EXECTUNE: EFFECTIVE STEERING OF BLACK-BOX LLMS WITH GUIDE MODELS

Vijay Lingam*
AWS AI
vjlingam@amazon.com

Aditya Golatkar
AWS AI

Anwesan Pal
AWS AI

Ben Vo
AWS AI

Narayanan Sadagopan
AWS AI

Alessandro Achille
AWS AI

Jun Huan
AWS AI

Anoop Deoras
AWS AI

Stefano Soatto
AWS AI

ABSTRACT

For large language models deployed through black-box APIs, recurring inference costs often dominate one-time training costs, motivating composed agentic systems that amortize expensive reasoning into reusable intermediate representations. We study a broad class of such systems, termed *Guide–Core Policies* (GCOP), in which a *guide* model generates a structured *strategy* that is executed by a black-box *core* model. This abstraction subsumes base, supervised, and advisor-style approaches, which differ primarily in how the guide is trained. We formalize GCOP under a cost-sensitive utility objective and show that end-to-end performance is governed by *guide-averaged executability*: the probability that a strategy can be faithfully followed by the core. Our analysis reveals that existing instantiations of GCOP often fail to optimize executability under deployment constraints, leading to brittle strategies and inefficient computation. Guided by these insights, we propose EXECTUNE, a principled training recipe that combines teacher-guided acceptance sampling, supervised fine-tuning, and structure-aware reinforcement learning to directly optimize syntactic validity, execution success, and cost efficiency. Across mathematical reasoning and code-generation benchmarks, GCOP with EXECTUNE improves accuracy by up to **9.2%** over prior state-of-the-art baselines while reducing inference cost by up to **22.4%**. GCOP with EXECTUNE enables Claude Haiku-3.5 to surpass Sonnet-3.5 on math and code tasks and comes within **1.7%** absolute accuracy of Sonnet 4 at **38%** lower cost. Beyond efficiency, GCoP enables modular adaptation by updating guides without retraining the core.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong performance on complex reasoning and programming tasks, yet their deployment at scale remains constrained by inference cost, latency, and limited adaptability under black-box access. As inference is invoked repeatedly in downstream applications, these recurring costs often dominate one-time training expenses, motivating the design of agentic systems that amortize expensive reasoning into reusable intermediate representations or memories. Thus, agentic inference can be formally written as a cost-sensitive net utility objective (Achille & Soatto, 2026; Zabounidis et al., 2025; Kleinman et al., 2025)

$$\boxed{J^\pi(s_0) = V^\pi(s_0) - \lambda T^\pi(s_0)} \tag{1}$$

where, π is the agentic policy, s_0 an initial state (or task), and V^π denotes the expected task reward, T^π denotes the expected inference-time cost, and λ controls the trade-off between performance and

*Corresponding author.

computation. This formulation captures realistic deployment settings in which marginal improvements in accuracy must be weighed against increased latency and expense.

This objective suggests a simple design principle: separate expensive deliberation from cheap execution. To address this challenge, we propose a broad class of composed policies, *Guide-Core Policies* (GCoP) that decomposes reasoning into two stages: a *small trainable-guide* model generates a high-level plan, advice, or strategy, and a *small black-box core* model executes it to produce the final output for a given task. GCoP improves efficiency and reliability by making abstract reasoning learnable and reusable across tasks while keeping execution *cheap and robust*. Moreover, GCoP subsumes and outperforms baselines like prompting, supervised fine-tuning, and ADVISOR models (Asawa et al., 2025; Li et al., 2025), using a composed policy whose behavior is governed jointly by guide generation, execution reliability, and computational cost.

Our analysis identifies executability as the key component in GCoP: the probability that a guide-produced strategy is parseable and can be successfully followed by the core. We show via a student–teacher mixture analysis (see Figure 3) that the performance gap to a large black-box baseline is controlled by guide-averaged executability; we reduce this gap through *guide-training* which explicitly raises executability. The result is structured, repeatable, reliable plans—strategies that the core can consistently execute—yielding higher utility by turning high-level reasoning into predictable, reusable action (subsection A.4).

Despite this promise, existing guide–core and advisor-style instantiations rarely train the guide for the quantity that matters in GCoP: executability. Prompted or supervised guides can be informative yet underspecified, while advisor-style models typically optimize advice quality in isolation rather than the downstream constraints imposed by a smaller, cost-constrained core. Consequently, they often generate strategies that are difficult to parse, hard to follow, or mismatched to the core's capabilities, leading to brittle behavior and wasted inference (see Appendix C).

Guided by our theoretical insights, we propose EXECTUNE, a principled training recipe for learning effective executable guides. EXECTUNE proceeds in two stages. First, it initializes the guide using teacher-guided acceptance sampling and supervised fine-tuning to bias generation toward executable strategies. Second, it performs structure-aware reinforcement learning with explicit rewards for syntactic validity, execution success, and cost-efficient behavior, while penalizing brittle or unparseable outputs. This design directly aligns guide training with the utility objective induced by GCoP.

Main Contributions.

1. **GCoP framework.** We formalize **GCoP** as a composed guide-core policy abstraction, where a trainable guide model generates context that a black-box core executes. Our theory building on net-utility objective identifies *guide-averaged executability* as the key factor controlling the value gap to stronger core-only baselines.
2. **Teacher-guided acceptance sampling for strategy SFT.** We construct high-executability strategy data by using a strong teacher to propose strategies and the target core to validate them, yielding an SFT initialization tailored to the core's execution constraints (subsection 3.4).
3. **EXECTUNE: executable strategy post-training.** We propose EXECTUNE, which refines the guide with structure-aware GRPO using rewards for (i) correct strategy format/parseability and (ii) execution success, and penalties for missing, malformed, or digressive strategies (subsection 3.5).
4. **Improved utility at lower costs.** Across math and code benchmarks, GCoP with EXECTUNE improves accuracy by up to **9.2%** while reducing inference cost by up to **22.4%** over prior black-box steering baselines, and approaches next-generation cores (within **1.7%** of Sonnet-4) at **38%** lower cost (See Figure 1).

Beyond efficiency, GCoP enables modular adaptation: guides can be updated for domain adaptation, continual learning, or targeted unlearning without retraining the core. This modularity is particularly valuable in black-box deployment settings, where the core model cannot be modified directly.

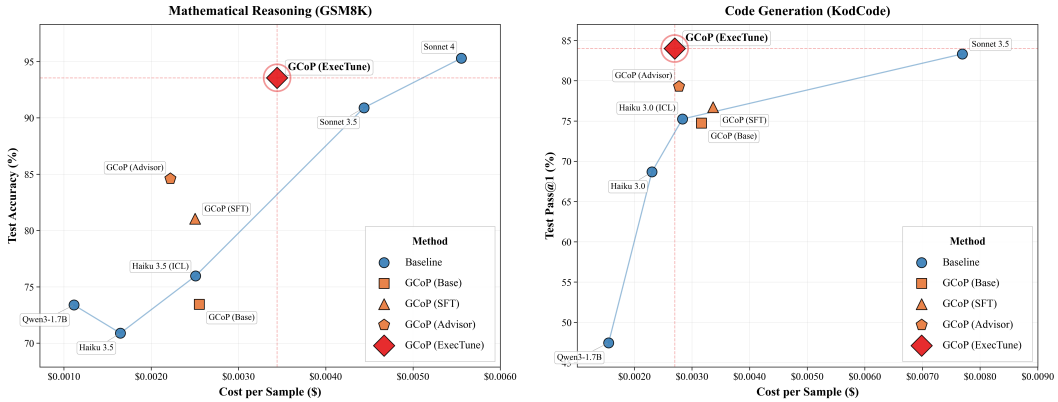

Figure 1: **Reward–cost trade-off.** Test performance versus *total inference cost* on GSM8K (left; core=**Haiku-3.5**) and KodCode (right; core=**Haiku-3.0**). A small **ExecTune** guide (**Qwen3-1.7B**) yields the best accuracy–cost trade-off among GCoP variants, outperforming prompting/ICL and GCoP(Base/SFT/Advisor) while remaining far cheaper than frontier baselines. GCoP(ExecTune) matches or exceeds **Sonnet-3.5** and approaches **Sonnet-4** at substantially lower cost, showing that executable strategies can reliably steer black-box cores toward large-model performance.

