AUTOLLMRESEARCH: TRAINING AGENTS FOR AUTOMATING LLM EXPERIMENT CONFIGURATION — LEARN FROM CHEAP, OPTIMIZE EXPENSIVE

Anonymous authors
Paper under double-blind review

ABSTRACT

Effectively configuring scalable large language model (LLM) experiments (e.g., architecture design, hyperparameter tuning) is crucial, as poor configuration choices can waste substantial computational resources and prevent models from realizing their full potential. Prior automated methods are designed for low-cost settings where repeated trial and error is feasible, but scalable LLM experiments are too expensive for such extensive iteration. To our knowledge, no work has addressed the automation of high-cost LLM experiment configurations, leaving this problem labor-intensive and dependent on expert intuition. Motivated by this gap, we propose AutoLLMResearch, an agentic framework that mimics how human researchers learn generalizable principles from low-fidelity experiments and extrapolate to efficiently identify promising configurations in expensive LLM settings. The core challenge is how to enable an agent to learn, through interaction with a multi-fidelity experimental environment that captures the structure of the LLM configuration landscape. AutoLLMResearch addresses this challenge through two key components: 1) *LLMConfig-Gym*, a multi-fidelity environment encompassing four critical LLM experiment tasks, built from real, already-executed public and in-house experiment sweeps that cost an estimated one million GPU hours to produce; 2) *A structured training pipeline* that formulates configuration research as a multi-turn Markov Decision Process and accordingly applies *Train/Test Experiment Curation*, *Trajectory Simulation*, *Policy Distillation* and *Multi-turn Reinforcement Learning* to incentivize cross-fidelity extrapolation reasoning. Extensive evaluation against diverse strong baselines on held-out experiments shows that, by interacting with low-fidelity experiments in the Gym, a 4B agent internalizes configuration capabilities that prompting alone does not provide: it outperforms all meta-training and classic baselines, improves 4.7$\times$ over its prompt-only backbone, and matches frontier LLM agents given the same context, while offering interpretable cross-fidelity reasoning. Our code and data are available at this repository.

1 INTRODUCTION

As Large Language Models (LLMs) are deployed at increasingly diverse scales and settings, tuning them hinges on configuration decisions, including hyperparameters (Kaplan et al., 2020), architecture-related settings (Sukthanker et al., 2024), training recipes (Yang et al., 2021), and data-mixture design (Ye et al., 2025), which together shape model quality and efficiency; poor choices waste substantial compute and prevent models from realizing their full potential (Hoffmann et al., 2022; Halfon et al., 2024). Yet identifying effective configurations remains labor-intensive and expert-driven, especially as experiments scale up and become costly to rerun, leaving this problem practically important yet insufficiently studied. Recent Auto-research methods (Karpathy, 2026; OpenAI, 2025; Jiang et al., 2025) and existing optimizers (Akiba et al., 2019) aim to automate configuration tuning. However, they are designed for low-cost settings (classical ML such as DecisionTree, SVM, etc) where agents can propose configurations, execute them repeatedly, and iterate on prior outcomes. This paradigm fails for large-scale LLM experiments (e.g., $\geq$7B models or $\geq$20B training tokens), where a single training run consumes hundreds of GPU hours and only a few trials are feasible, so they cannot converge on good settings within realistic budgets.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

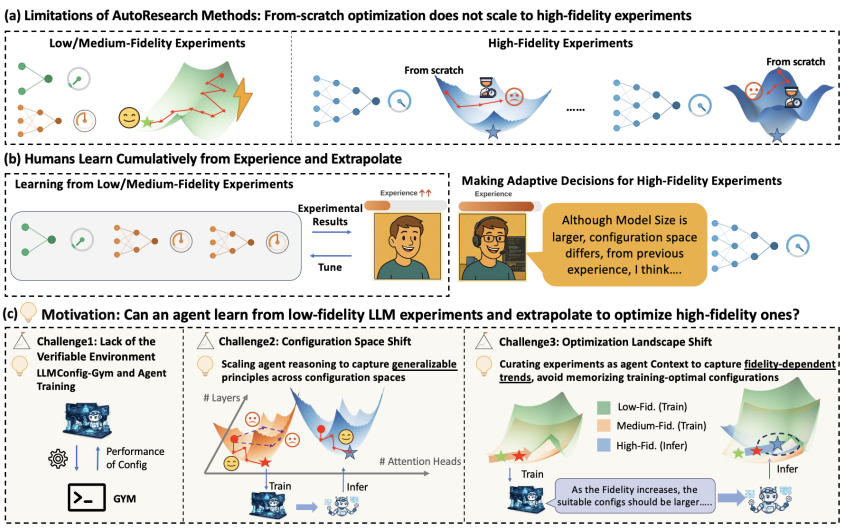


Figure 1: **Limitations of current methods, our motivation, and the three challenges** we address.

Table 1: **Comparison of approaches.** Columns C1–C3 correspond to the three challenges mentioned in Introduction. ✓ and × indicate whether a method is *designed for* that setting, not a measured capability. **AutoLLMResearch** targets cumulative experiential learning with a verifiable LLM experiment environment (LLMConfig-Gym) and both cross-fidelity shifts.

Category	Method	Target Experiments	Cumulative Experiential Learning	C1: Verifiable LLM Environment	C2: Space Shift	C3: Landscape Shift
Before LLM	Traditional HPO Tools	Low-Cost	×	×	×	×
	MetaBO (Volpp et al., 2020), NAP (Maraval et al., 2023) FSBO (Wistuba & Grabocka, 2021)	Low-Cost	✓	×	×	×
LLM-Based Prompting	LLM (GPT-5.6, Claude Fable 5, O4-mini, etc) (Liu et al., 2024)	Low-Cost	×	×	×	×
LLM-Based RL Training	**AutoLLMResearch (Ours)**	**High-Cost**	✓	✓	✓	✓

Motivated by this gap, we present, to our knowledge, the first systematic study on whether and how expensive LLM experiment configuration can be automated: under a fixed budget of costly trials, an agent decides the configurations that most shape an LLM training pipeline's outcome and cost the most to evaluate (architecture, pretraining hyperparameters, RL post-training hyperparameters, and data mixture). We draw inspiration from how human researchers optimize LLM experiments: developing generalizable principles from low-fidelity (low-cost) experiments and extrapolating them to high-fidelity (high-cost) settings. Our key insight is that LLM agents' text-based reasoning can perform cross-fidelity extrapolation, leading to our central question:

Can an agent learn from low-fidelity LLM experiments and extrapolate to optimize high-fidelity ones?

We further identify three challenges unique to this cross-fidelity scenario, illustrated in Fig. 1: 1) **Challenge 1: Lack of Verifiable Environment for LLM Experiments.** No existing environment provides verifiable multi-fidelity LLM experiment outcomes for agents to learn from experience. 2) **Challenge 2: Configuration Space Shift:** The configuration space differs between training (low-fidelity) and target (high-fidelity) experiments; the agent must reason across this shift and capture generalizable principles. 3) **Challenge 3: Optimization Landscape Shift:** Even within the same configuration space, the optimization landscape changes across fidelity levels, and optimal configurations do not necessarily transfer monotonically. No existing approach meets all three (Table 1): HPO tools (Akiba et al., 2019; Scikit-Optimize) and LLM-based auto-research methods optimize each experiment from scratch, while meta-training methods (Volpp et al., 2020; Maraval et al., 2023; Wistuba & Grabocka, 2021) do accumulate experience, but only at a single fidelity, encoding it as a fixed distribution over a fixed configuration space, a considerably easier setting.

To address them, we formulate configuration research as sequential decision making. At every step the agent observes curated lower-fidelity experiments and the history so far, and produces *reasoning* about fidelity-dependent trends followed by an *action* committing the next configuration. Its behavior is therefore characterized by the conditional policy $P(reason, act \mid obs)$. Such expertise is accumulated rather than recalled: it comes from having run and compared experiments. Prompting leaves the weights fixed, so past experiments can enter only through the context of each episode: the capability does not grow as experiments pile up, and nothing incentivizes the reasoning that transfers across

fidelities. We therefore optimize $P(\textit{reason}, \textit{act} \mid \textit{obs})$ **directly on** $\mathcal{D}_{\text{config}}$: LLMs provide a flexible interface for heterogeneous configuration spaces, while reinforcement learning against verifiable experimental outcomes turns past experiments into part of the policy itself (**cumulative experiential learning**) and lets it reason like a researcher rather than re-paying for it in context every episode.

Building on this idea, we propose **AutoLLMResearch**, an agentic framework with two key components: 1) **LLMConfig-Gym**, a multi-fidelity interactive environment spanning four critical LLM experiment tasks, which supplies the recorded outcomes of real, already-executed LLM experiments as verifiable rewards for each proposed configuration, supporting end-to-end multi-turn RL training. 2) **A structured training pipeline** that formulates configuration research as a multi-turn Markov Decision Process (MDP). It combines *Trajectory Simulation, Policy Distillation*, and *Multi-turn RL* to foster researcher-like extrapolation, e.g., from $\leq$3B-parameter, $\leq$8B-token experiments to 7B-parameter, 20B–100B-token ones. More broadly, training agents that extrapolate from cheap experiments to guide expensive ones relates to the long-term goal of AI assisting AI development (Good, 1966; Zhuge et al., 2026; Rank et al., 2026). In summary, our contributions are:

- We design LLMConfig-Gym and a training pipeline that let an agent conduct cumulative experiential learning by directly optimizing $P(\textit{reason}, \textit{act} \mid \textit{obs})$ via multi-turn RL, achieving cross-fidelity extrapolation.
- Experiments on the four LLMConfig-Gym tasks show that interacting with low-fidelity experiments internalizes optimization capability: our 4B agent beats all meta-training and classic baselines, cuts the regret of its prompt-only backbone by 4.7×, and reaches the level of frontier LLM agents (Claude Fable 5) at a fraction of their size, with its largest gains over them at early budgets; in-depth analysis further supports its generalization and interpretability.

2 Related Work

Methods for Automating LLM Experiment Configuration: Since no prior work learns a transferable prior from low-fidelity experiments for high-fidelity ones, we organize adjacent methods into: **1) Meta-learned optimizers:** MetaBO (Volpp et al., 2020), FSBO (Wistuba & Grabocka, 2021) and NAP (Maraval et al., 2023) learn a meta-probabilistic model over configurations from offline experiments; OptFormer (Chen et al., 2022) instead serializes tuning trajectories as text and trains a transformer to imitate optimizer behavior across heterogeneous search spaces. All learn from same-fidelity experience and assume many trials on the target task, so they cannot extrapolate across fidelity levels. **2) HPO tools and LLM-based AutoResearch methods:** Classical HPO tools, including multi-fidelity variants such as Hyperband (Li et al., 2018) and BOHB (Falkner et al., 2018), belong to a different paradigm: they search each target task from scratch online, so their low-fidelity evaluations are part of that search rather than experience accumulated across tasks, and the target fidelity must still be queried many times; they also optimize within a fixed configuration space and cannot handle the configuration-space shift. Recent LLM-based work (Liu et al., 2024; Zhang et al., 2023; Liu et al., 2025) uses LLMs as optimizers, proposing and refining configurations based on textual descriptions and past results. While LLM-based approaches handle diverse configuration spaces, both rely on on-policy methods assuming many experiment executions, which is infeasible when a single run costs hundreds of GPU hours. *Unlike all the above, we train a text-reasoning agent to learn such a prior and extrapolate it across fidelity levels.*

Incentivizing LLM Reasoning by RL with Verifiable Rewards: Reinforcement Learning with Verifiable Rewards (RLVR) has advanced the generalization and reasoning of frontier LLMs (OpenAI, 2024; Guo et al., 2025; Team et al., 2025). *We construct an RL Gym-style environment for LLM experiment configuration and leverage RLVR to make an LLM agent reason like a researcher, yielding sample-efficient, interpretable configuration decisions.*

3 LLMConfig-Gym: Environment for Training Agents

Here we introduce the design principles (tasks, organization, interface); the rest is in Appendix A.

Task Design and Hierarchical Organization: Surveying the key design choices in LLM experiments, we identify four representative tasks, each arising frequently in practice and covering a different part of LLM training; their optimization targets and configuration spaces are in Table 2. LLMConfig-

Table 2: **Task characteristics of four cross-fidelity LLM experiment configuration tasks.** Each task is categorized by its primary challenge: configuration space shift (Tasks 1 & 4) or optimization landscape shift (Tasks 2 & 3), along with the corresponding training and testing configuration spaces.

Task	Opt. Target	Primary Challenge	Training Space (Low/Med Fidelity)	Testing Space (High Fidelity)
Task 1: Model Architecture Design	Balance Perplexity & FLOPs	Configuration Space Shift	*GPT-M:* embed$\in\{256,512,1024\}$, layers$\in\{22,23,24\}$, per-layer: heads$\in\{8,12,16\}$, mlp_ratio$\in\{2,3,4\}$, bias$\in\{T,F\}$	*GPT-L:* embed$\in\{320,640,1280\}$, layers$\in\{34,35,36\}$, per-layer: heads$\in\{8,16,20\}$, mlp_ratio$\in\{2,3,4\}$, bias$\in\{T,F\}$
Task 2: Pretraining Hyperparameter	Loss ↓	Optimization Landscape Shift	Learning_Rate$\in\{2^{-12},\ldots,2^{-6.5}\}$, Batch_Size$\in\{32\text{K},\ldots,4\text{M}\}$ *Train:* training tokens$\leq$ 8B, model params 60M–537M	*Test:* training tokens$\geq$ 20B, model params 215M–537M
Task 3: RL GRPO Tuning	Reward ↑	Optimization Landscape Shift	Learning_Rate$\in\{10^{-6},5\times10^{-6},10^{-5}\}$, Batch_Size$\in\{16,32,64\}$, $\lambda_{\text{KL}}\in\{0,10^{-3}\}$ *Train:* MMLU subsets, 256 samples	*Test:* GSM8K/DAPO, 768–1536 samples
Task 4: Data Mixture for Instruction-Tuning	Test Performance ↑	Configuration Space Shift	*Different data mixture ratio space between training (Qwen2.5-3B) and testing (Qwen2.5-7B); details in Section A.6*	

Gym adopts a hierarchical structure: **Task** → **Fidelity** → **Experiment**. For each task, we collect experiments at multiple fidelities from open-source datasets such as HW-GPT-Bench (Sukthanker et al., 2024) or offline grid-search runs. Rather than imposing a rigid definition, we expose fidelity-related metadata (model size/training tokens/etc).

Unified Format, Interface and Sufficient Text-Based Context: Since LLM experiment runs are too costly for online interaction, we unify all tasks into an offline **Lookup Table**. The Gym does not simulate experiments: it replays outcomes recorded in advance, Tasks 2–4 from complete LLM training runs actually executed at the stated scales and Task 1 from HW-GPT-Bench's released evaluations (Sukthanker et al., 2024) (provenance in Table 4). The core API is a *tell* function returning a configuration's performance and experimental details **within seconds**. Each task is queried in the form that decision takes: Tasks 1 and 4 present a candidate list from which the agent selects, and on Tasks 2 and 3 the agent proposes coordinates on the grid of the target fidelity. We open-source it as a **broadly reusable** environment for training and evaluation (More in Section E). We further design per-task metadata (task and configuration descriptions) so the agent can interpret each problem.

4 Methodology of AutoLLMResearch

4.1 Problem Formulation: MDP for Agentic Training

We formulate LLM configuration as a sequential decision process. The **environment** is our LLMConfig-Gym, which provides a *tell* function that receives a configuration x and returns its performance y and experimental details; the **observation** is that return, as text. The **policy** $\pi_\theta(a_k \mid s_k)$ is parameterized by an LLM and optimized end-to-end via RL, with an **action space** $a_k \in \mathcal{A}$ of two text-based steps per trial. `Think`: given the context, the agent reasons step by step to identify a promising configuration, in tokens that stay inside the trajectory; `Execute`: it emits that configuration x_t as a tool call, the only part of the action the Gym receives, and observes its performance. The **state** is $s_t = (C, H_t, t, T)$, where C is the experiment's static context (task description, optimization direction, configuration space and candidate list, curated lower-fidelity demonstrations, and fidelity metadata), varying across experiments, $H_t = \{\langle x_1, y_1\rangle, \ldots, \langle x_{t-1}, y_{t-1}\rangle\}$ is the history up to step t, and T is the total budget; the **transition** appends the latest evaluation, $H_{t+1} = H_t \cup \{\langle x_t, y_t\rangle\}$, and increments t. The **objective** maximizes expected reward under a budget, $\arg\max_{\pi_\theta} \mathbb{E}\left[R\left(\{y_1, \ldots, y_T\}\right)\right]$, with each turn $s_{t-1} \xrightarrow{\texttt{Think}} x_t \xrightarrow{\texttt{Execute}} \texttt{GYM} \rightarrow y_t$ for $t = 1, \ldots, T$. The policy is thus the conditional capability of Section 1, $\pi_\theta(a_t \mid s_t) = P(\textit{reason}, \textit{act} \mid \textit{obs})$.

`Think` as Researcher-Level Reasoning. Optimizing π_θ directly shapes `Think`, letting the agent analyze concrete experiments with their fidelity information step by step. Prior meta-training methods instead model $p(x_i \mid s_k)$ as a categorical distribution over a fixed configuration set $\{x_1, \ldots, x_n\}$ and merely select the $\arg\max$; our text-based `Think` internalizes the reasoning steps that lead to better configurations rather than a fixed distribution.

4.2 Step1: Train/Test Experiment Curation

This step builds training and testing samples from Gym experiments addressing Challenge 1.

Addressing Challenge 2 via Rich Text Information throughout Input and Rollout. Because configuration spaces shift across fidelities, the agent has to reason from principles rather than from remembered configurations. We therefore *build rich textual context, so the LLM's pretrained domain knowledge can be brought to bear on the problem.* In the input frame of Fig. 2, the agent's input has three modules: **Task** (description and optimization target); **Context** (fidelity information, in-context

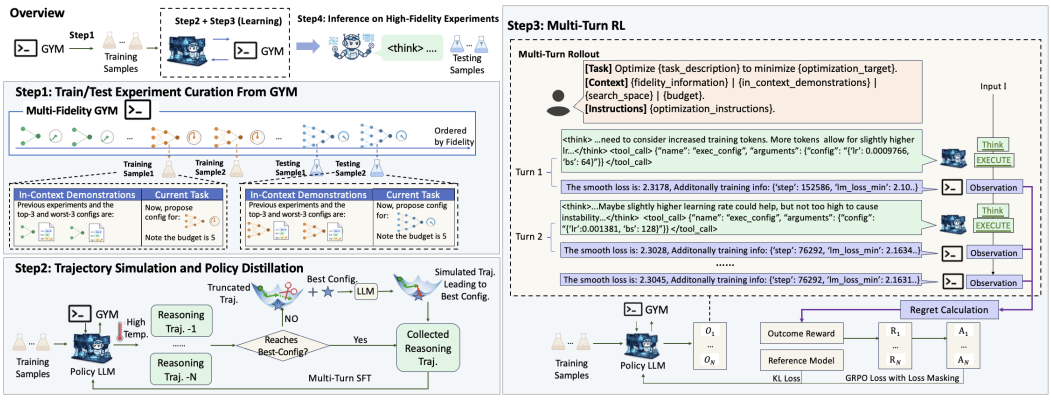


Figure 2: **Overview of our framework**. Step 1 curates multi-fidelity training and testing experiments with in-context demonstrations. Step 2 collects successful reasoning trajectories via high-temperature sampling for policy distillation. Step 3 further optimizes the policy with multi-turn RL. Step 4 deploys the trained policy on various unseen high-fidelity experiments.

demonstrations, configuration space, budget); **Instructions**. Including budget lets the agent adapt its exploration–exploitation trade-off to the remaining turns. We also enrich rollout feedback: after each Gym interaction, the agent receives target performance and task-specific experiment details (e.g., critic scores in RL tuning). This grounding makes overlapping cross-fidelity patterns more visible and amplifies low-to-high fidelity transfer.

Addressing Challenge 3 via Lower-Fidelity Experiments as In-Context Demonstrations. To extrapolate across fidelities rather than overfit to training-time optima, the agent must capture configuration *trends*. We therefore *give it lower-fidelity results and instruct it to reason about how configurations should change as fidelity increases*. We order experiments by fidelity (e.g., model size, dataset size) and split them into low-/medium-/high-fidelity sets L, M, H, then construct one-to-many pairs: for training, each $m_i \in M$ is paired with all L as in-context demonstrations; for testing, each $h_i \in H$ is paired with M. For each pair, Top-K configurations from the lower-fidelity side are concatenated with fidelity information in the prompt. Unlike prior meta-training methods that treat each experiment as an independent sample (m_i or h_i alone), *associating experiments across fidelities puts the agent in a cross-fidelity transfer setting, encouraging transferable reasoning*.

4.3 Step2: Trajectory Simulation and Policy Distillation

Training directly on LLMConfig-Gym exposes two drawbacks: without explicit supervision toward the best configuration the agent settles into local optima, and across multi-turn reasoning and environment interaction the base agent forgets instructions and produces format errors. For each curated training sample, we augment with different budget settings, then sample 20 rollout trajectories at temperature 0.8. Trajectories that reach the best configuration are added to a trajectory set PD. For samples whose trajectories all fail (i.e., stuck at local minima), we apply **Trajectory Simulation**: take the trajectory with the local-best configuration, randomly truncate the last or second-to-last trial, and prompt the LLM with 1) the truncated trajectory, 2) the best configuration, and 3) instructions to continue toward the best configuration. The truncated prefix and newly generated suffix are concatenated and added to PD. Finally, we perform **Policy Distillation** on the base LLM via multi-turn SFT on PD, applying loss masking on Gym observations and instruction tokens so the agent learns *how to reason and interact with the environment over long horizons to reach the best configuration.* Note that Trajectory Simulation uses best configurations of **training** experiments only; test experiments are never involved. These come from the same completed training sweeps on which the meta-training baselines are trained (Section B.4).

