bmr_index_for_bmr_and_tmr,
                     tried tmr_index_for_bmr_and_tmr once, then called
                     Finish with explanation",
  "test_criterion": "Check if model stops retrying after detecting HTML
                    error and either tries alternative once or calls Finish",
  "estimated_impact": "12% of action selection failures"
}
```

Figure 11: Example extracted rule in JSON format showing all required fields. This rule addresses the failure mode of repeatedly retrying APIs after service-level errors.

Example Rule in Serialized Natural Language Format

```
Rule 1:
  WHEN: When API response is a string starting with '<!DOCTYPE' or '<html>'
        or contains 'Application Error'
  THEN: Immediately recognize this as service unavailability (NOT a parameter
        error). Try alternative function once if available. If no alternative
        or alternative fails, call Finish with explanation. Never retry same
        function after HTML error.
  WRONG BEHAVIOR: Received HTML error from bmr_index_for_bmr_and_tmr, then
                  retried bmr_index_for_bmr_and_tmr with same parameters 3
                  times
  CORRECT BEHAVIOR: Received HTML error from bmr_index_for_bmr_and_tmr, tried
                    tmr_index_for_bmr_and_tmr once, then called Finish with
                    explanation
  TEST CRITERION: Check if model stops retrying after detecting HTML error and
                  either tries alternative once or calls Finish
```

Figure 12: Serialized natural language version of the rule from Figure 11. This format is inserted into inference prompts.

LLM-as-Judge Filtering Prompt

```
You are an expert evaluator for AI agent action selection tasks. Your job is to identify the
BEST action between two options given the context.

# Task
Given a user query, conversation history, available APIs, and two possible actions (Action A
and Action B), determine which action is BEST and whether it's significantly better.

# Evaluation Criteria
Compare both actions on:
1. **Correctness**: Does it match what the user requested?
2. **Parameter Quality**: Are parameters accurate, complete, and optimal?
3. **Logical Next Step**: Is it the most logical and appropriate next step given the
  conversation history?
4. **Context Awareness**: Does it consider conversation history and previous attempts?
5. **API Usage**: Does it use the most appropriate API/tool?

# Decision Rules
- Mark as VALID only if one action is CLEARLY BETTER than the other AND is the most logical
  next step
- If both actions are equally good or bad, mark as INVALID
- If both would fail, mark as INVALID
- If neither action is the most logical next step based on conversation history, mark as
  INVALID (even if one is technically correct)
- The best action must be: (1) correct, (2) superior to the alternative, AND (3) the most
  logical next step given the context

# Examples

## Example 1: VALID (Action A is best)
User requested: Get social sentiment data surrounding specific stocks on Twitter
Action A: Call API with social parameter set to 'twitter'
Action B: Call API with social parameter set to 'stocktwits'
Verdict: VALID - Action A directly matches user's explicit request for Twitter data

## Example 2: INVALID (Both reasonable)
User requested: Search for phones
Previous attempts: None
Action A: Search for 'phone'
Action B: Search for 'smartphone'
Verdict: INVALID - Both are reasonable first attempts, no clear winner

## Example 3: VALID (Action B is best)
User requested: Search for phones
Previous attempts: Searched 'phone' twice, got same results
Action A: Search for 'phone' again
Action B: Search for 'smartphone' instead
Verdict: VALID - Action B adapts to context, Action A repeats failed attempt

## Example 4: INVALID (Neither is logical next step)
User requested: Find product reviews
Previous attempts: None, just started conversation
Action A: Delete user account
Action B: Update payment method
Verdict: INVALID - Both are valid actions but neither is the logical next step for finding
reviews
```

Figure 13: Claude-4.5-Sonnet prompt used for filtering preference pairs. The prompt evaluates whether candidate actions represent optimal next steps given the state.

LLM-as-Judge Filtering Prompt (Cont.)