2 Related Work

Recent advances in large language models (LLMs) have sparked growing interest in methods that adapt black-box APIs to user needs without requiring model retraining or internal access. Our work intersects with three key research threads: (1) prompt optimization for black-box LLMs, (2) learned steering via guide models and controllers, and (3) knowledge distillation and cost-efficient deployment strategies. This section surveys each area and situates our framework in the broader landscape.

2.1 Prompt Optimization for Black-Box LLMs

Prompt optimization methods aim to elicit desirable behaviors from LLMs through natural language instructions, without modifying model parameters. Early strategies included gradient-free techniques like AutoPrompt (Shin et al., 2020), which searched for discrete token triggers. Black-box-compatible methods have since emerged, including evolutionary and bandit-style approaches such as PromptWizard (Agarwal et al., 2024), PromptBreeder (Fernando et al., 2023), and BBPO (Cheng et al., 2024), which optimize prompts through trial-and-error. GEPA (Agrawal et al., 2025), DoT (Lingam et al., 2025) and Self-Refine (Madaan et al., 2023) introduce reflective or self-critiquing mechanisms, showing that language models themselves can evaluate and improve prompts via feedback loops.

These methods generate task-general or input-agnostic prompts. However, static optimization may fall short for dynamic tasks or personalization. This limitation motivates adaptive prompting techniques that produce input-conditional guidance at inference time.

2.2 Learned Steering with Guide Models

Guide models generate instance-specific natural language instructions to steer black-box LLMs. Guide Models (Asawa et al., 2025) and Directional Stimulus Prompting (Li et al., 2023) train lightweight controllers to craft tailored prompts, using a combination of supervised learning and reinforcement learning (RL) from the LLM's feedback. Matryoshka Pilot (M-Pilot) (Li et al., 2025) treats the LLM as an environment, with a learned guide breaking complex tasks into subtasks across multiple turns.

These approaches echo structured prompting techniques like Least-to-Most Prompting (Zhou et al., 2022) and Program-of-Thought prompting (Chen et al., 2022), which inject intermediate reasoning steps to improve task decomposition. At decoding time, methods like FUDGE (Yang & Klein, 2021) and DExperts (*et al.*, 2021a) steer generation through learned discriminators or ensemble scoring,

though they require token-level access. Guide models operate fully at the language level and work even with closed-source APIs. They complement alignment frameworks such as InstructGPT (Ouyang et al., 2022) and Constitutional AI (Bai et al., 2022), which apply RLHF or AI-based critiques. Prompt rewriting frameworks like BPO (Cheng et al., 2024) further demonstrate that modifying the user's query alone can boost output quality from API-tuned models. Tool-augmented prompting methods such as ReAct (Yao et al., 2022) and Auto-CoT (Zhang et al., 2022) highlight how LLMs can dynamically generate intermediate reasoning or query tools to improve response quality.

Our work builds on these principles, using a two-stage approach: an offline supervised step followed by online RL, improving robustness under limited feedback. Unlike prior work, we focus on leveraging learned guides for scale-bridging and cost-sensitive inference.

2.3 Knowledge Distillation and Cost Efficiency

To make LLM deployment more affordable, knowledge distillation techniques aim to transfer capabilities from large models to smaller ones. In black-box contexts, this is done using only the teacher's outputs. Step-by-step distillation (Hsieh et al., 2023) and Chain-of-Thought distillation (Do et al., 2025) show that intermediate rationales help teach reasoning to smaller models. Recent approaches like MiniLLM (Gu et al., 2023) and Direct Preference Knowledge Distillation (DPKD) (Li et al., 2024) use reward modeling or reverse-KL objectives to preserve quality and stylistic traits. However, distillation may strip away safety mechanisms: Jahan and Sun (Jahan & Sun, 2025) showed that models cloned from API outputs can retain task ability while losing alignment safeguards. To reduce inference cost without retraining, system-level methods route queries through model cascades. FrugalGPT (Chen et al., 2023) and ABC (Kolawole et al., 2024) dynamically assign tasks to models of varying size based on confidence or agreement. Tool-augmented prompting complements these strategies. Toolformer (Schick et al., 2023) and ReAct (Yao et al., 2022) teach LLMs to call APIs mid-generation, enhancing factuality and reasoning.

Our method aligns with these directions by training a guide that enables a compact LLM to benefit from the reasoning capacity of a larger one without direct distillation. The result is a cost-effective, steerable deployment strategy that preserves model modularity and API compatibility.

3 Method

Overview. We study *Guide-Core Policies* (GCoP): composed agentic systems where a *trainable guide* (e.g., Qwen3-1.7B (*et al.*, 2025)) produces an explicit, *parseable* strategy and a (typically cheaper) *black-box core* (e.g., Claude Haiku 3[1]) executes conditioned on it. Our goal is to understand when such composed systems can match strong but expensive large-model baselines under realistic deployment constraints, and how to train guides whose strategies are reliably executable by the target core model. This section (i) formalizes GCoP under a cost-sensitive net-utility objective, (ii) identifies *guide-averaged executability* as the primary bottleneck governing the value gap to large-model baselines, and (iii) introduces ExecTune, a training recipe that explicitly aligns guide outputs with downstream execution constraints.

3.1 Objective: reward–cost net utility

We model tasks as finite-horizon interactions with reward bounded by $R_{\max}$, and measure inference-time deployment cost (tokens, latency, or monetary) via $T_\pi(\cdot)$. For any policy π and initial state s_0, we optimize the net utility

$$\boxed{J_\pi(s_0) \;\triangleq\; V_H^\pi(s_0) \;-\; \lambda\, T_\pi(s_0),} \tag{2}$$

where $V_H^\pi(s_0) \triangleq \mathbb{E}[\sum_{t=0}^{H-1} r_t \mid s_0, \pi]$ and $\lambda \geq 0$ trades off performance and compute. This objective captures deployment regimes where small gains in accuracy must be weighed against recurring inference costs.

[1] https://www.anthropic.com/news/claude-3-haiku

3.2 GCoP AS A COMPOSED POLICY FAMILY

Let $\mathcal{Z}$ denote a *strategy* space (e.g., plan, advice, program sketch, structured prompt). A guide policy $\pi_g(z \mid s)$ samples a strategy $z \in \mathcal{Z}$ given state s, and a black-box core $\pi_c(a \mid s, z)$ produces the final action/output conditioned on (s, z). Composing guide and core yields the induced policy

$$\boxed{\pi_{gc}(a \mid s) \triangleq \sum_{z \in \mathcal{Z}} \pi_g(z \mid s)\, \pi_c(a \mid s, z).} \tag{3}$$

Different GCoP instantiations correspond to different ways of obtaining π_g (Base, SFT, advisor-style, or ours), while keeping the target black-box core model fixed.

3.3 EXECUTABILITY CONTROLS THE VALUE GAP

A central failure mode in guide–executor systems is *non-executability*: the guide outputs strategies that are not parseable or not faithfully followable by the target core, leading to wasted compute and brittle behavior. We formalize this via a per-(state,strategy) success probability $q(s, z) \triangleq \Pr(G = 1 \mid s, z)$, where $G = 1$ denotes "good execution" under the environment/validator.

We define the *guide-averaged executability* as:

$$\boxed{\alpha(s) \triangleq \mathbb{E}_{z \sim \pi_g(\cdot|s)}[q(s, z)].} \tag{4}$$

Under a student–teacher mixture view (detailed in Appendix A.4), the induced policy π_{gc} can be expressed as a mixture between a strong teacher policy π_L and an aggregate "bad-execution" component, weighted by $\alpha(s)$. This yields the following bound on the value gap:

Theorem 1 (Value gap controlled by executability)**.** *Under the mixture model assumptions in Appendix A.4, for any s_0,*

$$\boxed{V_H^{\pi_L}(s_0) - V_H^{\pi_{gc}}(s_0) \;\le\; 2HR_{\max}\, \mathbb{E}_{s \sim d_L}\big[1 - \alpha(s)\big],} \tag{5}$$

where d_L is the (average) state visitation distribution under π_L.