4.4 Step3: End-to-End Multi-Turn Reinforcement Learning

After Policy Distillation, we apply Multi-Turn RL via GRPO (Shao et al., 2024). For each configuration task we sample G trajectories $\{O_i\}_{i=1}^{G}$, where $O_i = (I, a_1, \hat{y}_1, \text{obs}_1, \ldots, a_n, \hat{y}_n)$. Each trajectory receives a scalar reward r_i, from which we compute a group-normalized advantage shared by all its tokens. We then optimize the standard clipped GRPO objective with a KL penalty, applying

loss masking to experiment observations and instruction tokens so the agent learns only `Think` and `Execute`; the full objective is in Appendix B.5.

Regret-Based Outcome Reward. To teach extrapolative reasoning we design **cumulative regret**, which scores behavior across all T turns rather than only the best-found configuration. Raw metrics differ in direction across tasks, so we map each to a utility y under a single *larger-is-better* convention, negating the minimized ones (perplexity, FLOPs and training loss in Tasks 1 and 2); this convention is used throughout the paper. Given utilities $\{y_1, \ldots, y_T\}$, we normalize the gap between the cumulative utility $\sum_{t=1}^{T} y_t$ and the upper bound $T \cdot y_{\text{best}}$ by the worst-case range: $R_{\text{outcome}} = -\big(T \cdot y_{\text{best}} - \sum_{t=1}^{T} y_t\big) \big/ \big(T \cdot y_{\text{best}} - T \cdot y_{\text{worst}}\big)$ if the agent proposes T distinct valid configurations, and $R_{\text{outcome}} = -1$ otherwise (on repeats or invalid outputs), where y_{best} and y_{worst} are the best and worst utilities in the lookup table for that training experiment. This design has two benefits: 1) summing over T distinct trials rewards consistently strong proposals rather than a single lucky one, reducing overfitting in cross-fidelity transfer, while the metric reported in Section 5 scores the best configuration found within the budget; 2) the worst reward (-1) on repeated or invalid outputs explicitly penalizes trivial replay and format errors.

Make Reward Dense via Most-Similar Configuration Matching. With the above reward, some tasks assign -1 to most trajectories, resulting in a sparse learning signal. The root cause is **format violations in long reasoning chains**: despite explicit Gym-call instructions, the agent occasionally produces minor format errors in later turns, especially for long configurations, and each such error marks an entire rollout trajectory invalid. To mitigate this, we add **most-similar configuration matching** during Gym queries: if the generated configuration lies in valid space, we use it directly; otherwise, we query the valid configuration with the longest common substring (Python Software Foundation, 2026). This substantially reduces sparse-reward cases and stabilizes training. Matching is part of the Gym protocol at both training and test time, applied identically to all LLM-based agents; Table 18 quantifies how often invalid outputs occur.

5 Experiments

We address three research questions (RQs): **RQ1 (Effectiveness and Generalization):** Does training agents improve performance over classic, meta-training, and LLM-based baselines? Does RL training strengthen $P(\textit{reason}, \textit{act} \mid \textit{obs})$ beyond frontier-scale prompting, and how well does it generalize across scenarios? **RQ2 (Interpretability):** What transferable reasoning principles do agents learn from low-fidelity experiments, and how do they benefit high-fidelity ones? **RQ3 (Training Dynamics):** How do our training strategies stabilize agent learning?

Benchmarks and Baselines. We evaluate on all four LLMConfig-Gym tasks (Table 2). For each task, the agent is trained on low-/medium-fidelity experiments and tested on high-fidelity ones. We adopt a **budget-constrained** protocol with budgets from 1 to 5 per task. No prior method targets all four tasks, so we include every method that can decide from the low-fidelity training experiments plus at most $T \leq 5$ high-fidelity evaluations; this excludes online multi-fidelity HPO (Hyperband, BOHB, ADMIRE-BayesOpt (Ouyang et al., 2025)), which fits each new task with many target-fidelity queries, closed-form scaling laws for Task 2 (Step Law (Li et al., 2025)), which are fitted on the sweep our test experiments are drawn from, and OptFormer (Chen et al., 2022), for which no pretrained model is public. We adopt baselines from three categories: **(1) Normal Baselines:** *Random Search* (uniform sampling) and *Top-K Warm-Start (WS)* (reusing the top-K training configurations). **(2) Meta-Training Methods:** NAP, MetaBO, and FSBO, which perform end-to-end meta-optimization offline on the same low-/medium-fidelity training experiments as our agent, not expected to extrapolate across fidelities. **(3) LLM-Based Methods:** Prompt-only agents given **exactly the same curated Top-K low-fidelity context, budget, observation history, and protocol as our agent** but without training: frontier reasoning models (Claude Fable 5, Claude Opus 5, GPT-5.6 Sol, O4-mini (OpenAI, 2025)) and, as open references, our untrained Qwen3-4B-Base backbone and the larger Qwen3-14B-Base. This isolates the value of training from the value of context and from model scale (Section C.4 further ablates the context itself). Implementation details of every baseline (model versions, decoding, parsing and repair rules, how the meta-training methods are adapted to each task, and the number of runs) are in Section B.4.

Implementation Details. 1) LLMConfig-Gym: contains 4 representative LLM configuration tasks, built from real, already-executed public and in-house experiment sweeps costing an estimated $\sim$1M GPU hours, multiple fidelity levels, and a unified sub-second querying API. Provenance and

cost per task are in Table 4; details in Appendix A. **2) Training Agent:** We use Qwen3-1.7B and Qwen3-4B (Yang et al., 2025) as backbones, trained on a single node with 4 NVIDIA A100 GPUs (80GB): Policy Distillation via LlamaFactory (Zheng et al., 2024b) and end-to-end Multi-Turn GRPO via Verl (Sheng et al., 2025), with the agent–Gym interaction implemented as a tool call (`exec_config`). Full training stack, prompts, and hyperparameter settings are in Appendix B.

Evaluation Metrics. We use normalized regret as a unified metric across tasks: Regret $= \min_{h \in \mathcal{H}_T} \frac{y^*_{\text{best}} - f(h)}{y^*_{\text{best}} - y^*_{\text{worst}}}$, where $\mathcal{H}_T$ is the set of configurations proposed under budget T, $f(h)$ is its utility under the larger-is-better convention of Section 4.4, and y^*_{best}, y^*_{worst} are the best and worst utilities over *all valid configurations* of the test experiment's lookup table, which LLMConfig-Gym provides directly. The Gym returns the outcome already recorded for the exact configuration proposed, so every regret is a difference between recorded experiment outcomes. Regret is therefore a fixed property of the environment, independent of which methods are compared; regret 0 means the global optimum was found, and lower is better. Every method is evaluated under the same protocol: at budget T it reports the best configuration among its first T proposals, with no best-so-far smoothing across budgets, and a proposal outside the test experiment's configuration space is handled identically for every method (Section B.4). Raw task metrics (loss, accuracy, perplexity, FLOPs) of the configurations found are in Section C.3.

5.1 RQ1: Effectiveness and Generalization

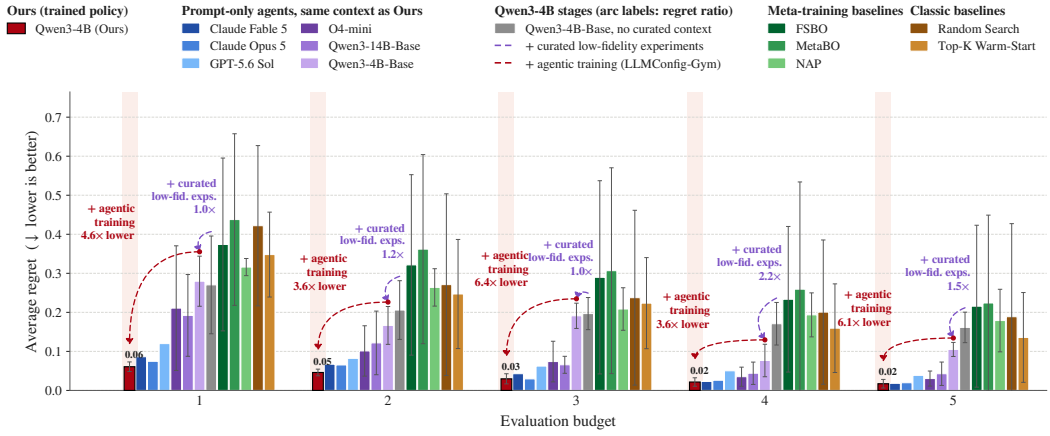


Figure 3: **Overall performance (average regret over the four tasks) under budgets 1–5**. Our 4B agent has the lowest regret at every budget. The two dashed arcs decompose the Qwen3-4B staircase: from the prompt-only backbone without any low-fidelity context (grey) to the same backbone given the curated low-fidelity experiments in context (purple), and from there to our trained agent (red); arc labels are the regret ratio at each budget. Error bars: std over 3 runs in regret units, consistently smallest for our agent (Claude Fable 5, Claude Opus 5 and GPT-5.6 Sol are evaluated once because of API cost; O4-mini three times); per-task and per-budget values are in Section C.2.

Overall Performance. Per-task results are in Section C.1. **Result 1:** Our 4B agent attains the lowest regret at every budget and beats all meta-training and classic baselines by a wide margin. Against prompt-only agents given exactly the same curated context, training reduces the regret of its own Qwen3-4B backbone from 0.164 to 0.035 (4.7×) and brings a 4B open model to the level of frontier LLM agents at a small fraction of their size. The two arcs in Fig. 3 locate that gain: the curated context alone improves the prompt-only backbone only modestly, because the untrained model often fails to *act* in the experiment loop at all, and training removes this behavioral bottleneck before it improves the decisions themselves (Section C.4 and Fig. 12).

Decomposing Policy Performance into Extrapolation and Refinement. At budget 1, the interaction history is empty, so performance isolates ***first-proposal extrapolation*** from the curated low-fidelity context alone. From budget 2 onward, previous high-fidelity outcomes enter the history H_t, progressively enabling ***feedback-driven refinement***. **Result 2:** RL provides its largest advantage when high-fidelity feedback is absent or scarce. **Extrapolation:** at budgets 1–2 our 4B agent achieves 19% and 29% lower regret than the best frontier agent, an advantage the ablation attributes to RL rather than to distillation alone (Table 3). **Refinement:** our agent keeps the lowest regret at every budget, and the strongest frontier agents close the gap to within 0.003 on average only at budgets 3–5,

after three to five expensive high-fidelity trials have been spent. RL training on $\mathcal{D}_{\text{config}}$ supplies the extrapolation that precedes such evidence, the capability a tight budget makes decisive.

Performance across Different Low-to-High Fidelity Scenarios: Every test experiment is unseen during training, and what changes from training to testing is the quantity that dominates cost: GPT-M to GPT-L architectures (Task 1), 2–8B to 20–100B tokens (Task 2), MMLU to GSM8K/DAPO data (Task 3), 3B to 7B target models (Task 4). Tasks 1 and 4 additionally use disjoint train and test configuration spaces (Table 9). **(1) Addressing Challenge 2: Configuration-Space Shift.** Results in Fig. 13 and 16 show that our Qwen3-4B agent reaches near-zero regret within two to three budgets, while meta-training and classic baselines stay far higher until budget 3. The strongest frontier agents match our agent on Task 1, but on the high-dimensional Task 4 our agent cuts the regret of the best frontier agent by roughly half or more at every budget, where the curated context alone is insufficient. **(2) Addressing Challenge 3: Optimization-Landscape Shift.** Here the optimal region itself moves between fidelities. Fig. 14 and 15 show our method clearly outperforms meta-training and classic baselines at every budget; the closest, Top-K WS, still has at least 2.5× our regret at budgets 4–5. The strongest frontier agents are competitive here: on Task 3, they and our agents all reach regret below 0.04, and on Task 2 Claude Fable 5 and Opus 5 attain lower regret than our 4B agent. **Result 3:** Text-based reasoning transfers across both shift types: the agent generalizes where the configurations themselves cannot be reused, and the one place a frontier agent keeps an edge is Task 2.

Ablation Studies. **Result 4:** Both experiment context and the training pipeline are necessary. As shown in Table 3, every component is essential. **(a) Experiment Context Information.** Relative to Qwen3-4B-Base, removing either **Train/Test Experiment Curation** (asking the agent to optimize each experiment individually as in prior meta-training) or **Rich Text Information** from the prompt raises regret, confirming that curation enables cross-fidelity extrapolation with a strong starting point and rich text grounds the agent's reasoning in task-specific knowledge. The same curated context also lowers the regret of untrained prompt-only models by 16–45% relative (Section C.4). **(b) Training Pipeline.** Policy Distillation alone teaches the reasoning process but does not directly optimize regret and stays clearly behind the best frontier agent; Multi-Turn RL alone yields higher regret because under long reasoning chains the agent loses track or converges to local optima without distillation supervision. Combining both halves the regret of distillation alone, with the largest gain at budget 1 (0.162→0.061): distillation anchors instruction following over long horizons, while regret-driven RL pushes the agent toward lower-regret configurations.

Table 3: **Ablation studies** on different components of our method.

Method	Avg. Regret ↓
Ablations Experiment Context on Base Model	
w/o Train/Test Experiment Curation	0.201
w/o Rich Text Information	0.202
Qwen3-4B-Base	0.164
Ablations Training Pipeline	
+ Policy Distillation	0.076
+ Multi-Turn RL	0.121
+ Policy Distillation + Multi-Turn RL	**0.035**

Amortized Cost Analysis under Scalable Deployment. Because our agent accumulates experience across experiments instead of optimizing each task from scratch, its upfront investment is amortized. Appendix C.7 compares the total compute needed to reach the same target performance: methods that restart on every experiment accumulate cost linearly, whereas ours pays once and then less per experiment (Fig. 20). **Result 5:** Training once on cheaper experiments and reusing the policy reduces total compute by 2.8× over 20 high-fidelity experiments, with the savings increasing as deployment scales.

5.2 RQ2: Interpretations of AutoLLMResearch Agent Capabilities

Reasoning Analysis: What Transfers from Low-to-High Fidelity Experiments. Beyond quantitative performance, *what does the agent learn that enables low-to-high fidelity extrapolation?* We trace the agent's text-based reasoning trajectories on **both the training and test sets** across three stages (**Before Training**, **During Training**, **Inference**). **Case Study 1** (Fig. 18, Appendix C.5) examines Challenge 2 (Configuration Space Shift) on Model Architecture Configuration; **Case Study 2** (Fig. 19, Appendix C.5) examines Challenge 3 (Optimization Landscape Shift) in Pretraining Hyperparameter Configuration. **Result 6:** Across both case studies, the learned reasoning is consistent with transferable principle learning rather than configuration memorization: in Case Study 1 the agent applies a structural balancing rule that generalizes across shifted configuration spaces, and in Case Study 2 it identifies fidelity-dependent trends that remain useful after the optimization landscape shifts. We present these traces as illustrative examples; the optimization analysis of Fig. 4 provides complementary behavioral evidence, and Section C.9 varies the low-fidelity evidence directly to measure how far the agent's choices move with it.

Optimization Analysis: Text-Based Reasoning Prunes the Search Space. We compare three variants on the same task, and visualize their optimization trajectories and effective search region across repeated runs (setup in Appendix C.6). As shown in Fig. 4, three distinct behaviors emerge: **(1) Without Experiment Curation**, the agent explores erratically over the largest search region. **(2) The base model without training** recognizes low-fidelity trends but extrapolates them in the wrong direction, drifting toward overly conservative configurations. **(3) Our trained agent** correctly extrapolates from low-fidelity experience, lands near the global optimum from the first round, and concentrates subsequent search tightly around it. **Result 7:** Learned text-based reasoning acts as a search prior, providing a strong initialization and guiding optimization in the correct direction to substantially narrow the effective search space.

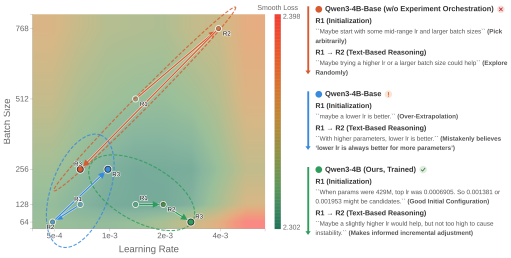


Figure 4: **Optimization trajectories on Task 2, over the ground-truth loss landscape (darker green = lower loss).** Arrows trace each method's Round 1→2→3; ellipses mark its *search region*.

Boundary of Extrapolation. Under an adversarial regime in which the optimum of the training experiments is *reversed* relative to the test experiments, our agent stays below the strongest meta-training baseline at every level and recovers as soon as experiments with intermediate optima are added, whereas NAP keeps re-proposing the reversed training optimum (Section C.9).

5.3 RQ3: Training Dynamics

Training Dynamics Across Tasks. We partition training samples by task type and track **Regret Mean@3**, **Critic Score Mean**, **Critic Score Min**, and **Response Length Mean** throughout training (metric definitions and analysis in Appendix C.8 and Fig. 21). **Result 8:** Training progressively improves configuration quality and interaction reliability while producing task-adaptive reasoning behavior. Execution and unique-configuration rates across the training stages are in Table 18.

Reward Densification via Most-Similar Configuration Matching. We compare the reward distributions of the RL training rollouts with and without the matching rule of Section 4.4. As shown in Fig. 5, without matching, 32% of the rollouts collapse to -1 because the generated configuration cannot be directly queried in the Gym; matching recovers them, producing a denser and more informative training signal.

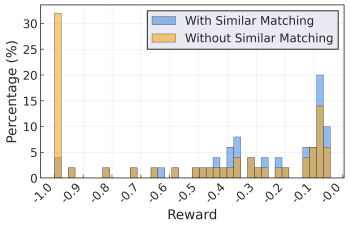


Figure 5: Reward distribution of the RL training rollouts, with and without Configuration Matching.

5.4 Scope of the Evaluation

The four tasks vary fidelity along the axis that dominates the cost of each decision (Table 2): the high-fidelity targets are GPT-L architectures of up to 0.77B parameters (Task 1), pretraining runs of 215M–537M parameters on 20B–100B tokens (Task 2), GRPO tuning of a 3B model on 768–1,536 samples for 30 epochs (Task 3), and instruction tuning of a 7B model (Task 4). Because every entry is an outcome recorded in advance, the protocol is offline and verifiable, and in every task the configuration space the agent decides over shifts between training and testing (Section 3). What we establish is therefore the quality of that decision against recorded outcomes; executing a selected configuration in a live pipeline, and proposing configurations beyond a recorded space, are the natural next steps (Section D).

6 Conclusion

We tackle automating high-cost LLM experiment configuration under strict budget constraints, a setting prior low-cost trial-and-error methods do not target. We propose **AutoLLMResearch**, an agentic framework that learns generalizable principles from low-fidelity experiments and extrapolates them to expensive high-fidelity ones, supported by *LLMConfig-Gym* and a *structured training pipeline*. Our evaluation spans architecture, pretraining, RL post-training, and data decisions at scales where a single run costs hundreds of GPU hours, with training experiments of at most 8B tokens and high-fidelity targets of 20B to 100B (Table 2). This is well beyond the small tabular or synthetic problems on which meta-optimization and LLM-as-optimizer methods are typically evaluated. In this setting, a 4B agent trained in the Gym reaches the level of frontier LLM agents given the same context, generalizes across fidelity shifts, and offers interpretable evidence of cross-fidelity reasoning.

THE USE OF LARGE LANGUAGE MODELS

In accordance with the ICLR 2027 AI policy, we disclose the use of large language models (LLMs) in the preparation of this work. Beyond the roles described below, LLMs are also the *subject* of this research: the trained research agents and the experiment configuration tasks studied in this paper are themselves LLM-based, and their use is fully described in the methodology and experiment sections. The disclosures here concern AI assistance to the authors during the research and writing process.

Writing assistance. We used LLM-based tools to aid and polish the writing of this paper, including grammar correction, improving the clarity and readability of author-written text, and suggesting alternative phrasings. All content was originally drafted by the authors, and every AI-suggested edit was reviewed, verified, and approved by the authors before inclusion. No scientific claims, results, or conclusions were generated by AI.

Retrieval and discovery. We used LLM-based search and retrieval tools to assist in discovering related work and identifying relevant literature. All references surfaced in this manner were manually verified by the authors against the original publications to confirm their existence, correctness, and relevance before being cited.

Research ideation and execution. We used LLMs as a brainstorming aid during research ideation, such as refining hypotheses and discussing experimental design choices, and as a coding assistant during research execution, such as helping implement utility code for experiment orchestration, data processing, and analysis scripts. The core research ideas, the framework design, all experimental decisions, and all conclusions are the authors' own. All AI-assisted code was reviewed and tested by the authors, and all experimental results reported in this paper were produced and verified by the authors.