```
# Input Format
INSTRUCTION:
{instruction_data}

CONVERSATION HISTORY:
{input_data}

ACTION A:
{action_a}

ACTION B:
{action_b}

# Your Task
Analyze the input and respond with a JSON object:
{
  "verdict": "VALID" or "INVALID",
  "best_action": "A" or "B" or "NONE",
  "reasoning": "Brief explanation comparing both actions and why one is clearly better",
  "confidence": "high" or "medium" or "low"
}

Note: Set best_action to "NONE" if verdict is INVALID
```

Figure 14: Claude-4.5-Sonnet prompt used for filtering preference pairs (Cont.). The prompt evaluates whether candidate actions represent optimal next steps given the state.

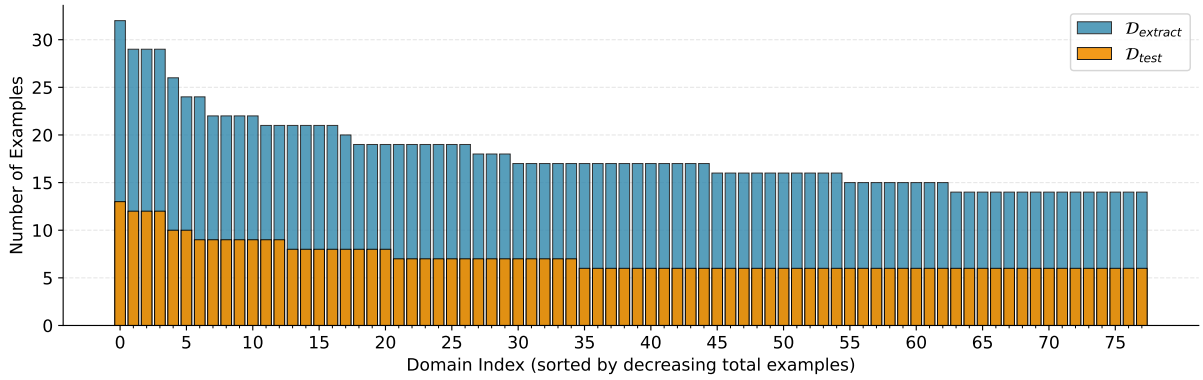

Figure 15: Distribution of examples across 78 domains in $\mathcal{D}_{\text{extract}}$ and $\mathcal{D}_{\text{test}}$. Domains are indexed in decreasing order of total examples. Each domain defines a unique API documentation $\mathcal{D}_{\text{API}}$ with corresponding action space $\mathcal{A}$.

Zero-Shot Baseline Prompt $\mathcal{P}_{\text{zero-shot}}$

```
{instruction_data}

{api_documentation}

--- CONVERSATION HISTORY ---
{input_data}

--- TASK ---
Based on the instructions, API documentation, and conversation history, what is the next
thought and action?

Assistant:
```

Figure 16: Zero-shot baseline prompt template.

Zero-Shot Baseline Prompt $\mathcal{P}_{\text{zero-shot-cot}}$

```
{instruction_data}

{api_documentation}

--- CONVERSATION HISTORY ---
{input_data}

--- TASK ---
Think step by step, examine previous steps and thoughts, then decide: Based on the instructions,
API documentation, and conversation history, what is the next thought and action?

Assistant:
```

Figure 17: Zero-shot-CoT baseline prompt structure with explicit reasoning instruction.

Few-Shot Baseline Prompt $\mathcal{P}_{\text{few-shot}}$

```
{instruction_data}

{api_documentation}

Here are some examples of how to perform the task:
{few_shot_examples}
--- END OF EXAMPLES ---

--- CONVERSATION HISTORY ---
{input_data}

--- TASK ---
Based on the instructions, API documentation, and conversation history, what is the next
thought and action?

Assistant:
```

Figure 18: Few-shot baseline prompt structure. The same template is used for 1/2/3-shot variants.

GEAR Inference Prompt Template $\mathcal{P}_{\mathbf{GEAR}}$

```
{instruction_data}

{api_documentation}

{guidelines}

--- CONVERSATION HISTORY ---
{input_data}

--- TASK ---
Based on the instructions, API documentation, and conversation history, what is the next
thought and action?