Thus, to close the gap to strong cores, the guide must *increase* $\alpha(s)$ *on task-relevant states*, i.e., produce strategies that the target core can reliably execute.

3.4 ACCEPTED-STRATEGY TARGET DISTRIBUTION FOR STRATEGY SFT

To increase $\alpha(s)$, we build a strategy dataset that matches the target core's execution constraints. We use *teacher-guided acceptance sampling*: a strong teacher proposes strategies z, and we *accept* those for which the target core (or validator) achieves high empirical success when executing z. This defines an *accepted-strategy* distribution $\pi_{\mathrm{acc}}(\cdot \mid s)$ (see Appendix A.6).

Acceptance sampling provably reweights toward high-executability strategies:

Theorem 2 (Accepted strategies have high expected success)**.** *Fix s and accept z iff $\hat{q}_K(s, z) \ge \tau$ using K validation trials. Then for any $\eta > 0$,*

$$\boxed{\mathbb{E}_{z \sim \pi_{\mathrm{acc}}(\cdot|s)}[q(s, z)] \;\ge\; (\tau - \eta)\Big(1 - \frac{e^{-2K\eta^2}}{A_s}\Big),} \tag{6}$$

where A_s is the acceptance rate.

We then initialize the guide by SFT on $(s, z) \sim \pi_{\mathrm{acc}}(\cdot \mid s)$, which directly trains the guide toward strategies the core can execute.

3.5 EXECTUNE: EXECUTABLE GUIDE POST-TRAINING

We propose EXECTUNE, a two-stage recipe that aligns guide generation with downstream executability and net utility.

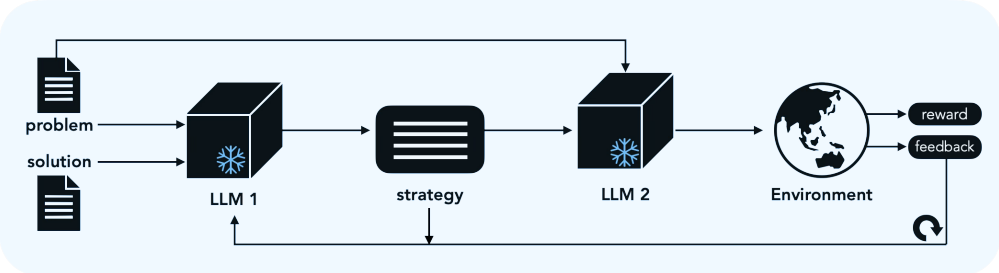

Figure 2: SFT dataset curation pipeline. A strong model (LLM-1; e.g., Claude Sonnet 4.5) extracts a high-level strategy from a (problem, solution) pair. A weaker frontier model (LLM-2; e.g., Claude Haiku 3) generates a solution conditioned on the problem and strategy. An environment provides reward and feedback; unsuccessful strategies are iteratively refined by LLM-1. The resulting (problem, strategy) pairs form the SFT corpus.

Stage 1: teacher-guided acceptance sampling + SFT. We curate executable strategies using the pipeline in Fig. 2 and fine-tune the guide to imitate accepted strategies, improving $\alpha(s)$ by construction.

Stage 2: structure-aware GRPO with executability shaping. We further refine the guide with GRPO (Shao et al., 2024) algorithm using a shaped reward that enforces a structured guide–core interface, penalizes leakage of final answers into the strategy, and discourages strategies that harm core behavior.

Structure validity. We require the guide to emit a well-formed `<strategy>...</strategy>` block that is parseable by a deterministic checker. Let

$$\mathbb{I}_{\text{str}} \triangleq \mathbf{1}\{\texttt{<strategy>} \text{ block is present and well-formed}\}.$$

Judge-based strategy quality and non-leakage. Let $J(s, z) \in [0, 1]$ be an LLM-as-a-Judge score applied to the *parsed* strategy, capturing strategy quality and verifying that it does not contain the final answer (e.g., full code). We treat J as an additional shaping signal.

No-negative-behavior shaping. Let y denote the final output produced by the core. We define a *non-degradation* term that penalizes strategies that reduce performance relative to running the core without the guide:

$$\Delta R(s, z) \triangleq R\big(\pi_c(\cdot \mid s, z)\big) - R\big(\pi_c(\cdot \mid s)\big), \tag{7}$$

where $R(\cdot)$ denotes the episodic task reward induced by the corresponding core execution distribution.[2]

Final shaped reward. Combining these components, we train the guide using the shaped reward

$$\boxed{\tilde{R} \triangleq \mathbb{I}_{\text{str}}\Big(R + \beta + \gamma\, J(s, z) - \kappa\,\big[-\Delta R(s, z)\big]_+\Big)} \tag{8}$$

where $\beta \geq 0$ is a small structural bonus, $\gamma \geq 0$ weights the judge score, $\kappa \geq 0$ penalizes degradations, and $[x]_+ \triangleq \max\{x, 0\}$. Thus malformed strategies receive $\tilde{R} = 0$ even if the final answer is correct; valid strategies are rewarded for execution success, high-quality non-leaky guidance, and are discouraged from harming baseline core behavior. Full judge design and additional implementation details are deferred to Appendix A.7.

Instantiations. Our framework subsumes common guide training choices: (i) **GCoP(Base)** (prompted guide; no fine-tuning), (ii) **GCoP(SFT)** (guide fine-tuned on curated strategies dataset), (iii) **GCoP(Advisor)** (guide trained following the recipe in (Asawa et al., 2025)), and (iv) **GCoP(ExecTune)** (ours), which explicitly optimizes executability under the deployment utility objective in Eq. equation 2.

[2]In deterministic settings, $R(\pi_c(\cdot \mid \cdot))$ is simply the reward of the realized output y.

4 EXPERIMENTS

In this section, we describe the models, datasets, and evaluation protocol used to instantiate GCoP in black-box settings, and present results across mathematical reasoning and code generation. Implementation and prompt details are deferred to appendices B and D.

Training datasets. For code generation, we use KodCode (Xu et al., 2025). Following Ekbote et al. (2025), we randomly sample 1,000 problems from KodCode and split them into train/test with an 80:20 ratio. For mathematical reasoning, we train on the GSM8K (Cobbe et al., 2021) training split.

Evaluation datasets. Guides trained on KodCode are evaluated on the KodCode test split and HumanEval (*et al.*, 2021b) (out-of-domain code). For mathematical reasoning, we evaluate on the GSM8K test split.

Models and nomenclature. We evaluate GCoP with *black-box cores* accessed via proprietary APIs and *open-weight guides*. We use **Claude 3.5 Haiku** as the core for GSM8K and **Claude 3 Haiku**[3] as the core for code tasks (KodCode/HumanEval), chosen because they are cost-efficient yet leave room for improvement. We report **Claude 3.5 Sonnet**[4] as a stronger core-only reference point. For guides, we experiment with **Qwen3-1.7B**.

Baselines and variants. We compare against: (i) **ICL** (core-only), where we retrieve the three nearest (problem, solution) training pairs via cosine similarity over prompt embeddings obtained using Cohere Embed v4 model[5] and prepend them to the core prompt; (ii) **Advisor-models** (Asawa et al., 2025), denoted ADVISOR in tables, which trains the guide with RL to provide helpful advice to a fixed black-box core.

In **guided generation** mode, the guide first emits an explicit, parseable strategy z, and the core produces the final answer conditioned on both the original problem and z. We evaluate the following guide training recipes (FT Recipe in tables):

- **None**: zero-shot prompted guide (no fine-tuning).
- **SFT**: supervised fine-tuning on accepted strategy data (subsection 3.4).
- **ADVISOR**: advisor-style RL (Asawa et al., 2025).
- **EXECTUNE (ours)**: SFT initialization followed by structure-aware GRPO (subsection 3.5).