Responsibility statement. The authors take full responsibility for the entire content of this paper. We have verified all statements, references, and results, and confirm to the best of our knowledge that the paper contains no unverified AI-generated claims, fabricated references, or misrepresented results.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pp. 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330701. URL https://doi.org/10.1145/3292500.3330701.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '22. IEEE Press, 2022. ISBN 9784665454445.

Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Richard Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc' Aurelio Ranzato, Sagi Perel, and Nando de Freitas. Towards learning universal hyperparameter optimizers with transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 32053–32068. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/cf6501108fced72ee5c47e2151c4e153-Paper-Conference.pdf.

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, 2021. URL https://arxiv.org/abs/2110.14168.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pp. 1437–1446. PMLR, 2018.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027.

Irving John Good. Speculations concerning the first ultraintelligent machine. volume 6 of *Advances in Computers*, pp. 31–88. Elsevier, 1966. doi: https://doi.org/10.1016/S0065-2458(08)60418-0. URL https://www.sciencedirect.com/science/article/pii/S0065245808604180.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Rushil Gupta, Jason Hartford, and Bang Liu. LLMs for Bayesian optimization in scientific domains: Are we there yet? In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 15482–15510, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.838. URL https://aclanthology.org/2025.findings-emnlp.838/.

Alon Halfon, Shai Gretz, Ofir Arviv, Artem Spector, Orith Toledo-Ronen, Yoav Katz, Liat Ein-Dor, Michal Shmueli-Scheuer, and Noam Slonim. Stay tuned: An empirical study of the impact of hyperparameters on llm tuning in real-world applications, 2024. URL https://arxiv.org/abs/2407.18990.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/c1e2faff6f588870935f114ebe04a3e5-Paper-Conference.pdf.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code, 2025. URL https://arxiv.org/abs/2502.13138.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Andrej Karpathy. autoresearch. https://github.com/karpathy/autoresearch, 2026.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=i1uGbfHHpH.

Houyi Li, Wenzhen Zheng, Qiufeng Wang, Hanshan Zhang, Zili Wang, Shijie Xuyang, Yuantao Fan, Zhenyu Ding, Haoying Wang, Ning Ding, Shuigeng Zhou, Xiangyu Zhang, and Daxin Jiang. Predictable scale: Part i, step law – optimal hyperparameter scaling law in large language model pretraining, 2025. URL https://arxiv.org/abs/2503.04715.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of machine learning research*, 18(185):1–52, 2018.

Siyi Liu, Chen Gao, and Yong Li. Agenthpo: Large language model agent for hyper-parameter optimization. In Beidi Chen, Shijia Liu, Mert Pilanci, Weijie Su, Jeremias Sulam, Yuxiang Wang, and Zhihui Zhu (eds.), *Conference on Parsimony and Learning*, volume 280 of *Proceedings of Machine Learning Research*, pp. 1146–1169. PMLR, 24–27 Mar 2025. URL https://proceedings.mlr.press/v280/liu25c.html.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=OOxotBmGol.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery, 2024. URL https://arxiv.org/abs/2408.06292.

Alexandre Maraval, Matthieu Zimmer, Antoine Grosnit, and Haitham Bou Ammar. End-to-end meta-bayesian optimisation with transformer neural processes. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 11246–11260. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/2561721d0ca69bab22b749cfc4f48f6c-Paper-Conference.pdf.

OpenAI. Learning to reason with LLMs. 2024. URL https://openai.com/index/learning-to-reason-with-llms.

OpenAI. Openai o3 and o4-mini system card. https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf, 2025.

OpenAI. Introducing deep research. https://openai.com/index/introducing-deep-research/, 2025.

Xu Ouyang, Shengzhuang Chen, Michael Arthur Leopold Pearce, Thomas Hartvigsen, and Jonathan Richard Schwarz. ADMIRE-bayesopt: Accelerated data MIxture RE-weighting for language models with bayesian optimization. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=0Euvm9zDpu.

Python Software Foundation. difflib — Helpers for computing deltas. 2026. URL https://docs.python.org/3/library/difflib.html.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. Posttrainbench: Can llm agents automate llm post-training?, 2026. URL https://arxiv.org/abs/2603.08640.

Juergen Schmidhuber. Goedel machines: Self-referential universal problem solvers making provably optimal self-improvements, 2006. URL https://arxiv.org/abs/cs/0309048.

Scikit-Optimize. Scikit-optimize: Sequential model-based optimization in python. URL https://scikit-optimize.github.io/.

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 1279–1297, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Rhea Sanjay Sukthanker, Arber Zela, Benedikt Staffler, Aaron Klein, Lennart Purucker, Jörg K. H. Franke, and Frank Hutter. Hw-gpt-bench: Hardware-aware architecture benchmark for language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 60776–60834. Curran Associates, Inc., 2024. doi: 10.52202/079017-1944. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/6ffdbf064df51857eb802a904aaaba63-Paper-Datasets_and_Benchmarks_Track.pdf.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, et al. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.

Gary Tom, Stefan P. Schmid, Sterling G. Baird, Yang Cao, Kourosh Darvish, Han Hao, Stanley Lo, Sergio Pablo-García, Ella M. Rajaonson, Marta Skreta, Naruki Yoshikawa, Samantha Corapi, Gun Deniz Akkoc, Felix Strieth-Kalthoff, Martin Seifrid, and Alán Aspuru-Guzik. Self-driving laboratories for chemistry and materials science. *Chemical Reviews*, 124(16):9633–9732, 2024. doi: 10.1021/acs.chemrev.4c00055. URL https://doi.org/10.1021/acs.chemrev.4c00055. PMID: 39137296.

Michael Volpp, Lukas P. Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. Meta-learning acquisition functions for transfer learning in bayesian optimization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryeYpJSKwr.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=y10DM6R2r3.

Martin Wistuba and Josif Grabocka. Few-shot bayesian optimization with deep kernel surrogates. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=bJxgv5C3sYc.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 17084–17097. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/8df7c2e3c3c3be098ef7b382bd2c37ba-Paper.pdf.

Jiasheng Ye, Peiju Liu, Tianxiang Sun, Jun Zhan, Yunhua Zhou, and Xipeng Qiu. Data mixing laws: Optimizing data mixtures by predicting language modeling performance. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=jjCB27TMK3.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Michael Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. Using large language models for hyperparameter optimization. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023. URL https://openreview.net/forum?id=FUdZ6HEOre.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024a. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In Yixin Cao, Yang Feng, and Deyi Xiong (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 400–410, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-demos.38. URL https://aclanthology.org/2024.acl-demos.38/.

Mingchen Zhuge, Ailing Zeng, Deyao Zhu, Sherry Yang, Vikas Chandra, and Jürgen Schmidhuber. AI with recursive self-improvement. In *ICLR 2026 Workshop Proposals*, 2026. URL https://openreview.net/forum?id=OsPQ6zTQXV.

APPENDIX

APPENDIX CONTENTS

A LLMConfig-Gym: Tasks and Dataset Construction **15**

A.1 LLMConfig-Gym Overview 16
A.2 Unified Interface of LLMConfig-Gym 16
A.3 Task 1: Model Architecture Configuration 17
A.4 Task 2: Pretraining Hyperparameter Configuration 18
A.5 Task 3: RL GRPO Tuning Configuration 18
A.6 Task 4: Data Mixture Configuration 19
A.7 Task Split on LLMConfig-Gym in Our Paper 20
A.8 Multi-Fidelity Optimization Landscape Analysis 20

B Implementation Details **24**

B.1 Prompts and Meta-Features 24
B.2 Interactions between Agent and LLMConfig-Gym as Function Calling 26
B.3 Hyperparameter Settings For Training 26
B.4 Baseline Implementation Details and Evaluation Protocol 28
B.5 Multi-Turn GRPO Objective 29

C Additional Experimental Results **30**

C.1 RQ1: Performance for Different Tasks 30
C.2 RQ1: Numeric Results and Run-to-Run Variability 32
C.3 RQ1: Raw Task Metrics of the Best Configurations Found 34
C.4 RQ1/RQ2: Effect of the Curated Low-Fidelity Context on Prompt-Only Models . . 34
C.5 RQ2-1: Full Case Study Details (Configuration Space Shift and Optimization Landscape Shift) 35
C.6 RQ2-2: From Optimization Perspective: Text-Based Reasoning Prunes Search Space 36
C.7 Amortized Cost Analysis under Scalable Deployment 39
C.8 RQ3: Training Dynamics, Metric Definitions, and Full Analysis 40
C.9 Stress-Testing the Boundary of Low-to-High Extrapolation 41

D Broader Impact and Future Work **43**

E Reproducibility and Release **43**

A LLMCONFIG-GYM: TASKS AND DATASET CONSTRUCTION

LLMConfig-Gym aggregates public experiment sweeps and in-house runs into a unified offline environment spanning four representative LLM configuration tasks. Table 4 lists, per task, the source of the experiments, how many of them the Gym exposes, and the compute their collection cost as reported by the source; the public sweeps were run by their respective authors and only the Task 3

runs were executed by us. Summing the reported collection costs gives roughly one million GPU hours, dominated by the Step Law pre-training sweep (Li et al., 2025); counted proportionally by the number of runs the Gym actually exposes, the figure is about 0.35M GPU hours. Task 1 is excluded from both sums because its values come from HW-GPT-Bench's openly released data rather than from runs executed for this work.

Table 4: **Provenance and collection cost of the experiments in LLMConfig-Gym.** "Runs in Gym" counts the experiments the Gym exposes; "Collection cost" is the compute reported by the source for the sweep the runs are drawn from (H800 GPU hours for Step Law, GPU hours as reported for ADMIRE, A100 GPU hours for our GRPO runs). Task 1 values come from HW-GPT-Bench's openly released data, so no training cost is attributed to them.

Task	Source	Runs in Gym	Collection cost (reported)
Task 1: Model Architecture	HW-GPT-Bench (Sukthanker et al., 2024)	600 architectures (20 per experiment, GPT-S/M/L)	not counted (open released data)
Task 2: Pretraining Hyperparameter	Step Law (Li et al., 2025)	1,236 of ∼3,700 runs, covering all 7 model sizes N and 7 data sizes D	∼1M H800 GPU hours for the full sweep (∼0.33M prorated by runs)
Task 3: RL GRPO Tuning	in-house (this paper)	396 GRPO runs over 22 experiments	∼4,000 A100 GPU hours (4 nodes × 4 A100 80GB)
Task 4: Data Mixture	ADMIRE IFT Runs (Ouyang et al., 2025)	460 runs (Qwen2.5-0.5B/3B/7B, full dataset)	>13,000 GPU hours

A.1 LLMConfig-Gym Overview

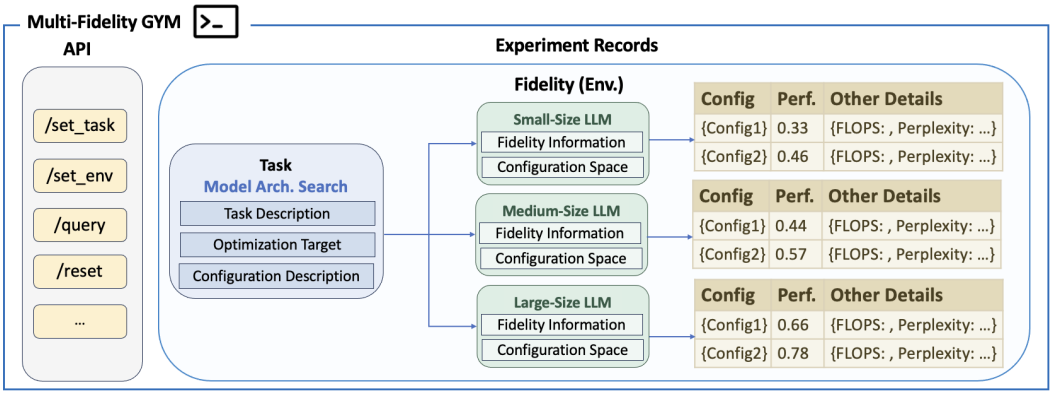


Figure 6: **LLMConfig-Gym overview.** A unified, lookup-table-based Gym organized by **Task** → **Fidelity** → **Experiment**.

A.2 Unified Interface of LLMConfig-Gym

LLMConfig-Gym is implemented as a lookup-table-based environment for fast, deterministic evaluation of configuration policies (Fig. 6). All four tasks are exposed through a unified API; a simplified code snippet is shown in Fig. 7. The core functions are:

- `list_tasks`: List all available configuration tasks (architecture search, data mixture, pretraining hyperparameters, RL hyperparameters).
- `set_task`: Select the active configuration task.
- `show_envs`: List available environments for the current task.
- `set_env`: Fix a concrete environment (e.g., dataset, model, training size).
- `show_configuration_space`: Return the configuration space for the current task and environment.

```python
from lmconfig_gym import LMConfigGym

# 1. Initialize and select RL tuning task
env = LMConfigGym()
print("Available tasks:", env.list_tasks())
# Available tasks: ['rl_grpo_tuning',
    'model_architecture_configuration',
    'pre-training_hyperparameter_tuning',
    'training_data_mixing_configuration']
env.set_task("rl_grpo_tuning")  # GRPO hyperparameter tuning

# 2. Inspect and fix task environment: dataset + model
env_space = env.show_envs()
# env_space = {
#     "dataset": ["gsm8k", "dapo", "mmlu_chemistry", "mmlu_history",
    "mmlu_physics"],
#     "model":   ["qwen2.5-1.5B-Instruct", "qwen2.5-3B-Instruct"],
#     "training_size": [256, 768, 1536]
#     "epoch": [15, 30]
# }
env.set_env(dataset="gsm8k", model="qwen2.5-3B-Instruct",
    training_size=256, epoch=30)

# 3. Inspect configuration space (hyperparameter grid)
config_space = env.show_configuration_space()
# config_space = {
#     "lr":         [1e-6, 5e-6, 1e-5],  # learning rate
#     "mb":         [16, 32, 64],        # GRPO mini-batch size
#     "kl":         [0.0, 1e-3],         # KL regularization coefficient
# }

# 4. Query a single configuration
cfg = {"lr": 5e-6, "mb": 32, "kl": 1e-3}
res = env.query(config=cfg)
# res = {
#     "score": 0.73,  # target metric (e.g., aggregated accuracy)
#     "additional_information": {
#         "critic_score_per_step": [...],  # per-step critic score
#         "...": "optional textual summary of training info",
#     },
# }
```

Figure 7: **Example usage of the `LMConfigGym` API for RL GRPO Tuning Configuration (Task 3).** The lookup-based interface supports task discovery, environment selection (dataset/model/training_size/epoch), configuration-space inspection, and querying configurations to obtain both target metrics and additional information.

- `query`: Given a configuration (by ID or dictionary), return the target metric (e.g., perplexity, loss, aggregated score) and additional textual information (training score array, etc.) from the lookup table.
- `init_datasets`: Initialize the predefined training/testing datasets; users can also define custom datasets by setting different environments.

A.3 Task 1: Model Architecture Configuration

Building on the 10k architecture evaluations openly released by HW-GPT-Bench (Sukthanker et al., 2024) across the GPT-S/M/L family, we construct a model architecture configuration task in LLMConfig-Gym. The agent chooses GPT-2-style models parameterized by model scale (GPT-S/M/L), embedding dimension (e), number of layers (l), per-layer attention heads $\{h_t\}$, per-layer MLP ratios $\{m_t\}$, and a global bias flag, following the HW-GPT-Bench configuration space. For each configuration s, the Gym returns its validation perplexity on OpenWebText2 (Gao et al., 2020)

from HW-GPT-Bench's open data, which downstream RL or black-box optimizers can turn into their own reward or acquisition functions. Task details are in Table 5.

Table 5: **Task 1: Model Architecture Configuration in LLMConfig-Gym**, built on top of the HW-GPT-Bench dataset.

Category	Item	Description
Environment (Fidelity)	Model Scale	One of `GPT-S`, `GPT-M`, `GPT-L`, as defined in HW-GPT-Bench (up to ∼0.77B parameters).
Configuration Space	Embedding dim. e	Embedding dimension choices for each model, taken directly from HW-GPT-Bench; GPT-S: $e \in \{192, 384, 768\}$, GPT-M: $e \in \{256, 512, 1024\}$, GPT-L: $e \in \{320, 640, 1280\}$.
	Layers l	Number of Transformer blocks from the depth set of the chosen model; GPT-S: $l \in \{10, 11, 12\}$, GPT-M: $l \in \{22, 23, 24\}$, GPT-L: $l \in \{34, 35, 36\}$.
	Per-layer heads $\{h_1, \ldots, h_l\}$	For each layer, the number of attention heads is drawn from the allowed head set of that model; GPT-S: $h_t \in \{4, 8, 12\}$, GPT-M: $h_t \in \{8, 12, 16\}$, GPT-L: $h_t \in \{8, 16, 20\}$, for each layer t.
	Per-layer MLP ratios $\{m_1, \ldots, m_l\}$	For each layer, the MLP ratio is chosen from the MLP-ratio set of the model; $m_t \in \{2, 3, 4\}$, for each layer t.
	Bias flag b	$b \in \{\text{On}, \text{Off}\}$ for all models, indicating whether linear layers use bias.
Outputs	Target metric	For each architecture s, the Gym returns validation perplexity $\text{PPL}_{\text{val}}(s)$ on OpenWebText2 and the corresponding FLOPs.
Meta-features	Task description	Large-model configuration optimization: the agent proposes architectures, observes PPL_{val} and FLOPs, and learns to select better configurations.
	Optimization target	Primary target is the sum of min–max normalized validation perplexity PPL_{val} and min–max normalized FLOPs over the configuration space of the chosen model scale (lower is better).
	Config. description	Structured and human-readable descriptions of each configuration (model scale, e, l, heads, MLP ratios, bias) for use as features or text prompts.

A.4 Task 2: Pretraining Hyperparameter Configuration

Building on the Step Law pre-training study (Li et al., 2025), which trains over 3,700 LLMs from scratch across seven model sizes N and seven dataset sizes D, we construct a pre-training hyperparameter tuning task in LLMConfig-Gym. The tunable hyperparameters are the peak learning rate LR and the global token batch size BS, while (N, D) act as environment variables controlling the fidelity of each run. For each fixed (N, D), the Gym exposes Step Law's grid: LR from a logarithmic sequence of powers of two, $\text{LR} \in \{2^{-10.5}, 2^{-10.0}, \ldots, 2^{-7.0}\}$, and BS from a geometric progression in $\{32{,}768, \ldots, 4{,}194{,}304\}$ tokens/step (ratio $\sqrt{2}$). For each configuration, the Gym returns the final smooth training loss, validated by Step Law as an unbiased proxy for validation loss. This enables downstream RL or black-box optimization methods to learn hyperparameter policies that adapt across model and data scales. Task details are in Table 6.

A.5 Task 3: RL GRPO Tuning Configuration

To support agent training on LLM RL tuning, a widely deployed but expensive workflow, we collect an offline dataset of GRPO runs at 15 and 30 epochs across widely used datasets (GSM8K (Cobbe et al., 2021), DAPO-Math-17k (Yu et al., 2025), MMLU-Pro (Wang et al., 2024)) and two backbones (Qwen2.5-1.5B-Instruct and Qwen2.5-3B-Instruct), via grid search over critical GRPO hyperparameters to cover diverse combinations. All runs use 4 nodes $\times$ 4 NVIDIA A100 80G GPUs and total ∼4,000 GPU hours. Task details are in Table 7. To our knowledge, this is the first offline Gym for large-model RL configuration tuning; beyond the Gym, we also release the full Weights & Biases (W&B) logs to benefit the community.

Table 6: **Task 2: Pretraining Hyperparameter Configuration in LLMConfig-Gym**, built on top of the Step Law pre-training hyperparameter sweeps.

Category	Item	Description
Environment (Fidelity)	Model and data scale	Model size N (non-vocabulary parameters) and dataset size D (number of training tokens).
Configuration Space	Learning rate LR	Peak learning rate selected from a logarithmic sequence of powers of two, with exponents from -12.0 to -6.5 in increments of 0.5, i.e., LR $\in \{2^{-12.0}, 2^{-11.5}, \ldots, 2^{-6.5}\}$. Each experiment exposes the sub-grid swept at its own (N, D), so the range available to the agent varies across experiments.
	Batch size BS	Global token batch size selected from a geometric progression, ranging from 32,768 to 4,194,304 tokens per step, with each subsequent value multiplied by $\sqrt{2}$ relative to the previous one.
Outputs	Target metric	For each configuration $(\mathrm{LR}, \mathrm{BS})$ given N, D, the Gym returns the final smooth training loss.
Meta-features	Task description	Pretraining Hyperparameter Configuration: the agent proposes $(\mathrm{LR}, \mathrm{BS})$ under given (N, D), observes the resulting loss, and learns to select hyperparameters that generalize across model and data scales.
	Optimization target	Primary target is low smooth validation loss at high-fidelity settings (e.g., large N and D), with low-fidelity runs providing cheaper proxy information.
	Config. description	Human-readable descriptions of each configuration (conditioning variables N and D, and hyperparameters LR and BS) for use as structured features or as text prompts for LM-based agents.

Table 7: **Task 3: RL GRPO Tuning Configuration in LLMConfig-Gym**, built from offline GRPO runs on Qwen2.5-1.5B-Instruct and Qwen2.5-3B-Instruct over GSM8K, DAPO-Math-17k, and MMLU-Pro with different training sizes and epochs.