Assistant:
```

Figure 19: **GEAR** inference prompt template. Rules are inserted between API documentation and conversation history, providing explicit behavioral constraints during action selection.

GEAR+2-shot Inference Prompt Template $\mathcal{P}_{\text{GEAR+2-shot}}$

```
{instruction_data}

{api_documentation}

{guidelines}

Here are some examples of how to perform the task:
{few_shot_examples}
--- END OF EXAMPLES ---

--- CONVERSATION HISTORY ---
{input_data}

--- TASK ---
Based on the instructions, API documentation, and conversation history, what is the next
thought and action?

Assistant:
```

Figure 20: **GEAR**+2-shot inference prompt template. Examples placed between rules and query.

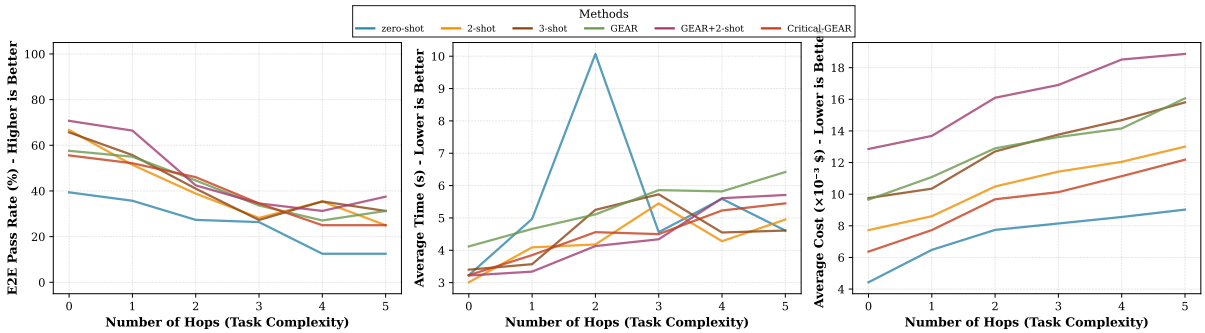

Figure 21: Performance, latency, and cost as a function of task complexity (number of hops) on Claude-4.5-Sonnet. All methods degrade with complexity, but GEAR+2-shot maintains highest performance across all levels while achieving competitive latency and cost.

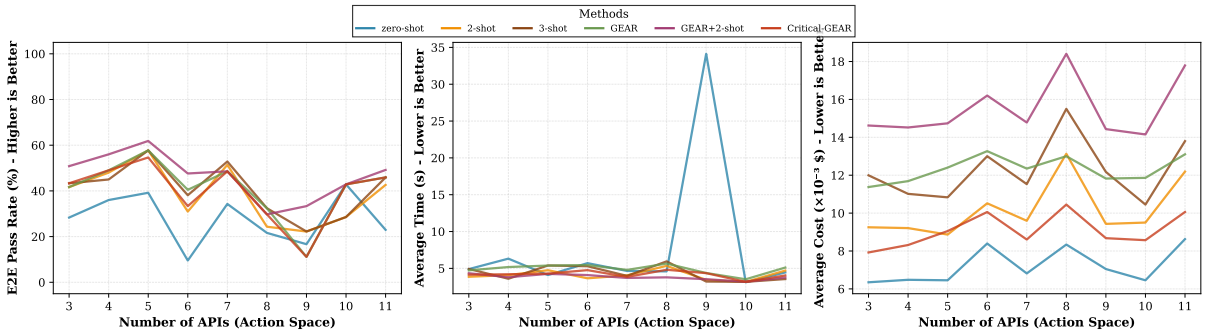

Figure 22: Performance, latency, and cost as a function of action space complexity (number of APIs) on Claude-4.5-Sonnet. Unlike trajectory complexity, performance does not show clear degradation with larger action spaces.

Method	Pass@1 (%)			Gain (%)	Efficiency			
	Action	Parameter	E2E	ΔE2E (%)	Latency (s)	ΔLat. (%)	In Tok.	Out Tok.
Claude-4.5-Sonnet *(Standard | Thinking)*								
Zero-shot	62.76 | 64.49	40.04 | 38.63	30.67 | 30.36	– | −1.0	5.40 | 7.57	– | −40.2	1,445 | 1,474	179 | 416
Zero-shot-CoT	62.52 | 64.52	38.14 | 38.23	28.70 | 30.58	−6.4 | −0.3	5.69 | 9.64	−5.4 | −78.5	1,459 | 1,488	246 | 504
1-shot	67.64 | 67.19	49.90 | 52.18	39.72 | 42.33	+29.5 | +38.0	4.05 | **7.01**	+24.9 | **−29.8**	2,041 | 2,070	136 | 356
2-shot	71.07 | 70.80	54.93 | 56.36	45.35 | 48.13	+47.9 | +56.9	4.12 | 7.04	+23.7 | −30.4	2,665 | 2,693	122 | 359
3-shot	73.30 | 73.96	55.74 | 58.38	47.02 | 50.04	+53.3 | +63.2	4.43 | 7.29	+17.9 | −35.1	3,340 | 3,368	123 | 364
GEAR (ours)	73.14 | 73.15	53.20 | 54.03	46.29 | 46.89	+50.9 | +52.9	5.08 | 8.81	+5.9 | −63.2	3,131 | 3,184	179 | 503