Metrics. We report Pass@1 for code generation and exact-match accuracy for GSM8K. Each results table is split into two blocks: (i) *core-only baselines* (no guide), and (ii) *guided generation* with the specified guide recipe.

4.1 MATHEMATICAL REASONING (GSM8K)

Results. Table 1a reports GSM8K accuracy. First, core-only ICL improves Claude 3.5 Haiku by **+5.08** points (70.89%→75.97%). Second, using an unfine-tuned guide yields only a modest gain (70.89%→73.46%), indicating that naive strategies are not consistently executable by the target core. Third, both SFT and ADVISOR substantially improve guided performance, with SFT being a strong and sample-efficient baseline. Finally, our EXECTUNE recipe achieves the best performance: **93.56%** accuracy, improving over ADVISOR by **+8.95** points (84.61%→93.56%) and over Claude 3.5 Haiku core-only by **+22.67** points (70.89%→93.56%). These gains support our thesis that explicitly optimizing the guide for executability and downstream success is critical in black-box settings.

4.2 CODE GENERATION (KODCODE, HUMANEVAL)

We train code guides on KodCode-train using the previously introduced recipes (None/SFT/ADVISOR/EXECTUNE). Strategy SFT data construction follows the same acceptance-sampling

[3] https://www.anthropic.com/news/claude-3-haiku

[4] https://www.anthropic.com/news/claude-3-5-sonnet

[5] https://cohere.com/blog/embed-4

We instantiate GCoP on GSM8K using **Claude 3.5 Haiku** as the black-box core and **Qwen3-1.7B** as the guide. For context, we additionally report the standalone performance of Qwen3-1.7B and Claude 3.5 Sonnet under standard prompting.

SFT strategy data construction. We construct the SFT corpus using teacher-guided acceptance sampling (Fig. 2). A strong teacher (by default **Claude Sonnet 4.5**) proposes an initial strategy. We concatenate the strategy with the original problem and query the *target core* (**Claude 3.5 Haiku**) to generate a solution, scored by exact-match on GSM8K. If the strategy fails, the teacher refines it using feedback and we repeat for a small number of iterations; accepted $(\text{problem}, \text{strategy})$ pairs form the SFT corpus. We also ablate different teachers (e.g., Sonnet 4, Sonnet 3.5) while keeping the target core fixed; Fig. 3 summarizes the effect of teacher choice and refinement iterations. Prompts are provided in Appendix E.

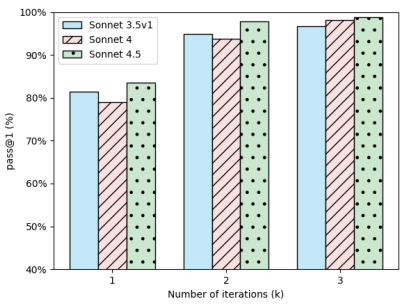

Figure 3: Effect of iterative strategy refinement during acceptance sampling: additional refinement iterations increase the probability that the target core solves the problem when conditioned on the proposed strategy.

Table 1: Performance on math and code benchmarks. Guided generation with ExecTune achieves the strongest results, demonstrating that optimizing guides for executability improves black-box core performance. FT Recipe denotes the guide fine-tuning recipe.

(a) **GSM8K (Math)**. ExecTune achieves 93.56% accuracy; **+22.67** points over Claude 3.5 Haiku alone.

Core (API)	Guide	FT Recipe	Acc.
Baselines (No Guided Generation)			
Qwen3-1.7B	—	—	73.39%
Claude 3.5 Haiku	—	—	70.89%
Claude 3.5 Haiku (ICL)	—	—	75.97%
Claude 3.5 Sonnet	—	—	90.90%
Guided Generation (Core = Claude 3.5 Haiku)			
Claude 3.5 Haiku	Qwen3-1.7B	None	73.46%
Claude 3.5 Haiku	Qwen3-1.7B	SFT	81.05%
Claude 3.5 Haiku	Qwen3-1.7B	Advisor	84.61%
Claude 3.5 Haiku	Qwen3-1.7B	ExecTune	**93.56%**

(b) **KodCode and HumanEval (Code)**. ExecTune enables Claude 3 Haiku to surpass Claude 3.5 Sonnet on KodCode (**84.00%** vs 83.33%) and HumanEval (**91.46%** vs 84.15%).

Core (API)	Guide	FT Recipe	KodCode Pass@1	HumanEval Pass@1
Baselines (No Guided Generation)				
Qwen3-1.7B	—	—	47.47%	71.39%
Claude 3 Haiku	—	—	68.69%	73.17%
Claude 3 Haiku (ICL)	—	—	75.25%	68.67%
Claude 3.5 Sonnet	—	—	83.33%	84.15%
Guided Generation (Core = Claude 3 Haiku)				
Claude 3 Haiku	Qwen3-1.7B	None	72.56%	79.27%
Claude 3 Haiku	Qwen3-1.7B	SFT	76.73%	84.15%
Claude 3 Haiku	Qwen3-1.7B	Advisor	79.29%	79.88%
Claude 3 Haiku	Qwen3-1.7B	ExecTune	**84.00%**	**91.46%**

template as GSM8K, except that success is defined by *unit-test execution*: a proposed strategy is accepted if the target core solution passes the associated tests. Unless otherwise noted, the black-box core for code is **Claude 3 Haiku**.

Results on KodCode and HumanEval. Table 1b reports KodCode (in-domain) and HumanEval (out-of-domain) Pass@1. Core-only ICL substantially improves KodCode (+6.56 points) but *hurts* HumanEval (-4.50 points), consistent with retrieval overfitting to the KodCode training distribution. Guided generation with an unfine-tuned guide already yields gains over core-only prompting (e.g., +3.87 points on KodCode and +6.10 on HumanEval), but fine-tuning is crucial for robust improvements. SFT provides a strong baseline and matches the stronger core (Claude 3.5 Sonnet) on HumanEval (84.15%). Advisor-style training improves KodCode further but degrades HumanEval relative to SFT, highlighting a generalization trade-off. Our ExecTune recipe achieves the best overall performance, improving over Advisor by **+4.71** points on KodCode (79.29%→84.00%) and **+11.58** points on HumanEval (79.88%→91.46%), and surpasses Claude 3.5 Sonnet on both benchmarks.

4.3 Qualitative analysis of generated strategies

We qualitatively compare strategies from different GCoP guide training recipes on a representative KodCode test problem, keeping the black-box core fixed (**Claude 3 Haiku**). Base (prompted) and

ADVISOR guides often produce generic or underspecified advice that does not sufficiently constrain the core's implementation, while SFT guides are typically better structured but can be verbose or present multiple competing directions, leading to inconsistent or incomplete code. In contrast, EXECTUNE produces the most *executable* strategies that are actionable, interface-aligned, and reliably steer the core to implement the intended solution. Overall, these examples reinforce that gains come from improving *strategy executability* (parseable and faithfully followed plans), not from longer or more elaborate guidance; full prompts and outputs are provided in Appendix C.

5 CONCLUSION & LIMITATIONS

We introduced GCOP, a unified abstraction for guide-core steering systems under black-box access, and formalized their deployment trade-offs through a cost-sensitive net-utility objective. Our analysis identifies *guide-averaged executability* as the key factor governing the performance gap between composed policies and stronger, more expensive cores. Guided by this perspective, we proposed EXECTUNE, a two-stage recipe that (i) curates executable strategy supervision via teacher-guided acceptance sampling and (ii) refines guides with structure-aware RL to enforce parseable, reliably executable strategies. Across mathematical reasoning and code generation benchmarks, EXECTUNE improves accuracy while preserving the cost advantages of cheaper black-box cores, and in several settings approaches or surpasses stronger cores at substantially lower inference cost. **Limitations:** our experiments focus on single-turn tasks where a single strategy is generated and executed once; we do not evaluate more challenging multi-turn agentic settings that require iterative tool use, long-horizon planning, memory, or recovery from intermediate failures. Extending GCOP and EXECTUNE to such multi-step environments raises additional questions around strategy updating, credit assignment across turns, and robustness under distribution shift, which we leave to future work.