Category	Item	Description
Environment (Fidelity)	Model size and Training Data size	Model size: `Qwen2.5-1.5B-Instruct` or `Qwen2.5-3B-Instruct`; Dataset: `GSM8K`, `DAPO-Math`, and `MMLU-Pro`; Sampled Training size: {256, 768, 1536}, Training Epoch: {15, 30}.
Configuration Space	Learning rate LR	GRPO learning rate; LR $\in \{1\times10^{-6}, 5\times10^{-6}, 1\times10^{-5}\}$.
	Batch size BS	RL batch size (per update); BS $\in \{16, 32, 64\}$.
	KL regularization λ_{KL}	Coefficient on the KL-penalty term between policy and reference model; $\lambda_{\mathrm{KL}} \in \{0,\ 1\times10^{-3}\}$.
Outputs	Target metric	For each hyperparameter configuration $s = (\mathrm{LR}, \mathrm{BS}, \lambda_{\mathrm{KL}})$ and fidelity level, the Gym returns the aggregated evaluation score (e.g., average accuracy) on the test set, used as the optimization signal.
Meta-features	Task description	GRPO-based RL hyperparameter tuning: the agent proposes $s = (\mathrm{LR}, \mathrm{BS}, \lambda_{\mathrm{KL}})$, observes task-level and aggregated performance, and learns to select hyperparameters for large-model RL training.
	Optimization target	Primary target is high aggregated evaluation performance at the high-fidelity (Qwen2.5-3B-Instruct) setting.
	Config. description	Human-readable descriptions of each configuration (chosen LR, BS, λ_{KL}, model scale, and dataset subset) for use as structured features or text prompts in LM-based agents.

A.6 Task 4: Data Mixture Configuration

Building on ADMIRE IFT Runs (Ouyang et al., 2025), a dataset of 460 full instruction-finetuning runs on the Tülu 3 collection (Lambert et al., 2025) using Qwen2.5 at 500M, 3B, and 7B parameters, each scale over its own set of data mixtures, we construct a data mixture configuration task in LLMConfig-Gym. The agent picks one of the precomputed mixtures available at that scale (76 at Qwen2.5-7B) together with a model scale $m \in \{\texttt{Qwen2.5-500M}, \texttt{Qwen2.5-3B}, \texttt{Qwen2.5-7B}\}$. For each configuration π, the Gym returns the average overall (ID + OOD) performance score across the 17 Tülu 3-style benchmarks. Task details are in Table 8.

Table 8: **Task 4: Data Mixture Configuration in LLMConfig-Gym**, built on top of the ADMIRE IFT Runs dataset for Tülu 3 and Qwen2.5 models.

Category	Item	Description
Environment (Fidelity)	Model size	Model size: `Qwen2.5-500M`, `Qwen2.5-3B`, or `Qwen2.5-7B`.
Configuration Space	Data mixture π	Mixture over the Tülu 3 instruction datasets; in our offline Gym, π is chosen from the precomputed mixtures of ADMIRE IFT Runs at the given scale (76 at Qwen2.5-7B). The mixture sets differ across scales, which is what makes Task 4 a configuration-space shift.
Outputs	Target metric	For each configuration s, the Gym returns the average *overall* (ID + OOD) performance score across the 17 Tülu 3-style benchmarks, as reported in ADMIRE IFT Runs.
Meta-features	Task description	Instruction-finetuning data mixture configuration: Given the targeted instruct-tuned model, the agent proposes π, observes overall (ID+OOD) performance, and learns to select mixtures to maximize overall performance.
	Optimization target	Average overall (ID + OOD) performance score.
	Config. description	Human-readable descriptions of each configuration (mixture index or weights over named Tülu 3 datasets, and model scale) for use as structured features or text prompts.

A.7 Task Split on LLMConfig-Gym in Our Paper

To probe cross-fidelity extrapolation, we define a **Low-Fidelity to High-Fidelity (L2H)** setting that orchestrates the collected Gym experiments: agents are trained on low-fidelity experiments and evaluated on high-fidelity ones. The detailed task splits are in Table 9. For Task 3 the test experiments also change the dataset (GSM8K and DAPO-Math instead of MMLU subsets) in addition to training size and epochs, so this split tests transfer across data domain and scale simultaneously.

Table 9: **Low-Fidelity to High-Fidelity (L2H) Task Splits in our experiments using LLMConfig-Gym.** The training column lists the RL training experiments; the in-context demonstration block additionally draws on lower-fidelity experiments reserved for demonstration only and never used as RL targets (Task 2: D=2B, N=214M; Task 3: the Qwen2.5-1.5B-Instruct runs; Task 4: Qwen2.5-0.5B).

Task	Training Experiments (Low-Fidelity)	Testing Experiments (High-Fidelity)
Task 1: Model Architecture	GPT-M	GPT-L
Task 2: Pretraining Hyperparameter (D: Training Tokens N: Model Parameters)	D is 2000000000, N is 268304384 D is 2000000000, N is 429260800 D is 2000000000, N is 536872960 D is 4000000000, N is 59968512 D is 4000000000, N is 119992320 D is 4000000000, N is 268304384 D is 4000000000, N is 429260800 D is 8000000000, N is 59968512 D is 8000000000, N is 119992320	D is 100000000000, N is 214663680 D is 80000000000, N is 268304384 D is 20000000000, N is 536872960 D is 20000000000, N is 429260800 D is 20000000000, N is 268304384
Task 3: RL GRPO Tuning	MMLU_Chemistry(256 training samples), Qwen2.5-3B, Epoch:15 MMLU_History(256 training samples), Qwen2.5-3B, Epoch:15 MMLU_Physics(256 training samples), Qwen2.5-3B, Epoch:15 MMLU_Math(256 training samples), Qwen2.5-3B, Epoch:30	GSM8K(768 training samples), Qwen2.5-3B, Epoch:30 GSM8K(1536 training samples), Qwen2.5-3B, Epoch:30 DAPO(768 training samples), Qwen2.5-3B, Epoch:30 DAPO(1536 training samples), Qwen2.5-3B, Epoch:30
Task 4: Data Mixture	Qwen2.5-3B	Qwen2.5-7B

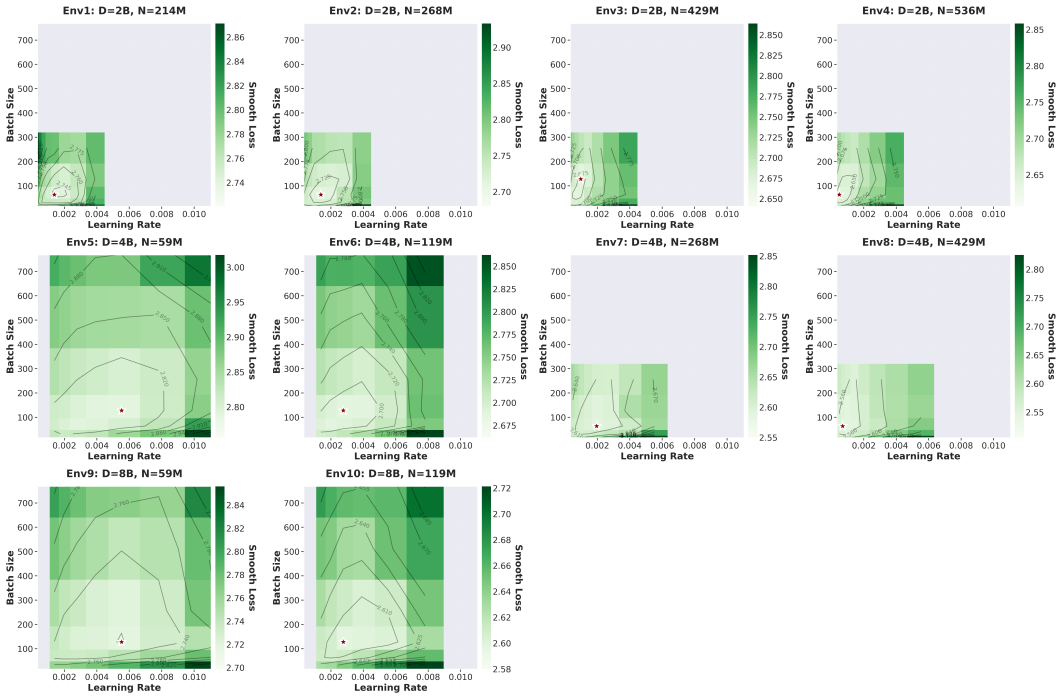
A.8 Multi-Fidelity Optimization Landscape Analysis

We present the optimization landscape per task: Task 1 (Fig. 8), Task 2 (Fig. 9), Task 3 (Fig. 10), and Task 4 (Fig. 11). These landscapes also visualize the two cross-fidelity shifts identified in the introduction: **Tasks 1 and 4 reflect Challenge 2 (configuration space shift)**, where the training and testing landscapes lie over disjoint configuration grids; **Tasks 2 and 3 reflect Challenge 3**

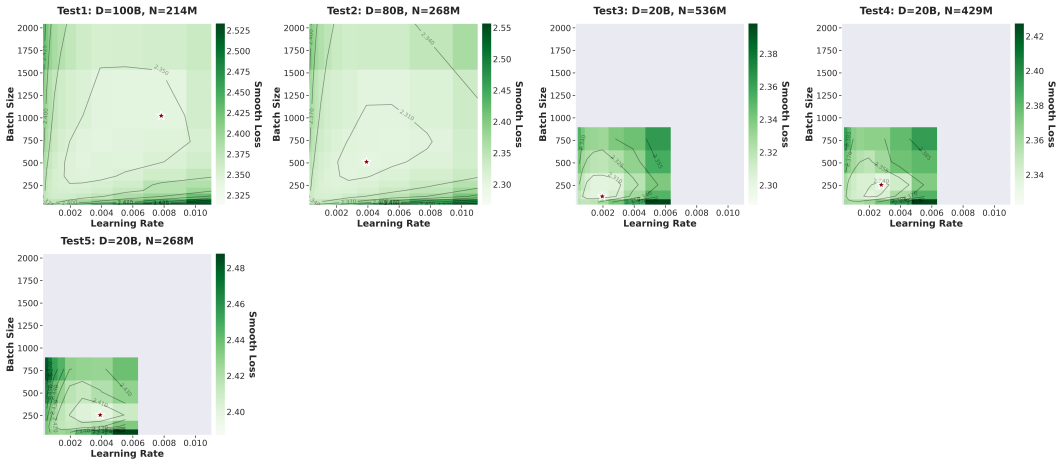
(**optimization landscape shift**), where the configuration space is (partially) shared across fidelities but the optimal region visibly moves between the training and testing heatmaps.



Figure 8: **Optimization landscape example for Task 1: Model Architecture Configuration in LLMConfig-Gym.** Each point corresponds to a sampled model configuration, colored by normalized performance (e.g., validation perplexity or regret). The rugged structure illustrates the presence of multiple local minima and the complexity of the search space.

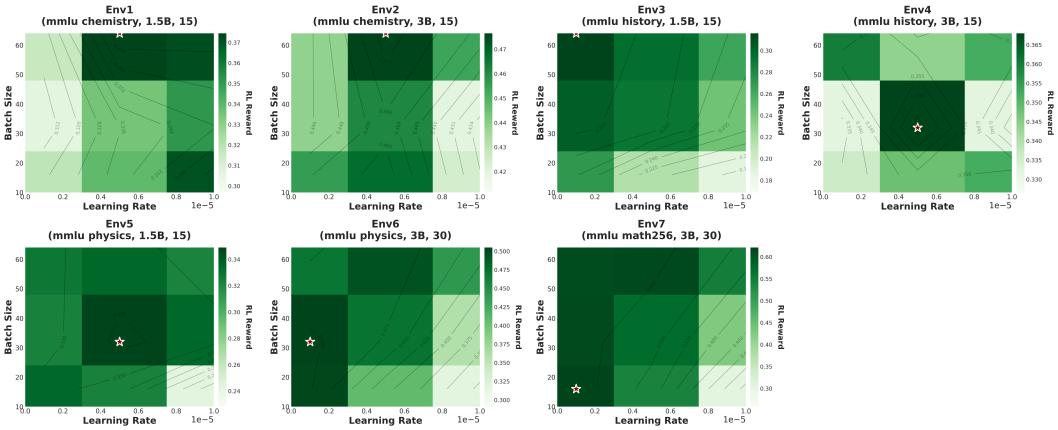


(a) Pre-training hyperparameter tuning, training set. Each point is a sampled $(\mathrm{LR}, \mathrm{BS})$ under fixed (N, D), colored by smooth training loss; the heatmap highlights regions of optimal performance.

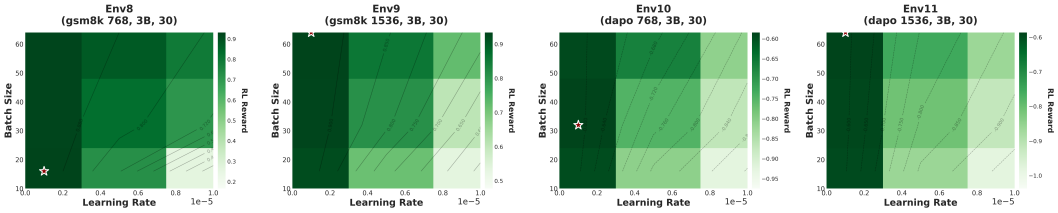


(b) Pre-training hyperparameter tuning, test set. Each point is a sampled $(\mathrm{LR}, \mathrm{BS})$ under fixed (N, D), colored by smooth training loss; the heatmap highlights regions of optimal performance.

Figure 9: **Optimization landscapes for Task 2: Pretraining Hyperparameter Configuration in LLMConfig-Gym.** (a) shows the landscape on the training set, and (b) on the test set.



(a) Optimization landscape for GRPO RL-tuning (training set). For a fixed model and dataset, each cell shows the evaluated accuracy at $(\mathrm{LR}, \mathrm{BS})$ averaged over the two KL coefficients $\lambda_{\mathrm{KL}} \in \{0, 10^{-3}\}$; the star marks the best cell.



(b) Optimization landscape for GRPO RL-tuning (test set). For a fixed model and dataset, each cell shows the evaluated accuracy at $(\mathrm{LR}, \mathrm{BS})$ averaged over the two KL coefficients $\lambda_{\mathrm{KL}} \in \{0, 10^{-3}\}$; the star marks the best cell.

Figure 10: **Optimization landscapes for Task 3: RL GRPO Tuning Configuration in LLMConfig-Gym.** (a) shows the landscape on the training set, and (b) on the test set. The search space is $\mathrm{LR} \times \mathrm{BS} \times \lambda_{\mathrm{KL}}$ (18 configurations per experiment); the heatmaps marginalize over λ_{KL} to show the $(\mathrm{LR}, \mathrm{BS})$ reward surface, whose optimum shifts across models and datasets.

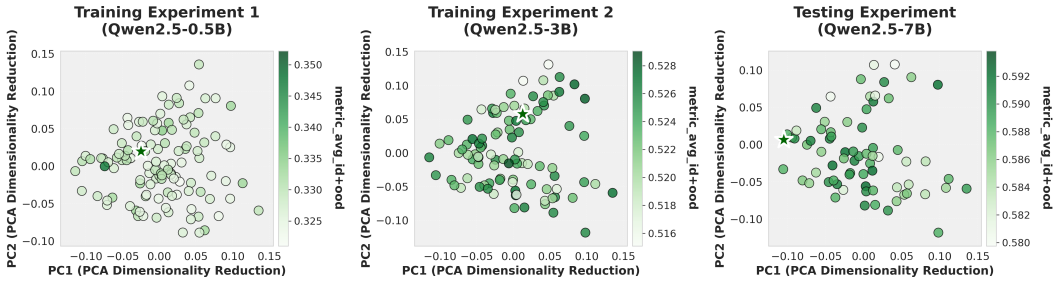


Figure 11: **Optimization landscape for Task 4: Data Mixture Configuration in LLMConfig-Gym.** Each point is a unique mixture configuration (proportions over data sources), colored by evaluation score, illustrating the effect of mixture ratios and the complexity of the search.

B Implementation Details

B.1 Prompts and Meta-Features

The prompts (Instructions and Context Meta-Features) we used for different tasks are reproduced verbatim below, including their original wording and typographical errors (e.g., the Task 2 prompt addresses the model as an "instruction tuning expert" although the task is pretraining hyperparameter selection; the environment variables and objective it states are the pretraining ones), because all reported runs used these exact prompts.

Prompt for Task 1: Model Architecture Configuration

You are a Large Language Model architecture expert. Your task is to optimize a Transformer model architecture to balance (1) training perplexity on the dataset and (2) FLOPs.
Optimization target: Perplexity bounds: - min (best) perplexity = 18.02 - max (worst) perplexity = 30.33. FLOPs bounds: - min FLOPs = 245883994112 - max FLOPs = 2584367595520 . We minimize the following objective (min–max normalized sum): Target = (perplexity - 18.02) / (30.33 - 18.02) + (flops - 245883994112) / (2584367595520 - 245883994112), Lower Target is better.
Configuration space (must strictly follow) The configuration space for the Transformer model architecture is as follows: Here are the meanings of the configuration space: - sample_embed_dim: embedding dimension of the Transformer model - sample_n_layer: number of layers of the Transformer model - sample_n_head: number of attention heads in each layer of the Transformer model (per-layer list) - sample_mlp_ratio: ratio of MLP dimension to embedding dimension of the Transformer model (per-layer list) - sample_bias: whether to use bias in the MLP of the Transformer model ("True"/"False" as a string)
Here are the candidate model architectures (numbered options):
Option 1: 'sample_embed_dim': 320, 'sample_n_layer': 35, 'sample_n_head': [8, 16, 8, 16, 8, 20, 16, 8, 8, 8, 16, 16, 16, 16, 16, 20, 20, 16, 16, 8, 8, 8, 8, 20, 8, 8, 16, 20, 20, 8, 20, 16, 8, 16, 20, 8], 'sample_mlp_ratio': [4, 4, 2, 2, 2, 4, 4, 2, 4, 3, 3, 2, 2, 2, 2, 4, 2, 4, 4, 4, 3, 3, 4, 4, 4, 2, 2, 2, 4, 3, 3, 3, 3, 4, 2], 'sample_bias': 'True'
.....
Option 20: 'sample_embed_dim': 1280, 'sample_n_layer': 35, 'sample_n_head': [16, 20, 20, 8, 20, 20, 16, 8, 8, 8, 16, 20, 8, 16, 8, 20, 8, 20, 8, 20, 8, 16, 8, 8, 16, 20, 20, 16, 20, 20, 20, 16, 20, 8, 16, 8], 'sample_mlp_ratio': [2, 2, 2, 2, 2, 3, 2, 2, 3, 3, 4, 4, 2, 4, 4, 2, 2, 4, 4, 3, 2, 3, 4, 2, 2, 4, 2, 4, 3, 3, 4, 3, 4, 4], 'sample_bias': 'False'
Output requirements (CRITICAL - violations will cause errors) 1) Select EXACTLY ONE architecture from the provided candidate architectures above. Do NOT create any new values and do NOT edit any candidate. 2) Output ONLY the final configuration dict (no extra text, no other words), wrapped between <config> and </config> tags.
Now Please think step by step, select the best architecture dict that achieves the lowest Target, and return the configuration dict between <config> and </config> tags. Think in 4 Steps: Step 1: Propose (Generate 5 Candidates) Based on domain knowledge, previous results, and remaining budget, generate 5 diverse, promising configurations exploring different regions of the search space. Note the configuration must be in the configuration space of the task! Step 2: Imagine (Value Estimation) For each candidate, estimate its performance by reasoning about: - Previous experimental results including the previous proposed configurations and their scores - Training dynamics (convergence, stability) - Model capacity and generalization - Hyperparameter interactions The score should be estimated score based on the previous experimental results, training dynamics, model capacity, and hyperparameter interactions. Step 3: Select Best Select the most promising configuration based on value estimations, domain knowledge, and previous results. Explain your choice and output it between <config> and </config> tags. Step 4: Call "exec_config" tool to get the score of the configuration You MUST have to call "exec_config" tool to query the score of the configuration in the final when you obtain config between <config> and </config> tags!
From previous low-fidelity experiments, here are previous related environment and Top5 score configurations which may be helpful for you to propose the next promising configuration. ###### Experiment Environment information: Medium Transformer architecture search, configuration space: "sample_embed_dim": one from [256, 512, 1024], "sample_n_layer": one from [22, 23, 24], "sample_n_head": for each layer, one from [8, 12, 16], "sample_mlp_ratio": for each layer, one from [2, 3, 4], "sample_bias": one from ["True", "False"], Top-5 configurations: 1. 'sample_embed_dim': 512, 'sample_n_layer': 23, 'sample_n_head': [16, 12, 12, 12, 16, 12, 16, 12, 12, 16, 8, 16, 12, 16, 12, 16, 12, 16, 8, 16, 8, 8, 16, 12], 'sample_mlp_ratio': [4, 2, 2, 2, 3, 4, 3, 2, 3, 4, 3, 3, 2, 4, 2, 4, 3, 3, 2, 3, 3, 4, 3, 4], 'sample_bias': 'True' ..... 5. 'sample_embed_dim': 512, 'sample_n_layer': 24, 'sample_n_head': [8, 8, 12, 16, 8, 16, 8, 16, 8, 16, 16, 16, 12, 8, 12, 16, 12, 16, 16, 16, 12, 12, 12, 16], 'sample_mlp_ratio': [4, 3, 2, 4, 4, 2, 3, 3, 4, 3, 3, 3, 3, 4, 3, 4, 4, 2, 4, 4, 4, 3, 4, 4], 'sample_bias': 'True' ######
Remember: 1. Consider your remaining budge is 1 , previous experimental results, best configurations from low-fidelity experiments when making decisions. 2. You MUST have to call "exec_config" tool to query the score of the configuration in the final when you obtain config between <config> and </config> tags!