Critical-**GEAR** (ours)	72.41 | 65.72	52.79 | 46.51	45.52 | 36.13	+48.4 | +17.8	4.18 | 8.35	+22.6 | −54.8	2,052 | 2,107	176 | 465
GEAR+2-shot (ours)	**75.78** | **79.31**	**58.54** | **62.04**	**51.59** | **56.75**	**+68.2** | **+85.0**	3.94 | 7.74	+27.0 | −43.4	4,336 | 4,924	145 | 489
2-shot+**GEAR** (ours)	74.94 | 72.04	58.38 | 57.21	50.46 | 47.95	+64.5 | +56.3	3.91 | 8.40	+27.6 | −55.6	4,320 | 4,397	147 | 472
DirectiveOnly-**GEAR**	68.90 | 67.92	49.62 | 48.19	41.67 | 39.63	+35.9 | +29.2	4.72 | 8.61	+12.5 | −59.4	2,331 | 2,378	201 | 523
PedagogyOnly-**GEAR**	69.80 | 66.26	48.61 | 50.43	40.66 | 39.97	+32.6 | +30.3	**3.82** | 7.84	**+29.2** | −45.3	2,279 | 2,315	170 | 483
Qwen3-30B-A3B *(Standard only)*								
Zero-shot	57.53	42.72	30.20	−1.5†	**0.63**	**+88.4**†	1,299	83
Zero-shot-CoT	57.94	41.35	30.68	+0.0†	0.64	+88.1†	1,313	89
1-shot	64.64	51.36	38.84	+26.6†	0.68	+87.5†	1,823	89
2-shot	66.43	54.05	43.11	+40.6†	0.69	+87.2†	2,366	89
3-shot	70.97	58.00	47.78	+55.8†	0.69	+87.3†	2,961	90
GEAR (ours)	72.81	56.25	47.74	+55.7†	0.85	+84.2†	2,850	114
Critical-**GEAR** (ours)	65.25	49.91	38.40	+25.2†	0.72	+86.7†	1,858	101
GEAR+2-shot (ours)	**82.25**⋆	**65.24**⋆	**59.69**⋆	**+94.6**⋆†	0.78	+85.6†	4,446	101
2-shot+**GEAR** (ours)	78.68	63.85	57.79	+88.4†	0.80	+85.2†	4,469	106
DirectiveOnly-**GEAR**	65.00	51.38	40.97	+33.6†	0.83	+84.7†	2,104	116
PedagogyOnly-**GEAR**	66.35	51.40	41.61	+35.7†	0.75	+86.1†	2,070	105
Qwen3-4B *(Standard only)*								
Zero-shot	57.53	37.57	26.45	−13.8†	**0.80**	**+85.2**†	1,299	84
Zero-shot-CoT	57.57	39.28	28.07	−8.5†	0.86	+84.1†	1,313	92
1-shot	62.88	48.93	37.78	+23.2†	0.89	+83.5†	1,823	93
2-shot	63.72	51.21	39.79	+29.7†	0.94	+82.6†	2,366	97
3-shot	68.09	57.41	45.15	+47.2†	1.02	+81.2†	2,961	104
GEAR (ours)	72.51	53.55	45.48	+48.3†	1.17	+78.3†	2,854	121
Critical-**GEAR** (ours)	63.42	45.17	33.96	+10.7†	0.99	+81.6†	1,830	104
GEAR+2-shot (ours)	**79.06**	**64.10**	**57.52**	**+87.5**†	1.17	+78.3†	4,388	113
2-shot+**GEAR** (ours)	72.22	58.07	47.48	+54.8†	1.32	+75.6†	3,917	132
DirectiveOnly-**GEAR**	66.42	47.01	36.59	+19.3†	1.16	+78.5†	2,103	122
PedagogyOnly-**GEAR**	64.28	49.25	38.57	+25.8†	1.07	+80.2†	2,077	112
Ministral3-14B *(Standard only)*								