REFERENCES

Alessandro Achille and Stefano Soatto. AI agents as universal task solvers. *Entropy (also arXiv:2510.12066)*, 2026.

Eshaan Agarwal, Joykirat Singh, Vivek Dani, Raghav Magazine, Tanuja Ganu, and Akshay Nambi. Promptwizard: Task-aware prompt optimization framework. *arXiv preprint arXiv:2405.18369*, 2024.

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

Parth Asawa, Alan Zhu, Matei Zaharia, Alexandros G Dimakis, and Joseph E Gonzalez. How to train your advisor: Steering black-box llms with advisor models. *arXiv preprint arXiv:2510.02453*, 2025.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Jiale Cheng, Xiao Liu, Kehan Zheng, Pei Ke, Hongning Wang, Yuxiao Dong, Jie Tang, and Minlie Huang. Black-box prompt optimization: Aligning large language models without model training. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3201–3219, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John

Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Cong Thanh Do, Rama Sanand Doddipatla, and Kate Knill. Effectiveness of chain-of-thought in distilling reasoning capability from large language models. In *Proceedings of the 18th International Natural Language Generation Conference*, pp. 833–845, 2025.

Chanakya Ekbote, Vijay Lingam, Behrooz Omidvar Tehrani, Jun Huan, sujay sanghavi, Anoop Deoras, and Stefano Soatto. Murphy: Reflective multi-turn reinforcement learning for self-correcting code generation in large language. In *First Workshop on Foundations of Reasoning in Language Models*, 2025. URL https://openreview.net/forum?id=x0Ir7cWEiA.

Alisa Liu *et al.* DExperts: Decoding-time controlled text generation with experts and anti-experts. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 6691–6706, Online, August 2021a. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.522. URL https://aclanthology.org/2021.acl-long.522/.

An Yang *et al.* Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Mark Chen *et al.* Evaluating large language models trained on code, 2021b. URL https://arxiv.org/abs/2107.03374.

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

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*, 2023.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8003–8017, 2023.

Sohely Jahan and Ruimin Sun. Black-box behavioral distillation breaks safety alignment in medical llms. *arXiv preprint arXiv:2512.09403*, 2025.

Michael Kleinman, Matthew Trager, Alessandro Achille, Wei Xia, and Stefano Soatto. E1: Learning adaptive control of reasoning effort. *NeurIPS Workshop on Efficient Reasoning (also arXiv:2510.27042)*, 2025.

Steven Kolawole, Don Dennis, Ameet Talwalkar, and Virginia Smith. Agreement-based cascading for efficient inference. *arXiv preprint arXiv:2407.02348*, 2024.

ChangHao Li, Yuchen Zhuang, Rushi Qiang, Haotian Sun, Hanjun Dai, Chao Zhang, and Bo Dai. Matryoshka pilot: Learning to drive black-box llms with llms. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Yixing Li, Yuxian Gu, Li Dong, Dequan Wang, Yu Cheng, and Furu Wei. Direct preference knowledge distillation for large language models. *arXiv preprint arXiv:2406.19774*, 2024.

Zekun Li, Baolin Peng, Pengcheng He, Michel Galley, Jianfeng Gao, and Xifeng Yan. Guiding large language models via directional stimulus prompting. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pp. 62630–62656, 2023.

Vijay Lingam, Behrooz Omidvar Tehrani, Sujay Sanghavi, Gaurav Gupta, Sayan Ghosh, Linbo Liu, Jun Huan, and Anoop Deoras. Enhancing language model agents using diversity of thoughts. In *The 13th International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ZsP3YbYeE9.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

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

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

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

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4222–4235, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.346. URL https://aclanthology.org/2020.emnlp-main.346/.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformer Reinforcement Learning. https://github.com/huggingface/trl, 2020.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.365. URL https://aclanthology.org/2025.findings-acl.365/.

Kevin Yang and Dan Klein. FUDGE: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3511–3535, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.276. URL https://aclanthology.org/2021.naacl-main.276/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Renos Zabounidis, Aditya Golatkar, Michael Kleinman, Alessandro Achille, Wei Xia, and Stefano Soatto. Re-forc: Adaptive reward prediction for efficient chain-of-thought reasoning. *NeurIPS Workshop on Efficient Reasoning (also arXiv:2511.02130)*, 2025.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A APPENDIX

A.1 METHOD DETAILS

This appendix provides the full formalism and details omitted from the condensed section 3 in the main paper. We include: (i) the student–teacher mixture model and derivations linking executability to the value gap, (ii) the accepted-strategy target distribution induced by teacher-guided acceptance sampling and its guarantees, and (iii) the full EXECTUNE post-training procedure, including structure-aware GRPO.

A.2 PRELIMINARIES: OBJECTIVE AND COST

We model the environment as a finite-horizon controlled process with state space $\mathcal{S}$, action space $\mathcal{A}$, and horizon $H \in \mathbb{N}$. At each step $t \in \{0, \dots, H-1\}$ the agent observes $s_t \in \mathcal{S}$, selects $a_t \in \mathcal{A}$, receives reward $r_t = r(s_t, a_t) \in [0, R_{\max}]$, and transitions according to the environment dynamics. An initial state is drawn from $s_0 \sim \mu$ (user-query distribution). For any policy π, define the H-step value

$$V_H^\pi(s) \triangleq \mathbb{E}\left[\sum_{t=0}^{H-1} r(s_t, a_t) \,\middle|\, s_0 = s, \pi\right].$$

Let $T_\pi(s)$ denote the expected total inference-time cost of deploying π from s (tokens, latency, or monetary cost). We optimize the reward–cost tradeoff

$$J_\pi(s) \triangleq V_H^\pi(s) - \lambda T_\pi(s), \tag{9}$$

where $\lambda \geq 0$ controls cost sensitivity.

A.3 GCOP: GUIDE–CORE POLICIES AS A POLICY FAMILY

Let $\mathcal{Z}$ denote a strategy space (plans, advice, program sketches, structured prompts). A trainable guide $\pi_g(z \mid s) : \mathcal{S} \to \Delta(\mathcal{Z})$ samples a strategy conditioned on state s. A black-box core $\pi_c(a \mid s, z) : \mathcal{S} \times \mathcal{Z} \to \Delta(\mathcal{A})$ executes actions conditioned on both (s, z). Composing guide and core yields the induced action policy

$$\pi_{gc}(a \mid s) \triangleq \sum_{z \in \mathcal{Z}} \pi_g(z \mid s)\, \pi_c(a \mid s, z). \tag{10}$$

Throughout, GCOP denotes the *policy family* defined by Eq. equation 10 under the objective Eq. equation 9, where different instantiations correspond to different guide-training procedures.

When can cheaper GCOP outperform a large model? Let π_L denote a large and costly baseline policy (e.g., frontier API model). A sufficient condition for π_{gc} to improve net utility over π_L from a fixed initial state s is

$$\lambda T_{\pi_L}(s) - \lambda T_{\pi_{gc}}(s) \;>\; V_H^{\pi_L}(s) - V_H^{\pi_{gc}}(s), \tag{11}$$

i.e., the cost advantage must dominate the value gap. The following sections show that the value gap is governed by a single quantity: the guide-averaged executability $\alpha(s)$.

A.4 STUDENT–TEACHER MIXTURE MODEL AND EXECUTABILITY

We analyze GCOP as a student system (core) that can match teacher behavior when the guide proposes a *good* (executable) strategy.