Prompt for Task 2: Pretraining Hyperparameter Configuration

You are a Large language model instruction tuning expert, your task now is to optimize the instruction tuning. The model is trained on the dataset the total number of training tokens seen by the model during training is: 100000000000 and the count of trainable model parameters excluding token embedding matrices is: 214663680. These two numbers influence the best learning rate and batch size of the instruction tuning.
Optimization target The optimization target is the smooth loss of the model training on dataset. The lower the smooth loss, the better.
Configuration space (must strictly follow) The configuration space for the instruction tuning is: "'lr': [0.007812, 0.0004883, 0.0002441, 0.0006905, 0.001953, 0.0009766, 0.001381, 0.0003453, 0.003906, 0.01105, 0.005524, 0.002762], 'bs': [736, 2048, 32, 128, 1024, 256, 512, 352, 192, 64]" In this configuration space: - lr: learning rate - bs: batch size
Output requirements (hard constraints) 1) All values must strictly follow the configuration space. 2) Output ONLY the final configuration dict (no extra text, no other words), wrapped between <config> and </config> tags.
Now Please think step by step and propose the best configuration dict that achieves the lowest Target, and return the configuration dict between <config> and </config> tags. Think in 4 Steps: Step 1: Propose (Generate 5 Candidates) Based on domain knowledge, previous results, and remaining budget, generate 5 diverse, promising configurations exploring different regions of the search space. Note the configuration must be in the configuration space of the task! Step 2: Imagine (Value Estimation) For each

candidate, estimate its performance by reasoning about: - Previous experimental results including the previous proposed configurations and their scores - Training dynamics (convergence, stability) - Model capacity and generalization - Hyperparameter interactions The score should be estimated score based on the previous experimental results, training dynamics, model capacity, and hyperparameter interactions. Step 3: Select Best Select the most promising configuration based on value estimations, domain knowledge, and previous results. Explain your choice and output it between <config> and </config> tags. Step 4: Call "exec_config" tool to get the score of the configuration You MUST have to call "exec_config" tool to query the score of the configuration in the final when you obtain config between <config> and </config> tags!
From previous low-fidelity expperiments, here are previous related environment and Top3 score configurations which may be helpful for you to propose the next promising configuration. Experiment Environment information: the total number of training tokens seen by the model during training is: 2000000000, and the count of trainable model parameters excluding token embedding matrices is: 214663680 In this environment, the Top-3 configurations are: 1. learning rate: 0.001381, batch size: 64.0 2. learning rate: 0.001953, batch size: 64.0 3. learning rate: 0.001953, batch size: 128.0 ###### ..... ###### Experiment Environment information: the total number of training tokens seen by the model during training is: 8000000000, and the count of trainable model parameters excluding token embedding matrices is: 119992320 In this environment, the Top-3 configurations are: 1. learning rate: 0.002762, batch size: 128.0 2. learning rate: 0.003906, batch size: 128.0 3. learning rate: 0.005524, batch size: 128.0 ######
Remember: 1. Consider your remaining budge is 2 , previous experimental results, best configurations from low-fidelity experiments when making decisions. 2. You MUST have to call "exec_config" tool to query the score of the configuration in the final when you obtain config between <config> and </config> tags!

Prompt for Task 3: RL GRPO Tuning Configuration

You are a Large language model RL-GRPO tuning expert, your task now is to optimize the RL-GRPO tuning. The model is Reinforcement learning trained on gsm8k dataset with 1536 samples, the base model is Qwen2.5-3B-Instruct, and the training epoch is 30.
Optimization target: The optimization target is the validation score of the RL-GRPO model training. The higher the validation score, the better.
Configuration space (must strictly follow) The configuration space for the RL-GRPO tuning is: "'lr': [1e-06, 5e-06, 1e-05], 'mb': [16, 32, 64], 'kl': [0, 0.001]" In this configuration space: - lr: learning rate of the RL-GRPO training - mb: mini-batch size for the gradient update of the RL-GRPO training - kl: weight of the KL divergence of the RL-GRPO training
Output requirements (hard constraints) 1) All values must strictly follow the configuration space. 2) Output ONLY the final configuration dict (no extra text), wrapped between <config> and </config> tags.
Now Please think step by step and propose the best configuration dict that achieves the highest validation score, and return the configuration dict between <config> and </config> tags. Think in 4 Steps: Step 1: Propose (Generate 5 Candidates) Based on domain knowledge, previous results, and remaining budget, generate 5 diverse, promising configurations exploring different regions of the search space. Note the configuration must be in the configuration space of the task! Step 2: Imagine (Value Estimation) For each candidate, estimate its performance by reasoning about: - Previous experimental results including the previous proposed configurations and their scores - Training dynamics (convergence, stability) - Model capacity and generalization - Hyperparameter interactions The score should be estimated score based on the previous experimental results, training dynamics, model capacity, and hyperparameter interactions. Step 3: Select Best Select the most promising configuration based on value estimations, domain knowledge, and previous results. Explain your choice and output it between <config> and </config> tags. Step 4: Call "exec_config" tool to get the score of the configuration You MUST have to call "exec_config" tool to query the score of the configuration in the final when you obtain config between <config> and </config> tags!
From previous low-fidelity expperiments, here are previous related environment and Top3 score configurations which may be helpful for you to propose the next promising configuration. Experiment Environment information: the dataset is mmlu_chemistry, the model is Qwen2.5-1.5B-Instruct, Note the Training epoch is 15 In this environment, the Top-3 configurations are: 1. kl loss weight: 0.001, learning rate: 1e-05, batch size: 64.0 2. kl loss weight: 0.001, learning rate: 5e-06, batch size: 64.0 3. kl loss weight: 0.0, learning rate: 5e-06, batch size: 32.0 ###### ..... ###### Experiment Environment information: the dataset is mmlu_history, the model is Qwen2.5-1.5B-Instruct, Note the Training epoch is 15 In this environment, the Top-3 configurations are: 1. kl loss weight: 0.0, learning rate: 1e-06, batch size: 64.0 2. kl loss weight: 0.0, learning rate: 1e-06, batch size: 32.0 3. kl loss weight: 0.001, learning rate: 5e-06, batch size: 64.0 ######
Remember: 1. Consider your remaining budge is 1 , previous experimental results, best configurations from low-fidelity experiments when making decisions. 2. You MUST have to call "exec_config" tool to query the score of the configuration in the final when you obtain config between <config> and </config> tags!

Prompt for Task 4: Data Mixture Configuration

You are an expert in data-centric optimization for large language model training. Your task is to optimize the training data mixture for a target language model by maximizing the metric_avg_id+ood. The model is trained on the dataset with different data mixture ratios. The base model is Qwen2.5-7B.
Problem formulation We consider d training data domains. A data mixture is a vector $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_d)$ lying on the probability simplex: $\pi_i \geq 0$, $\sum_{i=1}^{d} \pi_i = 1$. The metric_avg_id+ood of a data mixture $\boldsymbol{\pi}$ is defined as the average of performance metrics in id (internal distribution) and ood (out-of-distribution) data domains:

$$\text{metric_avg_id+ood}(\boldsymbol{\pi}) = \frac{1}{2}\left(\text{metric_avg_id}(\boldsymbol{\pi}) + \text{metric_avg_ood}(\boldsymbol{\pi})\right)$$

where metric_avg_id($\boldsymbol{\pi}$) is the average performance metric of the model trained on the id data domains, and metric_avg_ood($\boldsymbol{\pi}$) is the average performance metric of the model trained on the ood data domains.
The goal is to find the data mixture $\boldsymbol{\pi}$ that maximizes metric_avg_id+ood($\boldsymbol{\pi}$). Note that metric_avg_id+ood($\boldsymbol{\pi}$) is the average of performance metrics in id and ood data domains. The higher the metric_avg_id+ood($\boldsymbol{\pi}$), the better.
Configuration space (must strictly follow) The configuration space for the data mixture tuning is an array of ratios of different data domains. You must choose π from a discrete candidate set. The columns names (this defines the order of the array entries) for each ratio are: "['ratio_coconot_converted', 'ratio_evol_codealpaca_heval_decontaminated', 'ratio_flan_v2_converted', 'ratio_no_robots_converted', 'ratio_numinamath_tir_math_decontaminated', 'ratio_oasst1_converted', 'ratio_personahub_code_v2_34999', 'ratio_personahub_ifdata_manual_seed_v3_29980', 'ratio_personahub_math_v5_regen_149960', 'ratio_tulu_hard_coded_repeated_10',

'ratio_tulu_v3.9_aya_100k', 'ratio_tulu_v3.9_open_math_2_gsm8k_50k', 'ratio_tulu_v3.9_personahub_math_interm_algebra_20k', 'ratio_tulu_v3.9_sciriff_10k', 'ratio_tulu_v3.9_synthetic_finalresp_wildguardmixtrain_decontaminated_50k', 'ratio_tulu_v3.9_table_-gpt_5k', 'ratio_tulu_v3.9_wildchat_100k', 'ratio_tulu_v3.9_wildjailbreak_decontaminated_50k', 'ratio_tulu-3-sft-personas-math-grade']"
Here are the candidate data mixtures (numbered options), each mixture is an array of floats in [0, 1] corresponding to the above domain order: Option 1: [0.0108, 0.0976, 0.0344, 0.0462, 0.0498, 0.0062, 0.045, 0.0112, 0.0942, 0.0, 0.209, 0.072, 0.0172, 0.0248, 0.0176, 0.008, 0.0534, 0.0614, 0.1412] ..... Option 16: [0.0212, 0.0464, 0.0378, 0.0062, 0.023, 0.0052, 0.0232, 0.0476, 0.2244, 0.0002, 0.098, 0.0164, 0.0884, 0.0134, 0.1002, 0.0076, 0.0944, 0.0482, 0.0982]
Output requirements (hard constraints) 1) Select EXACTLY ONE mixture from the provided candidate mixtures above. Do NOT create any new values and do NOT edit any candidate. 2) Output ONLY the final configuration array (no extra text), wrapped between <config> and </config> tags.
Now Please think step by step, select the best data mixture from the provided candidates that maximizes metric_avg_id+ood, and return the configuration array between <config> and </config> tags. Think in 4 Steps: Step 1: Propose (Generate 5 Candidates) Based on domain knowledge, previous results, and remaining budget, generate 5 diverse, promising configurations exploring different regions of the search space. Note the configuration must be in the configuration space of the task! Step 2: Imagine (Value Estimation) For each candidate, estimate its performance by reasoning about: - Previous experimental results including the previous proposed configurations and their scores - Training dynamics (convergence, stability) - Model capacity and generalization - Hyperparameter interactions The score should be estimated score based on the previous experimental results, training dynamics, model capacity, and hyperparameter interactions. Step 3: Select Best Select the most promising configuration based on value estimations, domain knowledge, and previous results. Explain your choice and output it between <config> and </config> tags. Step 4: Call "exec_config" tool to get the score of the configuration You MUST have to call "exec_config" tool to query the score of the configuration in the final when you obtain config between <config> and </config> tags!
From previous low-fidelity expperiments, here are previous related environment and Top5 score configurations which may be helpful for you to propose the next promising configuration. Experiment Environment information: The data mixture ratios on Qwen2.5-3B, the size of the model is 3B Top-5 configurations: 1. [0.004, 0.085, 0.0568, 0.006, 0.0268, 0.0086, 0.052, 0.053, 0.1746, 0.0008, 0.1336, 0.095, 0.0116, 0.0258, 0.0466, 0.0078, 0.1366, 0.0586, 0.0168] .....
Remember: 1. Consider your remaining budge is 5 , previous experimental results, best configurations from low-fidelity experiments when making decisions. 2. You MUST have to call "exec_config" tool to query the score of the configuration in the final when you obtain config between <config> and </config> tags!

B.2 Interactions between Agent and LLMConfig-Gym as Function Calling

The exact tool-call schema (used by the verl function calling mechanism in our main-text setup) is shown below.

Tool Configuration

```
tools:
- class_name: "verl.tools.configcoder_tool.ConfigcoderTool"
  config: {}
  tool_schema:
    type: "function"
    function:
      name: "exec_config"
      description: "A tool for executing configuration dict or array and return the
execution results"
      parameters:
        type: "object"
        properties:
          config:
            type: "string"
            description: "The ONLY generated configuration dict or array between <
config> and </config> tags that will be executed"
        required: ["config"]
```

B.3 Hyperparameter Settings For Training

All training runs on a single node with 4 NVIDIA A100 GPUs (80GB). For Policy Distillation, we run LlamaFactory (Zheng et al., 2024b) with DeepSpeed ZeRO-3 offload (Aminabadi et al., 2022) and gradient checkpointing, performing full-parameter fine-tuning at learning rate $5e{-}6$ with a cosine scheduler, per-device batch size 1 and gradient accumulation 4. For end-to-end Multi-Turn GRPO, we use Verl (Sheng et al., 2025) with FSDP offloading and SGLang (Zheng et al., 2024a) as the inference engine, with training batch size 32, learning rate $1e{-}6$, max prompt length 8500, max response length 13000, max agent–Gym interactions per episode 5, and rollout count 8. The continuations of Trajectory Simulation (Section 4.3) are generated by Qwen3-14B with greedy decoding, and every Gym call inside a continuation is executed, so the appended observations are real. The agent–Gym interaction is implemented as a tool call (`exec_config`) via the verl function calling mechanism. The full hyperparameter dumps for Policy Distillation and End-to-End GRPO training are shown in the following code snippets.

HYPERPARAMETERS FOR POLICY DISTILLATION (SFT)

```
finetuning_type: full
template: qwen3
cutoff_len: 23000
per_device_train_batch_size: 1
gradient_accumulation_steps: 4
learning_rate: 5.0e-6
num_train_epochs: 8.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
```

HYPERPARAMETERS FOR RL

```
--config-path="$CONFIG_PATH" \
--config-name='configcoder_multiturn_grpo' \
custom_reward_function.path=$REWARD_PATH  \
algorithm.adv_estimator=grpo \
data.train_files=$DATA_FOLDER/train${data_suffix}.parquet \
data.val_files=$DATA_FOLDER/test${data_suffix}.parquet \
data.train_batch_size=32 \
data.max_prompt_length=8500 \
data.max_response_length=13000 \
data.filter_overlong_prompts=True \
data.truncation='error' \
data.return_raw_chat=True \
actor_rollout_ref.model.path=$MODEL_PATH \
actor_rollout_ref.actor.optim.lr=1e-6 \
actor_rollout_ref.model.use_remove_padding=True \
actor_rollout_ref.actor.ppo_mini_batch_size=16 \
actor_rollout_ref.actor.ppo_micro_batch_size_per_gpu=4 \
actor_rollout_ref.actor.use_kl_loss=True \
actor_rollout_ref.actor.kl_loss_coef=0.001 \
actor_rollout_ref.actor.kl_loss_type=low_var_kl \
actor_rollout_ref.actor.entropy_coeff=0 \
actor_rollout_ref.model.use_fused_kernels=False \
actor_rollout_ref.actor.use_dynamic_bsz=True \
actor_rollout_ref.actor.ppo_max_token_len_per_gpu=30000 \
actor_rollout_ref.rollout.log_prob_use_dynamic_bsz=true \
actor_rollout_ref.rollout.log_prob_max_token_len_per_gpu=34000 \
actor_rollout_ref.ref.log_prob_use_dynamic_bsz=true  \
actor_rollout_ref.ref.log_prob_max_token_len_per_gpu=34000 \
actor_rollout_ref.model.enable_gradient_checkpointing=True \
actor_rollout_ref.actor.fsdp_config.param_offload=False \
actor_rollout_ref.actor.fsdp_config.optimizer_offload=False \
actor_rollout_ref.rollout.log_prob_micro_batch_size_per_gpu=32 \
actor_rollout_ref.rollout.tensor_model_parallel_size=1 \
actor_rollout_ref.rollout.name=sglang \
actor_rollout_ref.rollout.gpu_memory_utilization=0.8 \
actor_rollout_ref.rollout.n=8 \
actor_rollout_ref.ref.log_prob_micro_batch_size_per_gpu=32 \
actor_rollout_ref.ref.fsdp_config.param_offload=True \
algorithm.use_kl_in_reward=False \
trainer.critic_warmup=0 \
trainer.logger=['console','wandb'] \
trainer.project_name='verl_grpo_configcoder' \
trainer.experiment_name="${data}${data_suffix}_${concrete_model}" \
trainer.val_before_train=True \
trainer.n_gpus_per_node=4 \
trainer.nnodes=1 \
trainer.save_freq=5 \
trainer.test_freq=5 \
trainer.validation_data_dir="./${data}${data_suffix}_${concrete_model}
_rollouts_configcoder_train/" \
actor_rollout_ref.rollout.multi_turn.tool_config_path="$PROJECT_DIR/examples/
sglang_multiturn/config/tool_config/configcoder_tool_config.yaml" \
trainer.total_epochs=50
```

We take the final checkpoint once training-set regret has stabilized.

B.4 Baseline Implementation Details and Evaluation Protocol

This section documents, for every method in Fig. 3, exactly what it observes, how its proposals are scored, and how many runs underlie each number. Every method sees the same candidate set per test experiment and its proposals are post-processed by the same rule (Section 4.4). Table 10 summarizes the protocol per method group; the paragraphs below give the implementation details.

Table 10: **Evaluation protocol per method group.** All methods are evaluated on the same 29 test experiments (10 / 5 / 4 / 10 for Tasks 1–4), the same finite candidate set per experiment, and budgets 1–5, with regret normalized by the lookup-table optimum and worst configuration of each test experiment (Section 5). "Runs" is the number of independent evaluation passes behind each reported mean.

Method group	**Curated Top-K low-fidelity context**	**Fidelity metadata**	**Budget semantics**	**Runs**
Ours (Qwen3-4B / 1.7B, PD+RL)	yes (same prompt)	yes (text)	best of first T proposals	3
Prompt-only Qwen3-4B / 14B	yes (same prompt)	yes (text)	best of first T proposals	3
Prompt-only frontier agents	yes (same prompt)	yes (text)	best of first T proposals	1 (O4-mini: 3)
NAP / MetaBO / FSBO	same training experiments, used as meta-training data	no (numeric inputs only)	best of first T proposals	5 seeds
Random Search	none	no	best of first T proposals	3
Top-K Warm-Start	top-5 training configurations by average rank	no	best of first T proposals	1 (deterministic)

Test episodes and candidate sets. The test set contains 29 experiments (10 GPT-L architecture experiments, 5 pretraining (N, D) settings, 4 GRPO experiments, 10 Qwen2.5-7B data-mixture experiments; Table 9). Each experiment exposes a finite candidate set that is identical for every method: 20 numbered architectures (Task 1), the $(\mathrm{LR}, \mathrm{BS})$ pairs present in the Step Law table for that (N, D) (40 for the three 20B-token settings, 120 for the 80B/100B-token settings; the prompt lists the grid swept at that (N, D), so every listed pair is present in the table) (Task 2), the $3 \times 3 \times 2$ grid (Task 3), and 16 numbered mixtures (Task 4; Section C.1). Budgets 1–5 give 145 (experiment, budget) cells; every reported average is a mean over these cells (per task, then over tasks).

LLM-based agents (ours and prompt-only). All LLM-based agents, trained or not, receive the identical prompt of Section B: task description, optimization target, configuration space, the curated Top-K low-fidelity block ($K = 5$ for Tasks 1 and 4, $K = 3$ for Tasks 2 and 3, with the fidelity metadata of each source experiment), the remaining budget, and the 4-step reasoning scaffold, and interact with the Gym through the same `exec_config` tool with the same feedback template. The tool applies the same rules to every agent: the text between `<config>` and `</config>` is parsed with `ast.literal_eval`; on the candidate-selection tasks (1 and 4) a proposal that fails to parse or is not in the candidate list is redirected to the most similar untried candidate (longest common substring of the lower-cased strings, normalized by the longer length; Section 4.4) and consumes one unit of budget; on the combination tasks (2 and 3) a proposal that fails to parse or is not in the lookup table returns an error message asking the agent to retry and does not consume budget. Repeated proposals are scored normally and consume budget. Each budget is a separate episode started from the same prompt; an episode in which no configuration is scored receives regret 1. Episode length is capped at 10 assistant turns for the Qwen agents (`multi_turn.max_turns`) and at $2T+3$ model calls for the API agents.

Qwen agents. Our agents are Qwen3-1.7B and Qwen3-4B after policy distillation and multi-turn RL (Section B.3); the prompt-only references are Qwen3-4B and Qwen3-14B without any task-specific training. All Qwen agents are served by SGLang through the verl validation loop with the default Qwen3 chat template (thinking mode on), max prompt length 8,500 and max response length 13,000 tokens per turn, and are evaluated three times on the full test set.

Frontier API agents. Claude Opus 5, Claude Fable 5, GPT-5.6 Sol, and O4-mini are called through their chat-completion APIs (September 2026) with provider-default sampling parameters and no reasoning-effort override; transient API errors are retried without re-scoring any configuration. O4-mini was run three times; the other three once each because of API cost, so no error bars are shown

for them. The no-context variants in Section C.4 use the same code path with only the Top-K block removed from the prompt.