Zero-shot	53.25	36.06	24.42	−20.4†	1.44	+73.4†	1,343	147
Zero-shot-CoT	53.76	37.32	25.69	−16.2†	1.51	+72.0†	1,357	158
1-shot	58.25	43.15	32.44	+5.8†	1.41	+73.9†	1,898	143
2-shot	61.90	47.67	37.14	+21.1†	**1.40**	**+74.1**†	2,472	140
3-shot	62.93	48.28	38.42	+25.3†	1.43	+73.6†	3,101	142
GEAR (ours)	**70.19**	49.49	**42.53**	**+38.7**†	1.61	+70.2†	3,008	159
Critical-**GEAR** (ours)	60.27	44.00	33.11	+8.0†	1.53	+71.7†	1,945	153
GEAR+2-shot (ours)	67.74	46.71	38.49	+25.5†	1.67	+69.0†	4,131	162
2-shot+**GEAR** (ours)	68.91	**51.44**	41.35	+34.8†	1.67	+69.2†	4,131	161
DirectiveOnly-**GEAR**	63.22	44.27	35.26	+15.0†	1.62	+69.9†	2,191	163
PedagogyOnly-**GEAR**	63.66	44.86	35.80	+16.7†	1.49	+72.4†	2,189	148

Table 8: **GEAR** results on $\mathcal{D}_{\text{test}}$ across all model configurations. E2E Pass@1 (%) is the primary metric. For Claude-4.5-Sonnet, values are shown as Standard | Thinking. ΔE2E = relative improvement over Claude-4.5-Sonnet Zero-shot (Standard, 30.67%); ΔLat. = speedup relative to Claude-4.5-Sonnet Zero-shot (Standard, 5.40s), where positive values indicate faster inference. Best results per model (per mode for Claude) in **bold**; ⋆overall best across all configurations. †Relative to Claude-4.5-Sonnet Zero-shot (Standard). Row colors: **GEAR** variants; ablation studies.

Method	Pass@1 (%)			Gain (%)	Efficiency			
	Action	Parameter	E2E	ΔE2E (%)	Latency (s)	ΔLat. (%)	In Tok.	Out Tok.
GPT-5.1 *(Standard | Thinking)*								
Zero-shot	55.80 | 62.58	37.83 | 45.45	27.24 | 35.68	– | +31.0	2.56 | 18.47	– | −620.5	1,286 | 1,281	124 | 1,236
Zero-shot-CoT	47.53 | 59.74	30.84 | 44.36	21.51 | 33.69	−21.0 | +23.7	2.47 | 16.18	+3.7 | −531.0	1,300 | 1,296	144 | 1,364
1-shot	60.35 | 67.54	44.26 | 53.92	33.93 | 44.46	+24.6 | +63.2	2.20 | 12.36	+14.1 | −382.1	1,775 | 1,773	125 | 1,057
2-shot	67.67 | 72.76	54.16 | 60.67	46.06 | 53.35	+69.1 | +95.9	2.26 | 11.74	+12.0 | −357.8	2,285 | 2,280	122 | 979
3-shot	**68.92** | **77.77**⋆	**56.94** | **65.94**⋆	**49.72** | **60.16**⋆	**+82.5** | **+120.9**⋆	2.21 | **11.58**	+13.7 | **−351.6**	2,834 | 2,835	130 | 987
GEAR (ours)	64.16 | 70.09	52.29 | 54.20	44.67 | 46.65	+64.0 | +71.3	2.45 | 15.18	+4.4 | −492.0	2,922 | 2,783	122 | 1,285