Good strategy and success probability. Fix a state–strategy pair $(s, z) \in \mathcal{S} \times \mathcal{Z}$. Let $A \in \mathcal{A}$ be the random action/output produced by the core under $\pi_c(\cdot \mid s, z)$. Let $G \in \{0, 1\}$ indicate whether execution is "good" (application-dependent; e.g., passes tests, achieves reward above a threshold). Define the success probability

$$q(s, z) \triangleq \Pr(G = 1 \mid s, z),$$

and conditional action distributions $\pi_{\text{good}}(a \mid s, z) \triangleq \Pr(A = a \mid s, z, G = 1)$ and $\pi_{\text{bad}}(a \mid s, z) \triangleq \Pr(A = a \mid s, z, G = 0)$. Then, by total probability,

$$\pi_c(\cdot \mid s, z) = q(s, z)\, \pi_{\text{good}}(\cdot \mid s, z) + (1 - q(s, z))\, \pi_{\text{bad}}(\cdot \mid s, z). \tag{12}$$

Teacher-aligned good execution (assumption). We assume that when execution is good, the core matches the teacher policy:

$$\pi_{\text{good}}(\cdot \mid s, z) = \pi_L(\cdot \mid s) \quad \text{for all } (s, z). \tag{13}$$

This is a modeling abstraction capturing the idea that "good" executions correspond to teacher-level behavior (high reward/validity).

Guide-averaged executability and induced mixture. Combining Eq. equation 12 with the composed policy Eq. equation 10 and the teacher-alignment assumption yields

$$\pi_{gc}(\cdot \mid s) = \alpha(s)\, \pi_L(\cdot \mid s) + (1 - \alpha(s))\, \rho_s(\cdot), \tag{14}$$

where the mixture weight is the *guide-averaged executability*

$$\alpha(s) \triangleq \mathbb{E}_{z \sim \pi_g(\cdot|s)}[q(s, z)] = \sum_{z \in \mathcal{Z}} \pi_g(z \mid s)\, q(s, z), \tag{15}$$

and ρ_s aggregates bad-execution behavior across strategies:

$$\rho_s(\cdot) \triangleq \frac{1}{1 - \alpha(s)} \sum_{z \in \mathcal{Z}} \pi_g(z \mid s)\, (1 - q(s, z))\, \pi_{\text{bad}}(\cdot \mid s, z), \qquad (\alpha(s) < 1). \tag{16}$$

Value gap bound. Let $d_L \triangleq \frac{1}{H} \sum_{t=0}^{H-1} d_{\pi_L, t}$ be the average state visitation distribution under π_L. Then:

Theorem 3 (Value gap of GCoP)**.** *Under Eq. equation 14, for any* s_0*,*

$$V_H^{\pi_L}(s_0) - V_H^{\pi_{gc}}(s_0) \;\le\; 2HR_{\max}\, \mathbb{E}_{s \sim d_L}[1 - \alpha(s)]. \tag{17}$$

Utility condition. Combining Eq. equation 17 with the net-utility objective implies that GCoP improves net utility whenever the cost advantage dominates the executability shortfall:

$$\lambda\big(T_{\pi_L}(s_0) - T_{\pi_{gc}}(s_0)\big) \;>\; 2HR_{\max}\, \mathbb{E}_{s \sim d_L}[1 - \alpha(s)]. \tag{18}$$

This motivates guide training procedures that explicitly increase $\alpha(s)$ on task-relevant states.

A.5 Instantiations of GCoP

Different methods correspond to different choices of π_g:

- **GCoP(Base):** fixed prompted guide emitting strategies in a prescribed format.
- **GCoP(SFT):** guide trained by supervised fine-tuning on strategy data.
- **GCoP(Advisor):** guide trained using the recipe proposed by Asawa et al. (2025).
- **GCoP(ExecTune) (ours):** guide initialized by SFT checkpoint on accepted strategies and refined via structure-aware GRPO to explicitly optimize executability.

A.6 Accepted-strategy target distribution via teacher-guided acceptance sampling

Curation pipeline (expanded). We operationalize executable strategy supervision using teacher-guided acceptance sampling (Fig. 2): a strong teacher proposes candidate strategies for each problem instance, and the target core executes conditioned on each strategy. An environment/validator assigns success/failure (and potentially feedback). We retain strategies that pass a validation threshold, optionally iterating teacher refinement when failures occur. The resulting accepted (problem, strategy) pairs form the SFT corpus.

Accepted-strategy distribution. Fix state s. Let $z \sim \pi_L(\cdot \mid s)$ be a teacher-proposed strategy. We evaluate it with K i.i.d. trials (or stochastic executions) to obtain an empirical success $\hat{q}_K(s, z)$. We accept iff $\hat{q}_K(s, z) \ge \tau$, for threshold $\tau \in (0, 1)$. This induces the accepted distribution $\pi_{\text{acc}}(\cdot \mid s)$:

$$\pi_{\text{acc}}(z \mid s) \propto \pi_L(z \mid s) \cdot \mathbf{1}\{\hat{q}_K(s, z) \ge \tau\}.$$

Let $A_s \triangleq \Pr_{z \sim \pi_L(\cdot|s)}(\text{accept})$ denote the acceptance rate.

Guarantee: acceptance increases expected success.

Theorem 4 (Accepted strategies have high expected success)**.** *Fix s and accept iff $\hat{q}_K(s,z) \geq \tau$. For any $\eta > 0$, define $\delta \triangleq e^{-2K\eta^2}$. Then*

$$\Pr_{z\sim\pi_{\mathrm{acc}}(\cdot|s)}\big(q(s,z) \leq \tau - \eta\big) \;\leq\; \frac{\delta}{A_s},$$

and consequently

$$\mathbb{E}_{z\sim\pi_{\mathrm{acc}}(\cdot|s)}[q(s,z)] \;\geq\; (\tau-\eta)\Big(1 - \frac{\delta}{A_s}\Big). \tag{19}$$

Implication for executability. Training the guide to imitate $\pi_{\mathrm{acc}}(\cdot \mid s)$ via SFT increases $\alpha(s) = \mathbb{E}_{z\sim\pi_g}[q(s,z)]$ toward the accepted-strategy expectation in Eq. equation 19, up to imitation error. This is the first stage of EXECTUNE.

A.7 EXECTUNE DETAILS

Stage 1: SFT on executable strategies. We initialize the guide π_g by supervised fine-tuning on accepted strategy pairs $(s,z) \sim \pi_{\mathrm{acc}}(\cdot \mid s)$, using a fixed output schema that includes a required `<strategy>` block. This biases generation toward strategies that the target core can execute, directly improving $\alpha(s)$.

A.8 STRUCTURE-AWARE GRPO FOR GUIDE TRAINING

RL view. We treat the GCOP system as the deployed policy: the guide samples $z \sim \pi_g(\cdot \mid s)$, the core executes conditioned on (s,z), producing an outcome evaluated by the environment/validator to yield episodic reward R (e.g., undiscounted return). We apply GRPO updates to the guide parameters using this composite reward signal.

Why structure-aware shaping is necessary. Task reward alone can yield degenerate behavior: the guide may omit the explicit strategy or hide it in free-form text, while the core still sometimes succeeds. Such solutions do not improve the reusable guide–core interface and do not reliably increase executability in the sense required by the mixture model.

Structure-aware reward. We enforce an explicit interface by requiring a well-formed `<strategy>...</strategy>` block (optionally alongside `<thinking>...</thinking>`). Let

$$\mathbb{I}_{\mathrm{str}} \triangleq \mathbf{1}\{\texttt{<strategy>}\text{ block present and syntactically valid}\}.$$

We use the shaped reward

$$\tilde{R} \triangleq R \cdot \mathbb{I}_{\mathrm{str}} + \beta\, \mathbb{I}_{\mathrm{str}}, \tag{20}$$

where $\beta \geq 0$ is a small bonus for producing a valid strategy block. Thus malformed or missing strategies receive $\tilde{R} = 0$ even if the final answer is correct, preventing "implicit strategy" collapse.

Connection to executability. This shaping explicitly encourages parseable strategies and reduces brittle behavior. Empirically, it increases the probability that the guide emits usable strategies and improves downstream execution reliability, i.e., increases $\alpha(s)$, tightening the gap bound in Theorem 3.