Meta-training baselines (NAP, MetaBO, FSBO). We use the official NAP code base (Maraval et al., 2023), which also provides the MetaBO and FSBO implementations, and train one model per task on exactly the training experiments of Table 9, i.e., the same low-/medium-fidelity experiments that form our agent's curated context. These methods take numeric inputs only, so each configuration is encoded as a vector: $(\mathrm{LR}, \mathrm{BS})$ for Task 2; $(\lambda_{\mathrm{KL}}, \mathrm{LR}, \mathrm{BS})$ for Task 3; the 19-dimensional mixture-ratio vector for Task 4; and for Task 1 a 75-dimensional vector $[e,\ l,\ h_1..h_{36},\ m_1..m_{36},\ b]$ with per-layer heads and MLP ratios zero-padded to the maximum depth of 36 and the bias flag as $\{0, 1\}$. Inputs are standardized and objectives are standardized per experiment, as in the original code. At test time the optimization domain of each method is the same finite candidate set the LLM agents see (the "cardinality domain" of the NAP environment), so these baselines select among identical candidates. This also fixes how the disjoint spaces of Tasks 1 and 4 are handled: the numeric encoding lets the meta-learned acquisition function score coordinate values it never saw in training (e.g., embedding width 640), but no fidelity information reaches these models because they have no textual input, which is the structural reason they cannot reason about cross-fidelity trends. Training uses the released hyperparameters: NAP episodes of length 5, 100 PPO iterations with 1,440 episodes per iteration, learning rate 3×10^{-5}; MetaBO 200 iterations with 1,200 episodes per iteration, learning rate 10^{-4}, features (posterior mean and std, incumbent, timestep, budget); FSBO meta-trains its deep-kernel GP for 2,000 epochs and then runs expected-improvement BO. Each method is evaluated with 5 seeds per test experiment as one sequential run of at least 5 steps; the budget-T regret is the best of its first T proposals.

Random Search and Top-K Warm-Start. Random Search draws uniformly at each step (over the candidate list on Tasks 1 and 4; independently per hyperparameter over the listed grid on Tasks 2 and 3) for 5 steps and 3 runs; the grid listed for an experiment is exactly the set swept at its (N, D), so every draw is a valid evaluation. Top-K Warm-Start ranks every configuration of the training experiments by its average rank across those experiments and proposes the top 5 in order at budgets 1–5, and is deterministic. Because Tasks 1 and 4 have no configuration shared between training and test spaces, Random Search is substituted for Top-K Warm-Start on those tasks when averaging, which is why Top-K Warm-Start carries a non-zero standard deviation despite being deterministic. Both report the best of the first T steps at budget T.

B.5 Multi-Turn GRPO Objective

Here we provide the full GRPO objective used in Step 3 (Section 4.4). Each trajectory receives a scalar reward r_i; letting $\mathbf{r} = (r_1, \ldots, r_G)$, we compute a group-normalized advantage shared by all tokens of trajectory i: $A_{i,t} = \big(r_i - \mathrm{mean}(\mathbf{r})\big)/\big(\mathrm{std}(\mathbf{r}) + \delta\big)$, $\forall t$. The resulting GRPO objective is $\mathcal{J}_{\mathrm{GRPO}}(\theta) = \mathbb{E}\big[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\mathcal{M}_i|}\sum_{t\in\mathcal{M}_i}\min\big(r_{i,t}A_{i,t},\ \mathrm{clip}(r_{i,t}, 1-\epsilon, 1+\epsilon)A_{i,t}\big) - \beta\,\mathbb{D}_{\mathrm{KL}}[\pi_\theta \,\|\, \pi_{\mathrm{ref}}]\big]$, where $r_{i,t} = \pi_\theta(o_{i,t} \mid q, o_{i,<t})/\pi_{\theta_{\mathrm{old}}}(o_{i,t} \mid q, o_{i,<t})$ is the per-token importance ratio, $A_{i,t}$ is the token-level advantage, and the mask $\mathcal{M}_i$ retains the agent's own tokens (reasoning, configuration, and tool call) and excludes the Gym's tool responses and the prompt.

C Additional Experimental Results

C.1 RQ1: Performance for Different Tasks

We report per-task average regret across budgets 1–5 for all baselines on the four LLMConfig-Gym tasks (Figs. 13, 14, 15, and 16), using the lookup-table normalization and per-budget protocol of Section 5. **Candidate restriction on Task 4.** Each Task 4 test experiment exposes 16 candidate mixtures drawn uniformly at random without replacement from the 76 mixtures of the underlying 7B table (the draws are fixed and released), and the Gym can score any table entry. To keep the configuration space identical for all methods, we score only proposals inside the candidate set and redirect any proposal outside it to the most similar candidate, exactly as the Gym handles malformed proposals (Section 4.4). This rule is applied uniformly to every method and every trajectory, and all Task 4 numbers in the paper are reported under it.

Qwen3-4B vs. Qwen3-1.7B. Fig. 12 compares our two trained agents. The 1.7B agent (average regret 0.105) already beats every meta-training and classic baseline in Fig. 3 and matches the 4B agent on Task 3 (0.016 vs. 0.019), but it is clearly behind on the tasks whose configuration spaces are larger or shift more between fidelities (Task 1: 0.150 vs. 0.015; Task 4: 0.123 vs. 0.042), and its budget-5 value on Task 2 includes one environment in which it fell into a degenerate reasoning loop and issued no evaluation (regret 1). We therefore use the 4B agent for the main comparison.

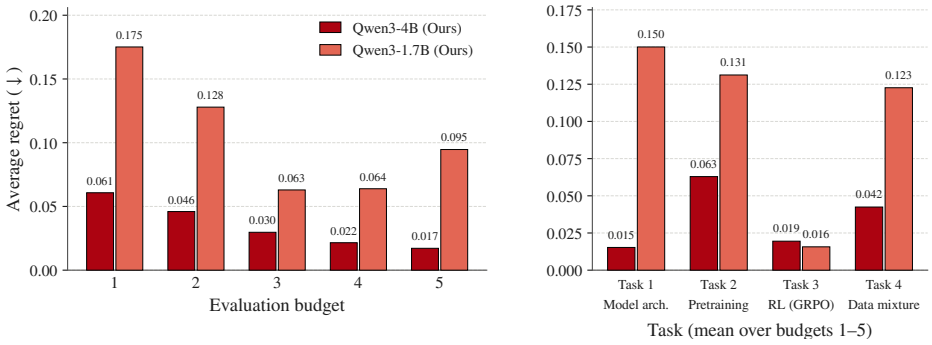


Figure 12: **Our trained Qwen3-4B vs. Qwen3-1.7B agents.** Left: average regret over the four tasks per evaluation budget. Right: per-task average regret over budgets 1–5. Same protocol and normalization as Fig. 3. The 1.7B value on Task 2 includes one environment in which, at budget 5, the agent fell into a degenerate reasoning loop and issued no evaluation (regret 1).

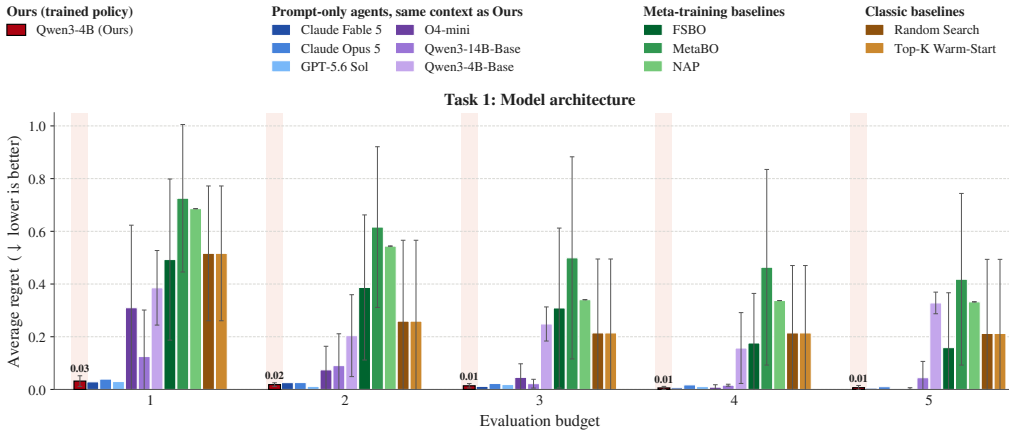


Figure 13: **Performance on Task 1: Model Architecture Configuration.** Average regret over the 10 test experiments; error bars: std over 3 runs (Claude Fable 5, Claude Opus 5 and GPT-5.6 Sol are run once).

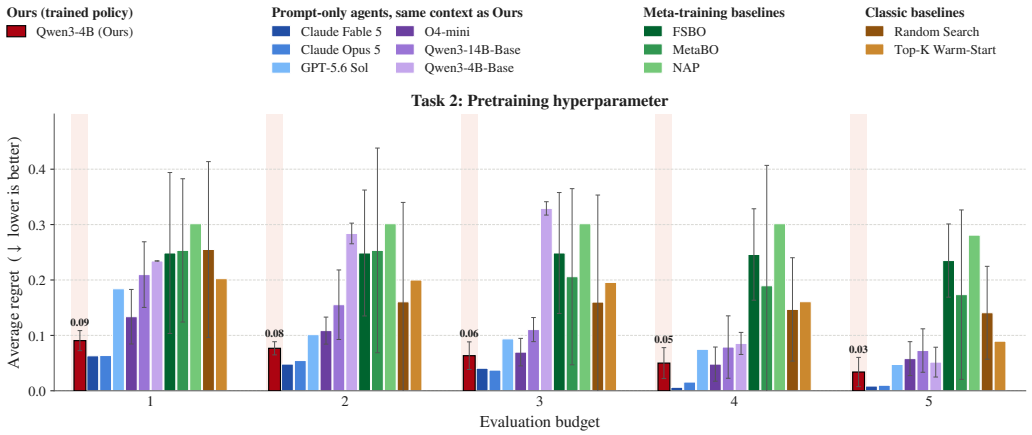


Figure 14: **Performance on Task 2: Pretraining Hyperparameter Configuration.**

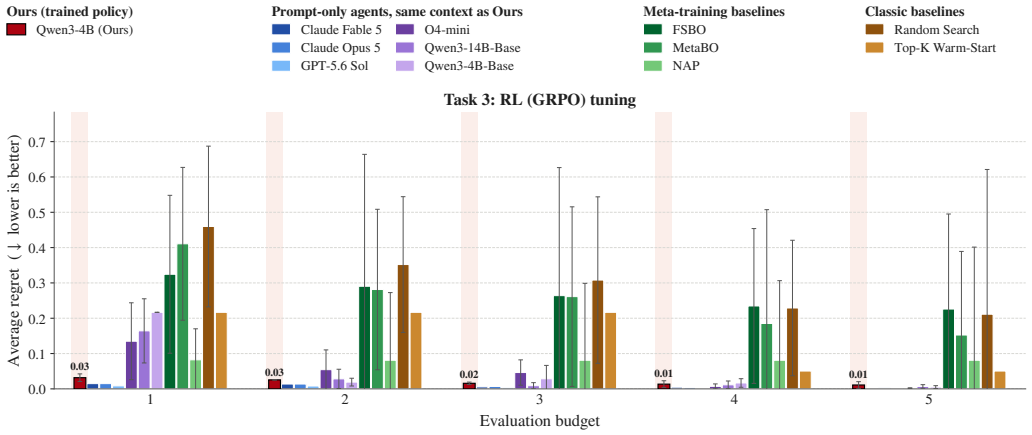


Figure 15: **Performance on Task 3: RL GRPO Tuning Configuration.**

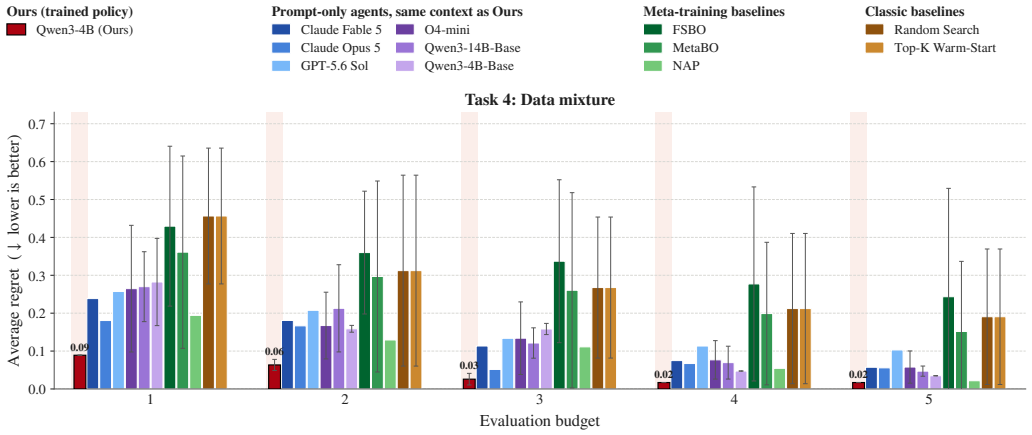


Figure 16: **Performance on Task 4: Data Mixture Configuration.**

C.2 RQ1: Numeric Results and Run-to-Run Variability

Tables 11 to 15 list the exact values behind Fig. 3 and Figs. 13–16: the average regret of every method at each budget, overall and per task, with the standard deviation over runs where repeated runs exist. The Qwen agents, O4-mini, and Random Search are run 3 times and the meta-training baselines with 5 seeds. Claude Fable 5, Claude Opus 5, and GPT-5.6 Sol are run once, because one pass over the 145 test episodes with these models is expensive and a single pass already suffices for the comparison we draw; their numbers therefore carry no standard deviation, and we do not claim differences to them that are smaller than the run-to-run variation of comparable methods.

Two observations put the single-run frontier numbers in perspective. First, our 4B agent is the most stable method in the comparison: its standard deviation averaged over budgets is 0.011 overall and never exceeds 0.03 on any task or budget, whereas the prompt-only Qwen models and O4-mini vary by 0.04–0.07 and the meta-training and classic baselines by 0.05–0.25. Its margins over the meta-training and classic baselines (at least 0.1 at every budget) and over its own prompt-only backbone (0.164 vs. 0.035 on average) are therefore many standard deviations wide. Second, its advantage is consistent rather than driven by a few cells: it has the lowest or tied-lowest average regret at every budget, is within 0.02 of the best method in 14 of the 20 (task, budget) cells, and is the best method on Task 4 at every budget. Relative to the strongest frontier agents, the average differences at budgets 3–5 (≤ 0.003) are smaller than our own run-to-run standard deviation and are not resolved by this experiment, while the gaps at budgets 1–2 (0.014 and 0.019) exceed it; this is why the main text describes the trained 4B agent as reaching the level of the frontier agents, with its clearest advantage at the smallest budgets and on Task 4.

Table 11: **Average regret over the four tasks at budgets 1–5 (B1–B5) and their mean.** Mean ± std over runs (3 runs for the Qwen agents, O4-mini, and Random Search; 5 seeds for NAP/MetaBO/FSBO; Claude Fable 5, Claude Opus 5, and GPT-5.6 Sol were run once because of API cost). Std is averaged over test experiments in regret units. Best value per column in bold.

Method	B1	B2	B3	B4	B5	Avg.
Qwen3-4B (Ours)	**0.061** ±0.012	**0.046** ±0.009	**0.030** ±0.013	**0.022** ±0.011	**0.017** ±0.011	**0.035** ±0.011
Qwen3-1.7B (Ours)	0.175 ±0.074	0.128 ±0.091	0.063 ±0.049	0.064 ±0.046	0.095 ±0.031	0.105 ±0.058
Claude Fable 5	0.086	0.067	0.043	0.023	0.018	0.047
Claude Opus 5	0.075	0.065	**0.030**	0.026	0.020	0.043
GPT-5.6 Sol	0.120	0.082	0.062	0.050	0.039	0.071
O4-mini	0.211 ±0.160	0.101 ±0.065	0.074 ±0.052	0.035 ±0.024	0.030 ±0.019	0.090 ±0.064
Qwen3-4B-Base	0.280 ±0.064	0.167 ±0.049	0.191 ±0.032	0.077 ±0.042	0.105 ±0.018	0.164 ±0.041
Qwen3-14B-Base	0.192 ±0.105	0.122 ±0.081	0.066 ±0.022	0.044 ±0.029	0.043 ±0.030	0.093 ±0.053
FSBO	0.374 ±0.221	0.322 ±0.231	0.290 ±0.247	0.234 ±0.186	0.216 ±0.207	0.287 ±0.219
MetaBO	0.438 ±0.220	0.362 ±0.242	0.307 ±0.264	0.259 ±0.275	0.224 ±0.225	0.318 ±0.245
NAP	0.316 ±0.022	0.264 ±0.048	0.209 ±0.054	0.194 ±0.056	0.179 ±0.080	0.232 ±0.052
Random Search	0.422 ±0.205	0.271 ±0.233	0.238 ±0.224	0.201 ±0.185	0.189 ±0.238	0.264 ±0.217
Top-K Warm-Start	0.348 ±0.109	0.247 ±0.140	0.224 ±0.117	0.159 ±0.114	0.136 ±0.115	0.223 ±0.119

Table 12: **Regret on Task 1: Model Architecture Configuration** at budgets 1–5; conventions as in Table 11.

Method	B1	B2	B3	B4	B5	Avg.
Qwen3-4B (Ours)	0.032 ±0.020	0.018 ±0.007	0.014 ±0.008	**0.006** ±0.006	0.007 ±0.008	0.015 ±0.010
Qwen3-1.7B (Ours)	0.261 ±0.207	0.267 ±0.279	0.099 ±0.134	0.076 ±0.103	0.047 ±0.042	0.150 ±0.153
Claude Fable 5	**0.029**	0.026	**0.012**	**0.006**	0.005	0.015
Claude Opus 5	0.039	0.026	0.022	0.017	0.010	0.023
GPT-5.6 Sol	0.030	**0.012**	0.019	0.011	**0.002**	**0.015**
O4-mini	0.310 ±0.313	0.074 ±0.090	0.046 ±0.052	0.009 ±0.009	0.004 ±0.003	0.088 ±0.093
Qwen3-4B-Base	0.386 ±0.141	0.204 ±0.155	0.248 ±0.065	0.157 ±0.134	0.328 ±0.041	0.265 ±0.107
Qwen3-14B-Base	0.125 ±0.177	0.090 ±0.121	0.022 ±0.017	0.016 ±0.004	0.044 ±0.062	0.059 ±0.076
FSBO	0.493 ±0.306	0.387 ±0.275	0.309 ±0.303	0.176 ±0.188	0.159 ±0.208	0.305 ±0.256
MetaBO	0.725 ±0.280	0.616 ±0.305	0.499 ±0.383	0.464 ±0.371	0.418 ±0.326	0.545 ±0.333
NAP	0.686 ±0.000	0.544 ±0.000	0.341 ±0.000	0.337 ±0.000	0.332 ±0.000	0.448 ±0.000
Random Search	0.516 ±0.256	0.259 ±0.307	0.214 ±0.281	0.214 ±0.256	0.212 ±0.281	0.283 ±0.276
Top-K Warm-Start	0.516 ±0.256	0.259 ±0.307	0.214 ±0.281	0.214 ±0.256	0.212 ±0.281	0.283 ±0.276

Table 13: **Regret on Task 2: Pretraining Hyperparameter Configuration** at budgets 1–5; conventions as in Table 11.

Method	B1	B2	B3	B4	B5	Avg.
Qwen3-4B (Ours)	0.091 ±0.018	0.077 ±0.012	0.063 ±0.025	0.050 ±0.028	0.034 ±0.026	0.063 ±0.022
Qwen3-1.7B (Ours)	0.112 ±0.006	0.083 ±0.034	0.092 ±0.032	0.093 ±0.043	0.276 ±0.046	0.131 ±0.032
Claude Fable 5	**0.063**	**0.048**	0.041	**0.006**	**0.009**	**0.033**
Claude Opus 5	0.064	0.055	**0.037**	0.016	0.010	0.036
GPT-5.6 Sol	0.184	0.101	0.094	0.075	0.047	0.100
O4-mini	0.134 ±0.049	0.109 ±0.024	0.070 ±0.025	0.048 ±0.031	0.058 ±0.031	0.084 ±0.032
Qwen3-4B-Base	0.234 ±0.000	0.284 ±0.019	0.329 ±0.012	0.085 ±0.020	0.052 ±0.027	0.197 ±0.016
Qwen3-14B-Base	0.210 ±0.059	0.155 ±0.063	0.110 ±0.022	0.079 ±0.056	0.073 ±0.039	0.125 ±0.048
FSBO	0.249 ±0.145	0.249 ±0.114	0.249 ±0.109	0.246 ±0.083	0.235 ±0.066	0.245 ±0.103
MetaBO	0.253 ±0.129	0.253 ±0.185	0.206 ±0.159	0.190 ±0.217	0.174 ±0.153	0.215 ±0.169
NAP	0.302 ±0.000	0.302 ±0.000	0.302 ±0.000	0.302 ±0.000	0.281 ±0.000	0.297 ±0.000
Random Search	0.255 ±0.159	0.161 ±0.179	0.160 ±0.193	0.147 ±0.093	0.141 ±0.084	0.173 ±0.142
Top-K Warm-Start	0.202 ±0.000	0.200 ±0.000	0.196 ±0.000	0.161 ±0.000	0.090 ±0.000	0.170 ±0.000

Table 14: **Regret on Task 3: RL GRPO Tuning Configuration** at budgets 1–5; conventions as in Table 11.