Critical-**GEAR** (ours)	52.08 | 61.05	41.42 | 48.99	32.66 | 39.57	+19.9 | +45.3	1.98 | 13.54	+22.9 | −428.2	1,895 | 1,835	126 | 1,339
GEAR+2-shot (ours)	67.53 | 73.03	54.62 | 58.57	49.04 | 51.84	+80.0 | +90.3	2.45 | 13.39	+4.5 | −422.3	3,924 | 3,788	122 | 1,106
2-shot+**GEAR** (ours)	64.28 | 70.50	53.74 | 57.65	45.66 | 49.99	+67.6 | +83.5	**1.80** | 13.48	**+29.7** | −425.9	3,917 | 3,776	127 | 1,169
DirectiveOnly-**GEAR**	52.35 | 63.45	40.99 | 48.92	33.04 | 40.97	+21.3 | +50.4	1.97 | 20.08	+23.2 | −683.3	2,137 | 2,069	139 | 1,289
PedagogyOnly-**GEAR**	53.23 | 59.79	41.99 | 46.82	33.92 | 37.31	+24.5 | +37.0	1.82 | 16.64	+29.1 | −548.9	2,089 | 2,007	123 | 1,406

Table 9: GPT-5.1 **GEAR** results on $\mathcal{D}_{\text{test}}$. E2E Pass@1 (%) is the primary metric. Values shown as Standard | Thinking. ΔE2E = relative improvement over GPT-5.1 Zero-shot (Standard, 27.24%); ΔLat. = speedup relative to GPT-5.1 Zero-shot (Standard, 2.56s), where positive values indicate faster inference. Best results per mode in **bold**; ⋆overall best across all configurations. Row colors: **GEAR** variants; ablation studies.

Method	Pass@1 (%)			Gain (%)	Efficiency			
	Action	Parameter	E2E	ΔE2E (%)	Latency (s)	ΔLat. (%)	In Tok.	Out Tok.
Gemini-2.5-Flash *(Standard | Thinking)*								
Zero-shot	41.80 | 44.97	32.39 | 37.68	17.57 | 17.93	– | +2.0	1.98 | 2.74	– | −38.7	1,390 | 1,390	73 | 125
Zero-shot-CoT	38.38 | 44.98	30.20 | 36.01	16.42 | 17.36	−6.5 | −1.2	2.06 | 3.03	−4.4 | −53.4	1,404 | 1,404	66 | 134
1-shot	63.03 | 64.42	47.48 | 51.10	39.15 | 39.60	+122.8 | +125.4	1.84 | 2.35	+6.9 | −18.9	1,961 | 1,961	74 | 113
2-shot	65.88 | 71.42	54.80 | 57.90	46.41 | 47.92	+164.1 | +172.7	**1.74** | 2.21	**+11.8** | −11.7	2,555 | 2,555	66 | 99
3-shot	**70.22** | **75.67**⋆	**58.51** | **62.26**⋆	**49.57** | **52.45**⋆	**+182.1** | **+198.5**⋆	1.76 | 2.15	+10.6 | −9.0	3,202 | 3,202	73 | 99
GEAR (ours)	61.84 | 62.95	47.11 | 51.44	37.10 | 38.83	+111.2 | +121.0	1.86 | 2.50	+5.7 | −26.3	3,148 | 3,147	98 | 134
Critical-**GEAR** (ours)	52.40 | 52.71	41.20 | 45.34	29.73 | 29.30	+69.2 | +66.8	1.88 | 2.35	+5.0 | −18.8	2,021 | 2,042	87 | 125