A.8.1 JUDGE-BASED STRATEGY SHAPING

Parsed-strategy judge score. Let z denote the strategy parsed from the guide output. We compute a scalar judge score

$$J(s,z) \in [0,1],$$

using an LLM-as-a-Judge that evaluates (i) the usefulness/specificity of the strategy for solving s, and (ii) whether z improperly contains the final answer (e.g., a full code implementation), which would break the intended guide–core separation. We treat $J(s,z)$ as a reward shaping signal used only for training the guide.

Non-leakage constraint. If the judge flags direct answer leakage, we set $J(s, z) = 0$ (or equivalently apply an additional penalty), ensuring that leaked strategies are not reinforced even when the downstream execution succeeds. This promotes reusable high-level strategies rather than copying final solutions into the strategy channel.

A.8.2 NO-NEGATIVE-BEHAVIOR SHAPING (ANTI-REGRESSION)

Baseline-vs-guided core comparison. To ensure that guidance does not induce harmful behavior, we compare the reward of the core running *without* a strategy to the reward when conditioned on the guide's strategy. Let $y_0 \sim \pi_c(\cdot \mid s)$ denote the core output without guidance and $y_z \sim \pi_c(\cdot \mid s, z)$ denote the output with guidance. Let $r(\cdot)$ be the task reward computed by the environment/validator. Define

$$\Delta R(s, z) \triangleq \mathbb{E}\big[r(y_z)\big] - \mathbb{E}\big[r(y_0)\big], \tag{21}$$

where the expectation is over core sampling (and any environment stochasticity). When $\Delta R(s, z) < 0$, the strategy has degraded the core's behavior relative to the unguided baseline.

Penalty form. We penalize only degradations via the hinge $[-\Delta R]_+$, yielding a term $-\kappa[-\Delta R(s, z)]_+$ in the shaped reward. This avoids suppressing beneficial strategies ($\Delta R \geq 0$) while discouraging brittle advice that reduces accuracy.

A.8.3 FINAL SHAPED REWARD FOR GRPO

Let $\mathbb{I}_{\text{str}}$ indicate whether the guide output contains a well-formed `<strategy>` block, and let R denote the episodic task reward obtained from executing the core conditioned on the strategy (i.e., from the guided rollout). Our full shaped reward is

$$\tilde{R} = \mathbb{I}_{\text{str}}\Big(R + \beta + \gamma J(s, z) - \kappa[-\Delta R(s, z)]_+\Big), \tag{22}$$

with hyperparameters $\beta, \gamma, \kappa \geq 0$. This shaping enforces (i) explicit structured strategies, (ii) high-quality non-leaky strategies, and (iii) a conservative "do-no-harm" constraint relative to the unguided core.

Practical notes (implementation-level). In our implementation, $\mathbb{I}_{\text{str}}$ is computed via a deterministic parser that checks: (i) both opening/closing tags exist, (ii) tags are properly nested, and (iii) the extracted strategy is non-empty and within a token budget. We optionally add additional interface constraints (e.g., enumerated steps, JSON schema) as additional binary indicators; the main paper uses the minimal `<strategy>` schema for generality.

A.9 PUTTING IT TOGETHER: CONDENSED ALGORITHMIC DESCRIPTION

EXECTUNE (informal). (1) Use a strong teacher to propose candidate strategies; execute with the target core and accept those that pass validation to form π_{acc} and an SFT corpus. (2) SFT the guide on accepted strategies to initialize π_g. (3) Run structure-aware GRPO on the guide using shaped reward $\tilde{R}$ in Eq. equation 20, where the environment/validator reward is computed from the core execution. This explicitly aligns guide generation with downstream executability, improving $\alpha(s)$ and thus net utility under Eq. equation 9.

A.10 PROOFS

A.10.1 PROOF OF THEOREM 1

Proof. From $\pi_{gc}(\cdot \mid s) = \alpha(s)\pi_L(\cdot \mid s) + (1 - \alpha(s))\rho_s(\cdot)$ we have

$$\pi_{gc}(\cdot \mid s) - \pi_L(\cdot \mid s) = (1 - \alpha(s))\big(\rho_s(\cdot) - \pi_L(\cdot \mid s)\big).$$

Taking total variation and using homogeneity plus $\text{TV}(\rho_s, \pi_L) \leq 1$ yields

$$\text{TV}\big(\pi_{gc}(\cdot \mid s), \pi_L(\cdot \mid s)\big) \;\leq\; 1 - \alpha(s). \tag{23}$$

A standard finite-horizon simulation bound for undiscounted returns with rewards in $[0, R_{\max}]$ gives

$$V_H^{\pi_L}(s_0) - V_H^{\pi_{gc}}(s_0) \;\leq\; 2HR_{\max}\,\mathbb{E}_{s\sim d_L}\big[\text{TV}\big(\pi_{gc}(\cdot \mid s), \pi_L(\cdot \mid s)\big)\big]. \tag{24}$$

Combining equations (23) and (24) proves equation (17). □

A.10.2 PROOF OF THEOREM 2

Proof. Let $\hat{q}_K(s,z) = \frac{1}{K}\sum_{i=1}^{K} X_i$ be the empirical mean of K i.i.d. Bernoulli trials with mean $q(s,z)$, and accept iff $\hat{q}_K(s,z) \geq \tau$. Define the bad set $\mathcal{B}(s) \triangleq \{z : q(s,z) \leq \tau - \eta\}$. If $z \in \mathcal{B}(s)$, then $\tau - q(s,z) \geq \eta$, and Hoeffding's inequality implies

$$\begin{aligned}\Pr(\text{accept} \mid s,z) &= \Pr\big(\hat{q}_K(s,z) \geq \tau \mid s,z\big) \\ &= \Pr\big(\hat{q}_K(s,z) - q(s,z) \geq \tau - q(s,z) \mid s,z\big) \\ &\leq \Pr\big(\hat{q}_K(s,z) - q(s,z) \geq \eta \mid s,z\big) \\ &\leq e^{-2K\eta^2} = \delta.\end{aligned}$$

Now use the definition of the accepted distribution $\pi_{\text{acc}}(z \mid s) \propto \pi_L(z \mid s)\Pr(\text{accept} \mid s,z)$ with normalization A_s. Then

$$\begin{aligned}\Pr_{z\sim\pi_{\text{acc}}(\cdot|s)}(z \in \mathcal{B}(s)) &= \sum_{z\in\mathcal{B}(s)} \pi_{\text{acc}}(z \mid s) = \sum_{z\in\mathcal{B}(s)} \frac{\pi_L(z \mid s)\Pr(\text{accept} \mid s,z)}{A_s} \\ &\leq \sum_{z\in\mathcal{B}(s)} \frac{\pi_L(z \mid s)\delta}{A_s} \leq \frac{\delta}{A_s},\end{aligned} \tag{25}$$

which is $\Pr_{z\sim\pi_{\text{acc}}(\cdot|s)}\big(q(s,z) \leq \tau - \eta\big) \leq \frac{\delta}{A_s}$. For the expectation bound, let $\mathcal{G}(s) = \mathcal{B}(s)^c = \{z : q(s,z) > \tau - \eta\}$. Since $q(s,z) \geq 0$ everywhere and $q(s,z) \geq \tau - \eta$ on $\mathcal{G}(s)$,

$$\begin{aligned}\mathbb{E}_{z\sim\pi_{\text{acc}}(\cdot|s)}[q(s,z)] &\geq \mathbb{E}\big[q(s,z)\mathbf{1}\{z \in \mathcal{G}(s)\}\big] \geq (\tau-\eta)\Pr(z \in \mathcal{G}(s)) \\ &= (\tau-\eta)\big(1 - \Pr(z \in \mathcal{B}(s))\big) \geq (\tau-\eta)\Big(1 - \frac{\delta}{A_s}\Big),\end{aligned} \tag{26}$$