Method	B1	B2	B3	B4	B5	Avg.
Qwen3-4B (Ours)	0.032 ±0.011	0.026 ±0.001	0.016 ±0.003	0.013 ±0.010	0.011 ±0.009	0.019 ±0.007
Qwen3-1.7B (Ours)	0.033 ±0.038	0.012 ±0.000	0.008 ±0.005	0.009 ±0.006	0.016 ±0.013	0.016 ±0.012
Claude Fable 5	0.015	0.014	0.006	0.005	**0.002**	0.008
Claude Opus 5	0.015	0.014	0.007	0.004	0.003	0.008
GPT-5.6 Sol	**0.008**	**0.008**	**0.003**	**0.002**	**0.002**	**0.005**
O4-mini	0.135 ±0.109	0.054 ±0.056	0.046 ±0.036	0.007 ±0.007	**0.002 ±0.001**	0.049 ±0.042
Qwen3-4B-Base	0.217 ±0.000	0.019 ±0.011	0.029 ±0.038	0.017 ±0.012	0.005 ±0.004	0.057 ±0.013
Qwen3-14B-Base	0.164 ±0.091	0.028 ±0.027	0.009 ±0.009	0.011 ±0.011	0.007 ±0.005	0.044 ±0.029
FSBO	0.324 ±0.224	0.290 ±0.374	0.264 ±0.363	0.235 ±0.219	0.226 ±0.269	0.268 ±0.290
MetaBO	0.411 ±0.216	0.281 ±0.227	0.261 ±0.254	0.185 ±0.322	0.152 ±0.237	0.258 ±0.251
NAP	0.082 ±0.088	0.081 ±0.192	0.081 ±0.218	0.081 ±0.225	0.081 ±0.320	0.081 ±0.209
Random Search	0.460 ±0.227	0.352 ±0.192	0.308 ±0.236	0.229 ±0.192	0.211 ±0.410	0.312 ±0.251
Top-K Warm-Start	0.217 ±0.000	0.217 ±0.000	0.217 ±0.000	0.050 ±0.000	0.050 ±0.000	0.150 ±0.000

Table 15: **Regret on Task 4: Data Mixture Configuration** at budgets 1–5; conventions as in Table 11.

Method	B1	B2	B3	B4	B5	Avg.
Qwen3-4B (Ours)	**0.089 ±0.000**	**0.063 ±0.015**	**0.026 ±0.015**	**0.017 ±0.000**	**0.017 ±0.000**	**0.042 ±0.006**
Qwen3-1.7B (Ours)	0.294 ±0.047	0.149 ±0.050	0.053 ±0.025	0.077 ±0.033	0.040 ±0.022	0.123 ±0.035
Claude Fable 5	0.238	0.181	0.113	0.075	0.057	0.133
Claude Opus 5	0.181	0.167	0.051	0.067	0.056	0.104
GPT-5.6 Sol	0.257	0.207	0.134	0.113	0.103	0.163
O4-mini	0.265 ±0.167	0.167 ±0.088	0.134 ±0.096	0.077 ±0.051	0.058 ±0.043	0.140 ±0.089
Qwen3-4B-Base	0.282 ±0.115	0.159 ±0.009	0.158 ±0.015	0.047 ±0.000	0.035 ±0.000	0.136 ±0.028
Qwen3-14B-Base	0.270 ±0.092	0.213 ±0.115	0.121 ±0.040	0.069 ±0.043	0.047 ±0.013	0.144 ±0.061
FSBO	0.430 ±0.211	0.360 ±0.162	0.337 ±0.215	0.277 ±0.256	0.243 ±0.286	0.329 ±0.226
MetaBO	0.361 ±0.254	0.297 ±0.252	0.260 ±0.258	0.199 ±0.188	0.152 ±0.185	0.254 ±0.227
NAP	0.194 ±0.000	0.129 ±0.000	0.111 ±0.000	0.054 ±0.000	0.022 ±0.000	0.102 ±0.000
Random Search	0.456 ±0.179	0.312 ±0.252	0.268 ±0.186	0.212 ±0.198	0.191 ±0.179	0.288 ±0.199
Top-K Warm-Start	0.456 ±0.179	0.312 ±0.252	0.268 ±0.186	0.212 ±0.198	0.191 ±0.179	0.288 ±0.199

C.3 RQ1: Raw Task Metrics of the Best Configurations Found

Normalized regret hides the absolute scale of each task. Tables 16 and 17 therefore report, for every method, the raw metric of the best configuration found at budgets 5 and 1, averaged over each task's test experiments, next to the lookup-table optimum and worst configuration. Averaged over the test experiments, the optimum-to-worst range is 0.29 in the Task 1 objective, 0.14 in Task 2 pretraining loss, 0.67 GSM8K accuracy and 0.39 DAPO reward in Task 3, and 0.012 accuracy in Task 4; a regret difference of 0.01 thus corresponds to roughly 0.003 objective, 0.0014 loss, 0.7 GSM8K accuracy points, 0.004 DAPO reward, and 0.0001 accuracy on Task 4, respectively. Task 4 is the task on which candidate mixtures differ least in absolute terms, so its regret differences should be read as rankings among close alternatives rather than large accuracy gaps. On Task 1, the objective couples perplexity and FLOPs; methods with a similar objective can reach it with different trade-offs (e.g., at budget 5 our 4B agent finds architectures with lower perplexity but higher FLOPs than the frontier agents).

Table 16: **Raw task metrics of the best configuration found at budget 5**, averaged over each task's test experiments (mean over 3 runs for Qwen models and O4-mini; the other frontier agents run once). Task 1 objective is the Gym target (min–max normalized log-perplexity + normalized FLOPs, lower is better); perplexity and FLOPs are those of that configuration. Task 2: smoothed pretraining loss (lower is better). Task 3: GSM8K validation accuracy (2 experiments) and DAPO validation reward in $[-1, 1]$ (2 experiments; higher is better). Task 4: average evaluation accuracy over the ID+OOD benchmark suite (higher is better). The last two rows give the lookup-table optimum and worst configuration, i.e., regret 0 and 1.

Method	T1 objective ↓	T1 PPL ↓	T1 FLOPs (T) ↓	T2 loss ↓	T3 GSM8K acc. ↑	T3 DAPO reward ↑	T4 acc. ↑
Qwen3-4B (Ours)	0.666	21.52	1.21	2.3417	0.933	-0.586	0.5921
Qwen3-1.7B (Ours)	0.677	23.49	1.12	2.3578	0.930	-0.588	0.5918
Claude Fable 5	0.665	22.71	0.76	2.3382	0.935	-0.580	0.5916
Claude Opus 5	0.666	22.72	0.76	2.3385	0.935	-0.581	0.5916
GPT-5.6 Sol	0.664	22.67	0.77	2.3431	0.935	-0.580	0.5910
O4-mini	0.665	22.72	0.76	2.3451	0.933	-0.579	0.5916
Qwen3-4B-Base	0.711	22.08	1.31	2.3448	0.932	-0.581	0.5919
Qwen3-14B-Base	0.677	22.77	0.93	2.3461	0.932	-0.582	0.5917
FSBO	0.710	22.78	0.76	2.3661	0.858	-0.690	0.5894
MetaBO	0.786	22.19	0.97	2.3597	0.851	-0.634	0.5906
NAP	0.760	20.88	1.38	2.3779	0.851	-0.590	0.5921
Random Search	0.725	22.53	0.84	2.3546	0.854	-0.700	0.5901
Top-K Warm-Start	0.725	22.53	0.84	2.3491	0.925	-0.609	0.5901
Lookup-table optimum (regret 0)	0.663	22.68	0.76	2.3372	0.935	-0.579	0.5923
Lookup-table worst (regret 1)	0.955	29.21	0.31	2.4807	0.264	-0.971	0.5808

Table 17: **Raw task metrics of the best configuration found at budget 1** (first proposal, no high-fidelity feedback yet); columns and conventions as in Table 16.

Method	T1 objective ↓	T1 PPL ↓	T1 FLOPs (T) ↓	T2 loss ↓	T3 GSM8K acc. ↑	T3 DAPO reward ↑	T4 acc. ↑
Qwen3-4B (Ours)	0.673	22.84	0.75	2.3495	0.918	-0.596	0.5913
Qwen3-1.7B (Ours)	0.739	22.51	1.00	2.3514	0.920	-0.591	0.5889
Claude Fable 5	0.672	22.86	0.75	2.3445	0.929	-0.588	0.5895
Claude Opus 5	0.675	22.94	0.74	2.3446	0.929	-0.588	0.5901
GPT-5.6 Sol	0.672	22.83	0.76	2.3578	0.928	-0.582	0.5893
O4-mini	0.754	24.69	0.62	2.3547	0.900	-0.655	0.5892
Qwen3-4B-Base	0.756	23.17	0.92	2.3653	0.881	-0.707	0.5900
Qwen3-14B-Base	0.700	23.50	0.71	2.3623	0.881	-0.666	0.5892
FSBO	0.807	22.27	1.24	2.3696	0.847	-0.756	0.5873
MetaBO	0.875	24.92	0.92	2.3674	0.775	-0.773	0.5881
NAP	0.864	18.98	2.03	2.3809	0.849	-0.590	0.5901
Random Search	0.814	24.02	0.92	2.3746	0.624	-0.768	0.5871
Top-K Warm-Start	0.814	24.02	0.92	2.3635	0.881	-0.707	0.5871
Lookup-table optimum (regret 0)	0.663	22.68	0.76	2.3372	0.935	-0.579	0.5923
Lookup-table worst (regret 1)	0.955	29.21	0.31	2.4807	0.264	-0.971	0.5808

C.4 RQ1/RQ2: Effect of the Curated Low-Fidelity Context on Prompt-Only Models

To separate the contribution of the curated Top-K low-fidelity context from that of training, we evaluate *untrained, prompt-only* models with and without this context under the same interaction protocol and budgets (Fig. 17). "w/o context" removes only the Top-K low-fidelity block from the prompt; the task description, configuration space, and feedback format are unchanged. Regret uses the same normalization as in Section 5 and is averaged over budgets 1–5 and the four tasks. Qwen3-4B-Base is averaged over 3 runs; the frontier models are run once due to API cost. The

context consistently lowers regret (16%–45% relative), with the largest gains at budgets 1–2 and the smallest gain for the strongest model (Claude Fable 5, 0.056→0.047). Even with the context, however, the best prompt-only model (Claude Opus 5, 0.043) remains above our trained Qwen3-4B agent (0.035 averaged over the same budgets, Fig. 3), and the untrained Qwen3-4B-Base only improves from 0.201 to 0.164 (the same two numbers appear as "w/o Train/Test Experiment Curation" and "Qwen3-4B-Base" in Table 3). The curated context is therefore helpful, but it does not by itself explain the gains obtained from training.

Why the context helps the prompt-only 4B model so little: a behavioral, not an informational, bottleneck. The per-budget context gain of Qwen3-4B-Base in Fig. 3 is uneven ($1.0\times$, $1.2\times$, $1.0\times$, $2.2\times$, $1.5\times$ at budgets 1–5). Inspecting the trajectories shows that the dominant factor is not what the model knows but how it behaves in the experiment loop: the untrained 4B model frequently repeats the same reasoning sentence until the generation limit and never calls `exec_config`, i.e. it fails to *act*. This happens in 133 of 435 episodes (31%) without the context and in 86 (20%) with it, versus 1 episode for prompt-only Qwen3-14B-Base. When all three runs of a (test case, budget) cell fail, the cell receives regret 1 (9 of 145 cells in each condition, at different cells), which produces the irregular per-budget pattern. Restricted to the episodes in which the model does act, the context lowers the regret of Qwen3-4B-Base from 0.170 to 0.116 (per task: Task 1 0.264→0.194, Task 2 0.157→0.127, Task 3 0.082→0.053, Task 4 0.176→0.090), with the largest gains at budgets 1–2 (0.316→0.235 and 0.216→0.123), i.e. the same pattern as for the frontier models. Our training removes this bottleneck in two steps. Policy distillation first teaches the agentic behavior: failures to act drop from 20% to 2.3% of episodes and the average regret from 0.164 to 0.076, before any interaction with the Gym. Multi-turn RL then optimizes the decisions made inside the optimization experiments: failures fall to 0.5% and regret to 0.035 (RL directly on the base model also fixes the behavior, 2.5% failures, but reaches only 0.121, Table 3). The remaining gap between the prompt-only 4B model and the frontier models, once it acts, is a transfer problem: the curated experiments come from a different configuration space (e.g. embedding widths $\{256, 512, 1024\}$ and 22–24 layers for Task 1, versus $\{320, 640, 1280\}$ and 34–36 layers at test time), so the model must map the pattern "the middle width is best" onto the new space. All ten Task 1 test optima use width 640; with the context, Qwen3-4B-Base proposes width 640 first in 67% of its acting episodes (53% without), but still picks 1280 in a sizeable minority, reasoning that a larger width "may give better perplexity".

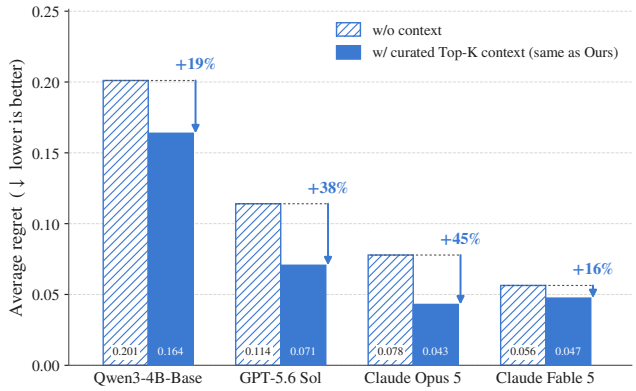


Figure 17: **Effect of the curated Top-K low-fidelity context on prompt-only models.** Hatched: without the context; solid: with the same curated context as our agent. Percentages denote the relative regret reduction from adding the context. Average regret over budgets 1–5 and the four tasks.

C.5 RQ2-1: Full Case Study Details (Configuration Space Shift and Optimization Landscape Shift)

We provide here the full reasoning-trajectory analysis for both case studies summarized in the main text.

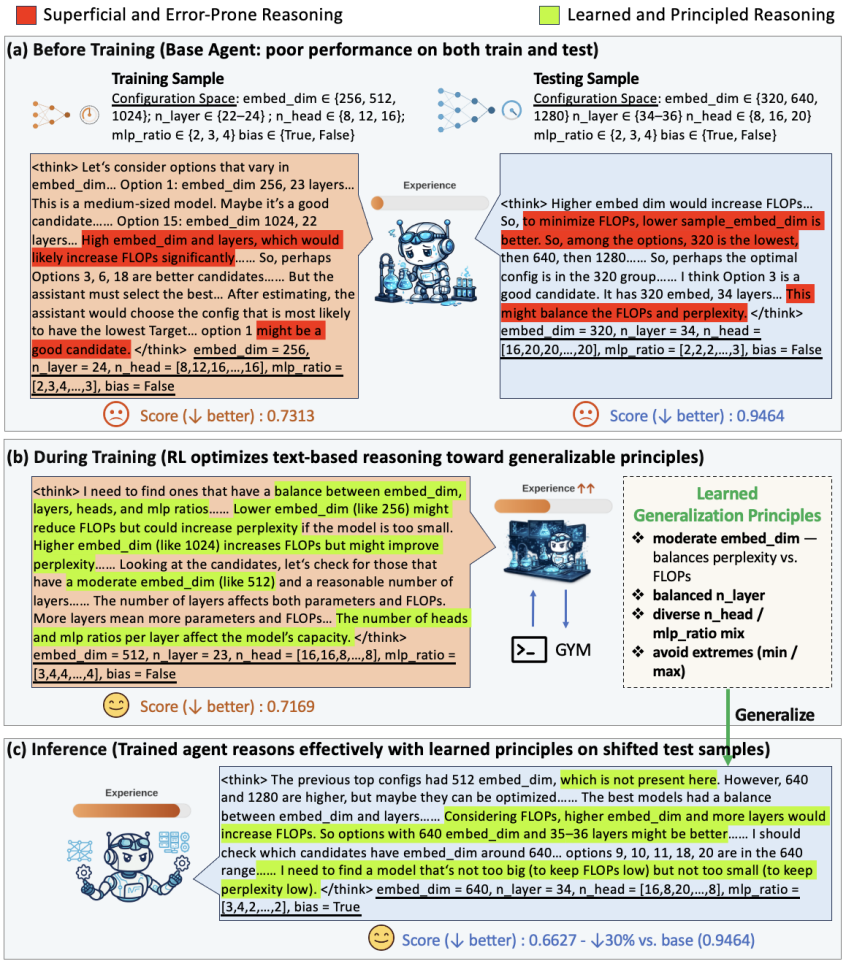
Case Study 1: Configuration Space Shift (Model Architecture). The training and test configuration spaces differ substantially (e.g., `embed_dim` shifts from $\{256, 512, 1024\}$ to $\{320, 640, 1280\}$; `n_layer` from {22–24} to {34–36}), so configurations optimal on the training set do not even exist

on the test set. **(1) Before Training.** The base agent reasons heuristically on both sets: it flags FLOPs concerns for large `embed_dim` but fails to analyze the perplexity vs. FLOPs trade-off, defaulting to the minimum (`embed_dim`=256 on training, 320 on test). The reasoning is essentially circular and yields poor scores on both (0.7313 train / 0.9464 test). **(2) During Training.** On the training set, RL rewards drive the agent toward explicit multi-objective analysis: "*lower `embed_dim` reduces FLOPs but may hurt perplexity; higher values help perplexity but increase FLOPs*". The agent then selects the moderate `embed_dim`=512 with a diverse architecture. Crucially, the reward shapes a *principle* ("balance extremes"), not a fixed configuration. **(3) Inference on Test Set.** Applying the same principle to the shifted test space, the agent selects the moderate `embed_dim`=640 (the middle value in $\{320, 640, 1280\}$), even though this exact configuration never appeared during training. The test score improves from 0.9464 to 0.6627 (a 30% relative gain), showing that what transfers across the configuration-space shift is not a configuration but a *scaling-aware balancing rule*.

Case Study 2: Optimization Landscape Shift (Pretraining Hyperparameter). Training and testing share the same configuration variables (`lr`, `bs`) but differ in fidelity (2B vs. 20B tokens), causing the optimization landscape to shift. **(1) Before Training.** On the training set, the base agent grounds its choice in vague claims about "previous top configs" without analyzing actual prior experiments. At test time it is worse: it latches onto an oversimplified trend ("more tokens ⇒ higher lr"), entirely ignores the parameter–lr relationship, and selects the highest available `lr`=5.5e-3 for a 536M-parameter model, the wrong direction. **(2) During Training.** On the training set, the agent learns to reason jointly over fidelity variables: "*higher parameters lead to lower lr to avoid divergence; more tokens allow slightly higher lr*". Rather than memorizing a single best configuration, it internalizes *fidelity-dependent trends* (e.g., at 268M params the best lr was 1.95e-3; at 429M, 6.9e-4) and uses them as a reasoning scaffold. **(3) Inference on Test Set.** At test time with 536M params and 20B tokens, the agent recalls both trends and weighs them against each other ("for 536M params the lr should be between 1.95e-3 and 6.9e-4, adjusted slightly upward for more tokens"); giving the larger token budget the greater weight, it selects the upper end, `lr`=1.95e-3, `bs`=128. This yields a 0.67% gain over the base agent while avoiding its catastrophic over-extrapolation, showing that what transfers across the optimization-landscape shift is a *fidelity-dependent scaling trend*.