GEAR+2-shot (ours)	69.34 | 68.47	51.41 | 55.78	45.04 | 46.35	+156.3 | +163.8	1.80 | 2.40	+8.7 | −21.5	4,305 | 4,308	94 | 115
2-shot+**GEAR** (ours)	69.33 | 69.60	53.15 | 55.42	44.77 | 46.27	+154.8 | +163.3	1.79 | **1.93**	+9.5 | **+2.4**	4,308 | 4,308	93 | 118
DirectiveOnly-**GEAR**	50.52 | 52.74	40.79 | 44.53	28.54 | 28.28	+62.4 | +61.0	1.84 | 2.52	+6.6 | −27.8	2,289 | 2,299	95 | 142
PedagogyOnly-**GEAR**	52.83 | 52.53	44.19 | 47.43	30.09 | 28.73	+71.3 | +63.5	1.75 | 2.37	+11.3 | −19.9	2,284 | 2,274	94 | 125

Table 10: Gemini-2.5-Flash **GEAR** results on $\mathcal{D}_{\text{test}}$. E2E Pass@1 (%) is the primary metric. Values shown as Standard | Thinking. ΔE2E = relative improvement over Gemini-2.5-Flash Zero-shot (Standard, 17.57%); ΔLat. = speedup relative to Gemini-2.5-Flash Zero-shot (Standard, 1.98s), where positive values indicate faster inference. Best results per mode in **bold**; ⋆overall best across all configurations. Row colors: **GEAR** variants; ablation studies.

Evaluation Model	E2E Accuracy (%)		Win Rate		Rule Count		Avg. Characters	
	Standard	Thinking	Standard	Thinking	Standard	Thinking	Standard	Thinking
Claude-4.5-Sonnet (Std)	41.07	43.42	30 (39.0%)	47 (61.0%)	8.9/4/12	9.0/5/12	720	746
Claude-4.5-Sonnet (Think)	41.91	44.07	30 (39.0%)	47 (61.0%)	9.2/7/11	9.0/5/12	722	743
GPT-5.1 (Std)	34.92	42.88	23 (29.9%)	54 (70.1%)	9.3/7/11	9.1/6/12	783	783
GPT-5.1 (Think)	40.64	44.38	32 (41.0%)	46 (59.0%)	9.2/6/11	8.8/6/12	727	768
Qwen3-30B (Std)	44.35	44.61	36 (46.8%)	41 (53.2%)	9.4/7/12	8.8/4/11	718	764
Qwen3-4B (Std)	37.54	43.97	26 (33.8%)	51 (66.2%)	9.4/7/12	9.0/6/12	736	739
Ministral3-14B (Std)	34.45	38.58	31 (39.7%)	47 (60.3%)	9.5/7/12	9.3/7/11	737	731
Gemini-2.5-Flash (Std)	27.49	35.53	23 (29.5%)	55 (70.5%)	9.8/8/12	9.3/6/12	739	741
Gemini-2.5-Flash (Think)	30.43	37.06	22 (28.2%)	56 (71.8%)	9.5/8/12	9.4/6/12	751	732
Average	36.99	41.64	252 (36.2%)	444 (63.8%)	9.3/4/12	9.1/4/12	735	749

Table 11: Extraction mode comparison across all models. Standard and Thinking columns show average E2E accuracy (%) using rules from each extractor. Rules shows mean/min/max rule count; Chars shows average characters per rule set.