which is equation 19. □

A.11 ENSURING $\delta < A_s$ IN PRACTICE

The lower bound in Theorem 2 depends on the ratio δ/A_s, where $\delta = e^{-2K\eta^2}$ is the validation error term and $A_s = \Pr_{z\sim\pi_L(\cdot|s)}(\text{accept})$ is the (unknown) acceptance rate. To make the bound nonvacuous in practice, we estimate A_s from teacher proposals and form a high-confidence lower bound. Concretely, draw M independent strategies $z_1, \ldots, z_M \sim \pi_L(\cdot \mid s)$, run the acceptance test, and set $Y_j \triangleq \mathbf{1}\{z_j \text{ is accepted}\}$ so that $\mathbb{E}[Y_j] = A_s$. The empirical acceptance rate is $\widehat{A}_s \triangleq \frac{1}{M}\sum_{j=1}^{M} Y_j$. By Hoeffding, for any $\varepsilon > 0$, $\Pr\big(A_s \leq \widehat{A}_s - \varepsilon\big) \leq e^{-2M\varepsilon^2}$, equivalently with probability at least $1 - e^{-2M\varepsilon^2}$ we have the lower confidence bound $A_s \geq \widehat{A}_s - \varepsilon$. We then choose the validation strength (via K and/or η) so that $\delta \leq \widehat{A}_s - \varepsilon$, i.e., $e^{-2K\eta^2} \leq \widehat{A}_s - \varepsilon$. On the same high-probability event this implies $\delta \leq A_s$ and hence $\delta/A_s \leq 1$, ensuring the acceptance-sampling lower bound is nonvacuous. More conservatively, selecting $\delta \leq \frac{1}{2}(\widehat{A}_s - \varepsilon)$ guarantees $\delta/A_s \leq 1/2$ under the same event, making the factor $(1 - \delta/A_s)$ quantitatively meaningful.

B IMPLEMENTATION DETAILS

We implement our framework on top of TRL (von Werra et al., 2020), which provides efficient distributed training and a modular implementation of GRPO. We integrate TRL with vLLM for fast inference and large-scale rollout execution, enabling scalable multi-turn training in our experiments. Prompts used for training and evaluation are listed in Appendix D. The Qwen3 base guide models are available for research use under the Apache 2.0 license.

For RL post training with GRPO, we set the KL regularization factor $\beta = 0.004$, learning rate to 10^{-6}, weight decay to 0.1, batch-size to 8, rollouts to 8 and max-completion-length to 2048. During evaluation, we set the temperature to 0.6 and top-p to 0.95 for all experiments that use Qwen3 models.

C QUALITATIVE ANALYSIS OF GENERATED STRATEGIES

We provide the full problem statement and the complete strategy outputs for Base, SFT, ADVISOR, and EXECTUNE below.

Problem

When you are allocated a room for a programming competition, you have access to a bookshelf with M books. You have a list of k books you are particularly interested in reading during your free time. You want to ensure you have these books before anyone else takes them. Given two lists: the first list contains the titles of the M books currently on the bookshelf, and the second list consists of the k book titles you are interested in. Write a function that checks whether all the titles in your list are present on the bookshelf.

Strategy from Baseline Guide

Check if every book in the interested_books list is present in the bookshelf list. If all are present, return True; otherwise, return False.

Strategy from SFT Guide

Guidance for the All Books Available Problem: Step 1: Understand the Problem

- You need to verify if every book in your interest list exists on the bookshelf
- Return True if all interested books are present; False otherwise

Step 2: Consider Data Structures

- Think about what data structure would make checking ‘membership” (is this book present?) efficient
- Consider what data structure would make checking ‘subset” (are all my books present?) efficient

Step 3: Key Insight - Set Theory

- This is fundamentally a subset relationship problem
- If every element in set A exists in set B, then A is a subset of B
- Python’s set data structure has built-in methods for subset checking

Step 4: Solution Approach

1. Convert both lists to sets (this enables efficient membership testing)
2. Use set operations to determine if one set is entirely contained within another
3. Return True if all interested books are present, False otherwise

Step 5: Consider Edge Cases

- What if interested_books has duplicates? (Set conversion handles this naturally)
- What if interested_books contains books not on the bookshelf? (Set difference will show this)

Step 6: Python Set Methods to Consider

- `.issubset()` or equivalent operators to check if all elements of one set are in another
- Set intersection vs. difference vs. union - think about which relates to ‘all elements present”

Key Hint: Instead of iterating through each interested book and checking if it exists in the bookshelf, leverage Python’s set operations for a more elegant and Pythonic solution.

Strategy from ADVISOR model

Check if all books in interested_books are present in the bookshelf. For each book in interested_books, if any book is not found in the bookshelf, return False. Otherwise, return True. Use the in' keyword for membership check. The function should handle duplicates in interested_books by checking each book individually.

Strategy from EXECTUNE Guide

Step-by-step Guidance for Checking Book Availability: 1. Understand the Core Task:

- You need to verify that every book in your `interested_books` list exists somewhere in the `bookshelf` list.
- Example: If interested_books is `["A", "B"]` and bookshelf is `["B", "C", "A"]`, return `True`.

2. Key Question to Guide Toward:

- 'How can I efficiently check if each book in my interest list exists in the bookshelf collection?"
- Consider: What Python operations or methods can check membership or presence?

3. Alternative Approaches to Consider:

- Set operations: Sets have mathematical operations that can help identify common elements.
- Membership testing: Python's `in` operator can check if an element exists in a collection.
- List methods: There are built-in methods for searching within lists.

4. Edge Cases to Think About:

- Empty `interested_books` list (should return `True` by convention)
- Duplicate books in bookshelf
- Case sensitivity (problem assumes case-sensitive matching)

5. Pythonic Approach: Leverage Python's set operations or membership testing for clean, readable code. The gold solution uses set intersection, but list membership testing is also valid. ***Hint:*** *Python has built-in operators/methods that can check if all elements of one collection exist in another. Consider `set` operations or the `in` operator for membership checking.*

D PROMPTS FOR TRAINING AND EVALUATION

EXECTUNE System Prompt (code generation)

You are an expert Python programmer. Generate guidance for the given coding problem. Your response should be a strategy that helps guide the solution, not the implementation itself. `OUTPUT FORMAT:` You MUST format your response as follows:
1. First, use `<think></think>` tags for your internal reasoning
2. Then, use `<strategy></strategy>` tags for your final strategy

Example format:
`<think>`Let me analyze the problem...`</think><strategy>`strategy to solve the problem`</strategy>`

EXECTUNE Evaluation System Prompt (mathematical reasoning)

You are an expert mathematician. A strategy/hint has been provided to help guide your solution. You should consider this strategy if it seems reasonable and helpful, but you are free to use your own approach if you believe it's better. Your goal is to find the correct answer. The expected format for the final answer is
`boxed{answer}`

EXECTUNE Evaluation System Prompt (code generation)

You are an expert Python programmer. A strategy/hint has been provided to help guide your solution. You should consider this strategy if it seems reasonable and helpful, but you are free to use your own approach if you believe it's better. Your goal is to write correct, working code.

Write your full implementation (restate the function signature). The expected format for the final answer is ```` ```python python code goes here``` ````

E PROMPTS FOR SFT DATA GENERATION

Prompt to generate strategies

You are an expert Python programmer. Provide a high-level strategy, key insights, or solving approach for the given coding problem. Your response should be a brief strategy that helps guide the solution, not the implementation itself. OUTPUT FORMAT: You MUST format your response as follows:

- First, use `<think>...</think>` tags for your internal reasoning
- Then, use `<strategy>...</strategy>` tags for your final strategy

Example format:
`<think>Let me analyze the problem...</think> <strategy>Your concise strategy here</strategy>`
`<problem>` {problem} `</problem>`
`<solution>` {solution} `</solution>`

Prompt to generate solution

You are a software engineering expert who generate coding solutions for a coding problem based on some hints provided to solve the problem. You task is to read the problem description carefully and leverage the provided hints to generate an optimal coding solution for the problem. Use the above debugging strategy as additional context to help solve the problem more effectively. Make the solution concise. You need to import all necessary packages. Your solution should be in

```` ```python
YOUR IMPLEMENTATION HERE
``` ````

`<problem>` {user_problem} `</problem>`
`<hints>` {hints} `</hints>`