C.6 RQ2-2: From Optimization Perspective: Text-Based Reasoning Prunes Search Space

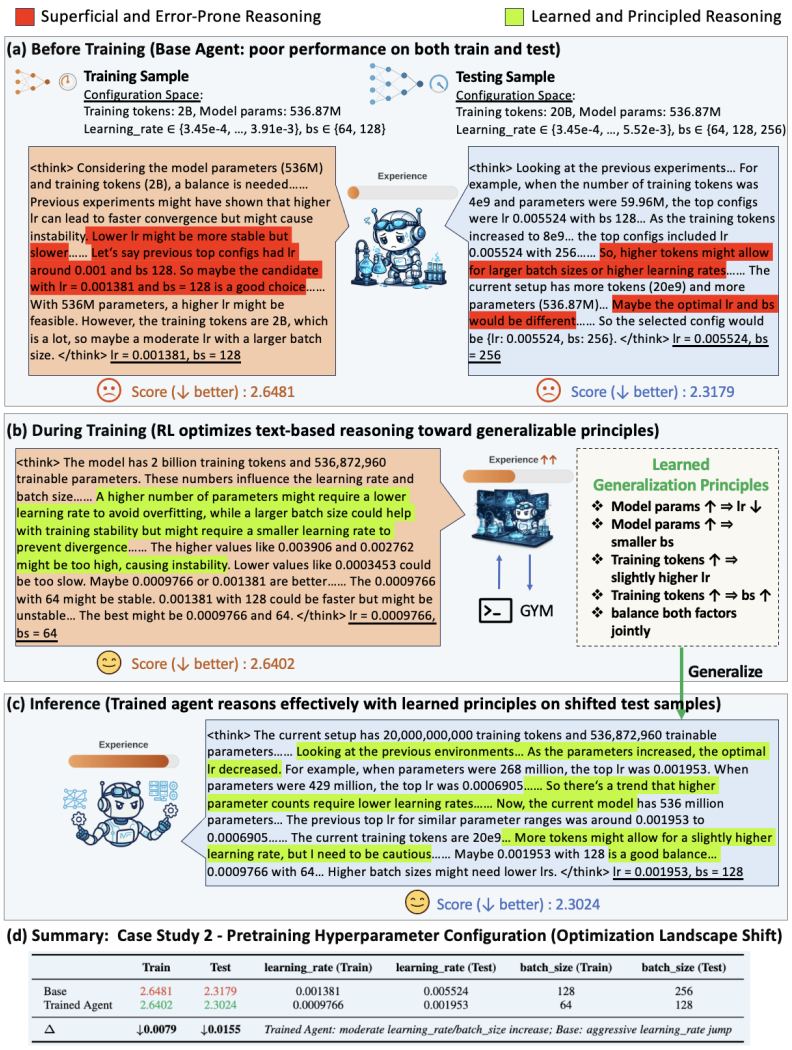
Beyond qualitative reasoning, we examine whether the agent's improved reasoning translates into measurable *optimization effectiveness*. To isolate the contributions of our two design modules (Experiment Curation and RL training with the Gym), we compare three variants on the same Pretraining Hyperparameter Configuration task (Task 2): **Qwen3-4B-Base (w/o experiment curation)**, **Qwen3-4B-Base**, and **Qwen3-4B (Ours, trained agent)**. For each method, we plot the trajectory R1→R2→R3 (R_k is the k-th round). Since each run is repeated three times, we further draw a dashed ellipse enclosing all explored configurations across repeated runs, characterizing the *effective search region*. The background heatmap is the ground-truth loss across the full configuration space. Three distinct behaviors emerge in Fig. 4: **(1) Without experiment curation**, the agent has no low-fidelity context and explores erratically, jumping from `lr=7.5e-4, bs=256` to `lr=4e-3, bs=768` without analyzing R1's feedback. Its search region is the largest, covering optimal and suboptimal configurations indiscriminately. This indicates that without fidelity-aware curation, the agent lacks the context to make informed decisions and falls back to random-like exploration; the wide, unfocused region visually confirms that the budget is wasted across the entire space rather than directed at promising areas. **(2) The base model without training** reads low-fidelity trends but extrapolates in the *wrong direction*: it over-generalizes "*lower lr is better for more parameters*" and progressively drifts toward overly conservative `lr=5e-4, bs=64` in R2, moving away from the optimum. Low-fidelity context alone is insufficient: without RL training to calibrate the reasoning process, the agent latches onto surface heuristics that lead to systematic errors. Its search region is narrower than variant (1) but is concentrated in the wrong area, showing that incorrect extrapolation can be worse than no extrapolation at all. **(3) Our trained agent** correctly extrapolates from low-fidelity experience: it lands at `lr=1.5e-3, bs=128` in R1 (already near the global best), then makes an informed incremental adjustment to `lr=2e-3, bs=128` in R2 to reach the global optimum (loss 2.302). Its search region is tightly concentrated around the optimum, confirming that RL training on text-based reasoning yields an agent that both identifies a strong starting point from the very first iteration and refines it through principled increments rather than broad exploration or surface heuristics.



(d) Summary: Case Study 1 - Model Architecture Configuration (Configuration Space Shift)

	Train	Test	embed_dim (Train)	embed_dim (Test)	mlp_ratio (Train)	mlp_ratio (Test)
Base	0.7313	0.9464	256 (min)	320 (min)	avg≈2.9	avg≈2.8
Trained Agent	0.7169	0.6627	512 (mid)	640 (mid)	avg≈3.2	avg≈3.0
Δ	↓0.0144	↓0.2837	*Trained Agent: balanced dim + diverse arch; Base: default-to-min*			

Figure 18: **Case Study 1 on Model Architecture Configuration, targeting Challenge 2 (Configuration Space Shift).** Reasoning trajectories of the base agent (*Before Training*) and our RL-trained agent (*During Training* and *Inference*) on the training set (`embed_dim` $\in \{256, 512, 1024\}$, `n_layer` $\in \{22–24\}$) and the disjoint test set (`embed_dim` $\in \{320, 640, 1280\}$, `n_layer` $\in \{34–36\}$). Highlighted phrases mark key reasoning steps. The base agent defaults to the minimum `embed_dim` on both sets, whereas the RL-trained agent learns a balanced-architecture principle (moderate `embed_dim`, balanced `n_layer`, diverse `n_head`/`mlp_ratio` mix) that transfers to the shifted test space, yielding a $\sim$30% lower test objective.



(d) Summary: Case Study 2 - Pretraining Hyperparameter Configuration (Optimization Landscape Shift)

	Train	Test	learning_rate (Train)	learning_rate (Test)	batch_size (Train)	batch_size (Test)
Base	2.6481	2.3179	0.001381	0.005524	128	256
Trained Agent	2.6402	2.3024	0.0009766	0.001953	64	128
Δ	**↓0.0079**	**↓0.0155**	*Trained Agent: moderate learning_rate/batch_size increase; Base: aggressive learning_rate jump*			

Figure 19: **Case Study 2 on Pretraining Hyperparameter Configuration, targeting Challenge 3 (Optimization Landscape Shift).** Reasoning trajectories on the training set (low-fidelity, 2B tokens) and the test set (high-fidelity, 20B tokens) under the same configuration variables (`lr`, `bs`) but different fidelities. The base agent ignores the parameter–lr scaling trend and over-extrapolates lr at test time, while the RL-trained agent learns fidelity-dependent trends (params ↑⇒ lr ↓; tokens ↑⇒ slightly higher lr; tokens ↑⇒ bs can ↑) and produces a calibrated configuration that improves over the base agent.

Together, these results show that text-based reasoning learned from experience acts as a *search prior*: it provides a strong initial configuration from the first round and enables correct trend extrapolation that prunes the search space (ours), whereas incorrect extrapolation or lack of grounding yields poor initializations and wastes budget in suboptimal regions (other variants). This complements the reasoning-level case studies in Appendix C.5, confirming that the transferable principles learned by our agent translate directly into measurable search efficiency gains. The three variants jointly validate our two-module design: Experiment Curation provides the necessary fidelity-aware context, and RL training with the Gym calibrates the agent's reasoning to leverage that context for effective extrapolation.

C.7 Amortized Cost Analysis under Scalable Deployment

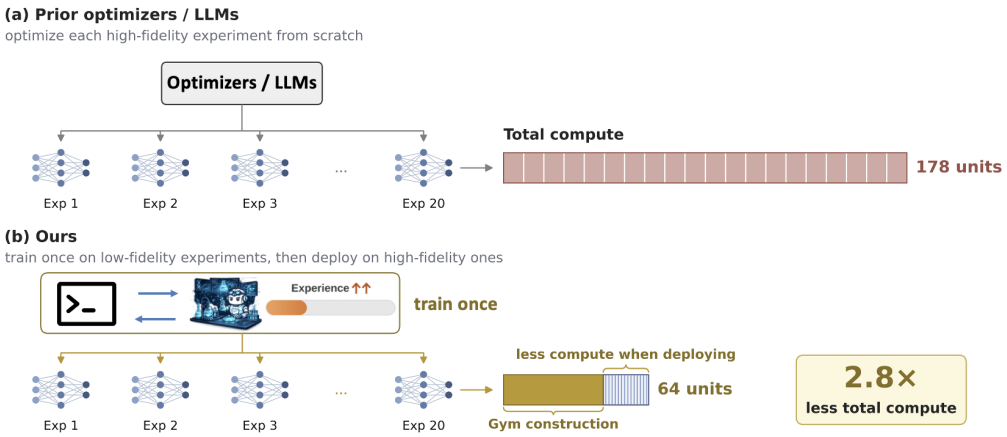


Figure 20: **Amortized compute: one-time training reduces total compute.** Total compute to reach the same target performance over 20 high-fidelity experiments, by method; one unit is our agent's average per-experiment optimization cost in GPU hours. (a) Prior optimizers and LLMs, averaged over Optuna (Akiba et al., 2019) and the same Qwen3-4B base LLM prompted without curated experiments or RL training, optimize every experiment from scratch. (b) Our agent is trained once on low-fidelity experiments and then deployed on high-fidelity ones, where it needs less compute per experiment.

Fig. 20 compares the total compute that different methods need to reach the *same target performance* on a sequence of high-fidelity experiments. The target for each experiment is the performance our agent reaches at budget $T = 5$. For every method we measure the GPU hours of the high-fidelity evaluations it consumes before reaching this target. The from-scratch reference ("prior optimizers / LLMs" in the figure) averages two methods: Optuna (Akiba et al., 2019), a standard hyperparameter optimizer, and the same Qwen3-4B base LLM prompted without curated experiments and without RL training, i.e., our backbone before either component of our method is applied. Both start from scratch on every experiment, so these evaluation hours are their only cost, and we average them over the two methods and over our test experiments to obtain an average per-experiment cost. All costs are then expressed in *units*, where one unit equals our agent's average per-experiment optimization cost in GPU hours, so the per-experiment cost of our agent is one unit by construction and the per-experiment cost of the from-scratch methods is the ratio of their GPU hours to ours. Our agent additionally pays a one-time cost: the GPU hours of the low-/medium-fidelity training experiments it learns from (Gym construction), converted into the same units. Since our test experiments are limited in number, the figure projects both totals to 20 high-fidelity experiments using these average per-experiment costs: the from-scratch methods accumulate a total that grows linearly with the number of experiments (178 units at 20), whereas our total is the one-time cost plus one unit per experiment (64 units at 20, i.e., $2.8\times$ less), and the ratio keeps improving as more experiments are configured because the one-time cost is amortized over all of them. This analysis illustrates the scalability advantage of learning once from cheap experiments, not precise savings for any specific deployment; in practice, the upfront cost can be absorbed by **idle GPU time**, since the low-/medium-fidelity training experiments are offline and can be scheduled whenever spare compute is available, and these grid-search experiments are ones researchers would typically conduct to explore scaling laws and tuning principles.

C.8 RQ3: Training Dynamics, Metric Definitions, and Full Analysis

Multi-Turn Instruction Following. Beyond reasoning quality, our agent also requires strong **multi-turn instruction following**: each episode iterates think→propose→Gym call under strict format constraints. We evaluate **Execution Rate** (fraction of successful Gym calls over the budget, capturing well-formatted tool calls) and **Unique Configuration Rate** (fraction of distinct proposed configurations, capturing avoidance of redundant submissions). **Result 9:** Distillation and RL jointly improve reliable multi-turn interaction. As shown in Table 18: 1) the base model performs poorly, often producing malformed function calls; 2) DirectRL reaches a high execution rate but a lower unique-config rate, as long reasoning chains cause the model to lose track and RL rewards saturate at repetitive trajectories, limiting overall performance; and 3) Policy Distillation improves both, and adding RL further boosts them.

Table 18: **Instruction-following capabilities evaluation across training methods. Execution Rate** and **Unique Config. Rate** measure the ratios of successful function calls and distinct configurations, respectively, to the allocated budget.

Method	Exec. Rate (%) ↑	Unique Cfg. (%) ↑
Base	82.3	80.2
DirectRL	97.9	94.7
Policy Distillation	97.0	96.1
Policy Distillation + RL Training	**98.6**	**98.2**

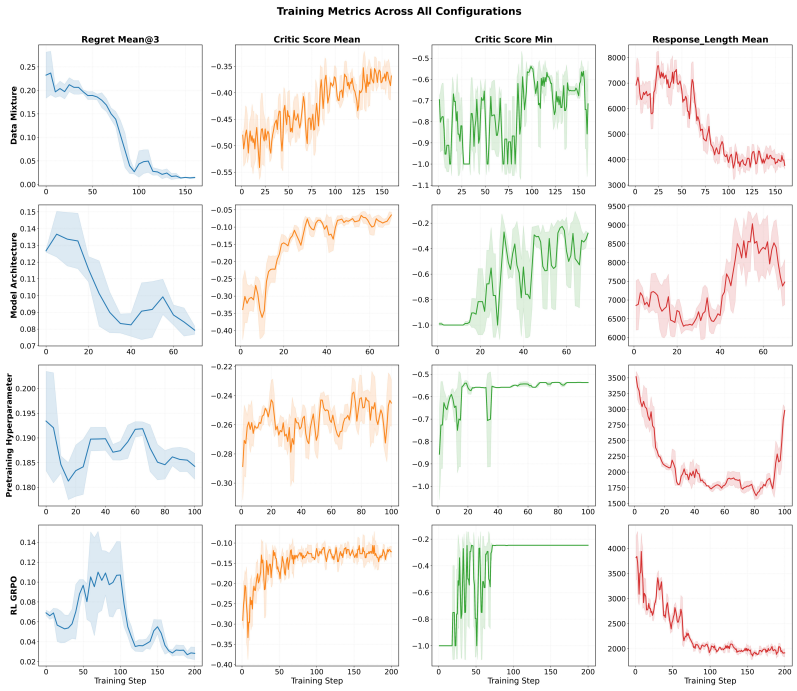


Figure 21: **Training dynamics across all four tasks**, showing **Regret Mean@3**, **Critic Score Mean**, **Critic Score Min**, and **Response Length Mean** over training steps.

We list here the precise definitions of the four metrics we track throughout training, with training samples partitioned by task type:

- **Regret Mean@3**: average regret on the held-out test set, measuring whether learned principles extrapolate to unseen high-fidelity experiments.
- **Critic Score Mean**: average reward on training samples, reflecting configuration quality on tasks the agent is trained on.
- **Critic Score Min**: minimum reward across training samples per batch, capturing worst-case behavior and the rate of invalid configurations.

- **Response Length Mean**: average response length (tokens), indicating whether the agent adapts its reasoning complexity to task types.

Fig. 21 shows the following trends across all four task categories. **Regret Mean@3 decreases consistently**, indicating that the agent progressively proposes higher-quality configurations; since regret is measured on the held-out high-fidelity test set, this confirms that principles learned from low-/medium-fidelity training experiments successfully extrapolate. **Critic Score Mean rises steadily** (gradually rather than in sudden jumps), suggesting the agent is internalizing transferable principles rather than memorizing specific configurations. **Critic Score Min** is particularly informative: for tasks with complex configuration spaces (Model Architecture, Task 1; RL GRPO Tuning, Task 3), it initially stays at -1.0 due to frequent format violations but gradually rises, confirming the agent learns to avoid catastrophic failures. **Response Length Mean** varies by task: longer trajectories for tasks requiring nuanced trade-off analysis (e.g., Model Architecture's multi-dimensional search space), shorter outputs for simpler spaces, a task-adaptive behavior that emerges naturally from RL training without explicit length supervision.

C.9 Stress-Testing the Boundary of Low-to-High Extrapolation

We further probe the boundary of transferability under an adversarial training regime in which the optimal region of the training experiments is *reversed* relative to the test experiments (Challenge 3), comparing against the strongest meta-training baseline (NAP). Both methods are re-trained on exactly the same reduced sets of training experiments and evaluated on the unchanged test set; regret follows the unified protocol of Appendix C.1 (per-case lookup-table normalization, no best-so-far accumulation for our agent) and is averaged over test experiments and budgets 1–5.

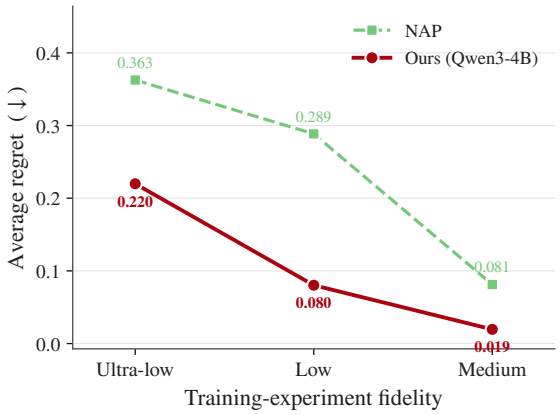


Figure 22: **Reversed optimal region (Challenge 3)** on the **RL GRPO Tuning Configuration** task (Task 3). "Ultra-low" trains only on the experiment whose optimum is the *worst* configuration at test time; "Low" adds experiments whose optima lie between the reversed and the test optimum; "Medium" is the main setup. NAP is re-trained on the same experiments as our agent at every level. Our agent stays below NAP at every level ($1.7\times$, $3.6\times$, $4.2\times$ lower regret) and recovers to 0.080 as soon as non-adversarial experiments are added, whereas NAP remains at 0.29–0.36 until the full training set is available.

Reversed Optimal Region (Challenge 3). We stress-test the agent's ability to capture configuration *trends* across fidelities by selecting training experiments whose optimal region contradicts the test set. All four Task 3 test experiments (GSM8K/DAPO, Qwen2.5-3B) are optimized by the smallest learning rate (10^{-6}). "Ultra-low" trains only on the MMLU-chemistry/Qwen2.5-1.5B experiment, whose optimum is the *largest* learning rate (10^{-5}), i.e. the worst configuration at test time; "Low" trains on three experiments (MMLU-physics/1.5B, MMLU-chemistry/3B, MMLU-physics/3B) whose optima (10^{-5}, 5×10^{-6}, 5×10^{-6}) lie between the reversed and the test optimum; "Medium" is the main setup. Fig. 22 shows that (1) NAP is severely misled when the training optimum contradicts the test one: its regret is 0.363 at "Ultra-low" and still 0.289 at "Low", and only drops to 0.081 with the full training set, because it can only re-propose the configurations that were optimal in its training experiments; (2) our agent also degrades under the purely adversarial split (0.220), but recovers to 0.080 once experiments with intermediate optima are added and reaches 0.019 at "Medium", staying below NAP

at every level. Because the agent starts from pretrained knowledge about how learning rates and KL coefficients should shift with model and data scale, contradicting training experiments move it less far from the test optimum than they move a purely data-driven meta-optimizer. We note the limit of this robustness: the agent trained on the purely adversarial split degrades further with additional RL steps on that split, so the recovery depends on at least some training experiments pointing in the right direction. **Result 10:** Under a reversed training optimum, our agent degrades more gracefully than NAP and recovers as soon as experiments with intermediate optima are available, showing that text-based reasoning captures fidelity-dependent trends rather than replaying training-time optima.

D Broader Impact and Future Work

This work lowers the compute and expert effort needed to configure expensive LLM experiments, and the same recipe applies wherever cheap trials can inform expensive decisions (e.g., catalyst optimization (Tom et al., 2024; Gupta et al., 2025)). The released agents select configurations from the finite, pre-executed candidate sets of an offline environment at scales up to 7B; we regard them as assistants whose proposals a human researcher decides to execute. LLMConfig-Gym exposes only aggregate metrics of runs on public sweeps and public datasets (GSM8K, DAPO-Math-17k, MMLU-Pro, Tülu 3) and does not redistribute their text; conclusions drawn from it inherit the limitations and biases of these benchmarks. The Gym entries, train/test splits, prompts, code, and the W&B logs of our GRPO runs will be released for research use after removing account identifiers, machine paths, and credentials. The compute newly consumed for this paper is the in-house GRPO sweep (∼4,000 A100 GPU hours, Table 4), agent training on a single 4-GPU node, and the API calls of the prompt-only baselines; the remaining ∼1M GPU hours of the Gym are reused public sweeps, and releasing the Gym as a lookup table means that no future evaluation on it re-executes an LLM experiment.

Future directions include expanding the Gym with more tasks, fidelity levels, and larger target scales; multi-objective configuration that balances competing goals; and validating trained agents on live experiment pipelines beyond the offline environment. More broadly, an agent that accumulates experience from cheap experiments to inform expensive ones relates to the long-term goal of AI assisting AI development (Good, 1966; Schmidhuber, 2006; Zhuge et al., 2026; Rank et al., 2026; Lu et al., 2024); our work is a small step within this agenda and a practical lever for assisting human researchers as experiment costs continue to scale.

E Reproducibility and Release

An anonymized repository with the code and data is available at `https://anonymous.4open.science/r/AutoLLMResearch-C5CB/README.md`. Every regret number in this paper can be regenerated from the released artifacts without executing any LLM experiment. We release: **(1) LLMConfig-Gym**: the lookup tables of all four tasks (every configuration with its metrics and additional information), the unified API (Section A), the train/test experiment splits (Table 9), the per-experiment candidate lists of Tasks 1 and 4, and the per-experiment optimum and worst values used for regret normalization. **(2) Prompts and tools**: the full prompt templates of all tasks and the `exec_config` tool schema (Section B), including the most-similar matching rule. **(3) Training**: the trajectory-simulation and policy-distillation data, the LlamaFactory and verl configurations (Section B.3), and the trained Qwen3-1.7B and Qwen3-4B agents. **(4) Evaluation**: the evaluation harness implementing the per-budget protocol of Section 5, the prompt-only API-agent harness, the adapted NAP/MetaBO/FSBO configurations, and the Random Search and Top-K Warm-Start scripts (Section B.4), together with the per-episode transcripts and scores of every run reported in the paper; the scripts that produce all figures and Tables 11–15 and 16–17 from these logs are included. **(5) GRPO sweep logs**: the W&B logs of the in-house Task 3 runs (Section A). The Qwen agents, the meta-training baselines, Random Search, and Top-K Warm-Start are fully reproducible from the released code and seeds. The frontier API agents are external services whose behavior may change over time; we therefore date the runs (Section B.4) and release their complete transcripts so that all reported numbers can be verified and the same protocol re-run on newer models.