HARA: Hierarchically Aligned Reference-Answer Reinforcement Learning for Generative Tool Routing

Anonymous ACL submission

Abstract

Large tool catalogs are often hierarchical, but tool routing systems commonly treat them as flat candidate sets. This mismatch is costly: once a model chooses the wrong high-level branch, the correct leaf tool may become unreachable. We study generative hierarchical tool routing, where a model first predicts a tool path and then retrieves leaf tools under the predicted sub-tree. We introduce HARA, a hierarchy-aligned reference-answer reinforcement learning framework that trains a router directly from the base model, without a cold-start supervised training stage or a curated corpus of successful routing traces. HARA makes this possible by coupling two training signals: reference-answer-conditioned posterior rollouts provide exploration when prior rollouts rarely reach useful branches, and level-wise path rewards assign credit at hierarchy-node boundaries rather than only at the terminal leaf. At inference time, the generated path localizes a sub-tree before dense retrieval and reranking. This lets HARA skip cold-start supervised training entirely: reference answers provide exploration signal without first constructing a cold-start corpus of successful routing trajectories. On FHTC-26K, a financial benchmark with 26K tools, HARA improves Qwen3-8B Pass@5 from 33.10% to **70.23%**. On ToolBench, HARA improves Recall@5 from 38.81% to **64.49%**. Ablations show that cold-start-free posterior exploration and hierarchy-compatible rewards are both necessary for stable reinforcement learning.

1 Introduction

Tool-augmented large language models (LLMs) increasingly operate over large API catalogs (Schick et al., 2023; Patil et al., 2023; Qin et al., 2024). In many deployed settings, especially finance, these catalogs are not flat lists: tools are organized by asset class, analysis type, statement category, provider, and leaf-level API. This structure makes routing errors asymmetric. A model that chooses the wrong high-level branch can exclude the correct tool from all downstream candidates, even when its local understanding of the query is mostly correct.

Most tool-selection and retrieval-augmented generation pipelines do not model this asymmetry (Lewis et al., 2020; Gao et al., 2024; Qu et al., 2024). Exact matching requires the model to reproduce catalog strings, making it brittle to paraphrases and path-format errors. Global dense retrieval is more tolerant, but searches all tools at once and therefore ignores the routing decisions encoded by the hierarchy. We instead study *generative hierarchical tool routing*: the model generates an interpretable path in the tool tree, and this path constrains retrieval to a localized sub-tree.

Learning such a router is difficult without a cold start. A natural approach is to first fine-tune on successful routing traces and then apply reinforcement learning. This can work, but constructing the SFT data is expensive: each example must pair a query with a correct reasoning trace that justifies the branch decisions. This process must be repeated when catalogs are reorganized or expanded. It also makes each new tool hierarchy depend on a supervised cold-start corpus and encourages the model to imitate catalog strings rather than learn robust routing behavior. Starting reinforcement learning (RL) directly from the base model is more attractive, but early prior rollouts rarely reach the right branch in a 26K-tool tree, and most rollout groups contain little useful signal.

The second challenge is that ordinary scalar rewards are poorly matched to hierarchical actions. In a tree, a leaf-level mistake should not erase all credit for a correct prefix; conversely, an early branch error should be penalized because it changes the reachable search space. Stable training therefore requires rewards that are dense enough for RL but still faithful to the hierarchy.

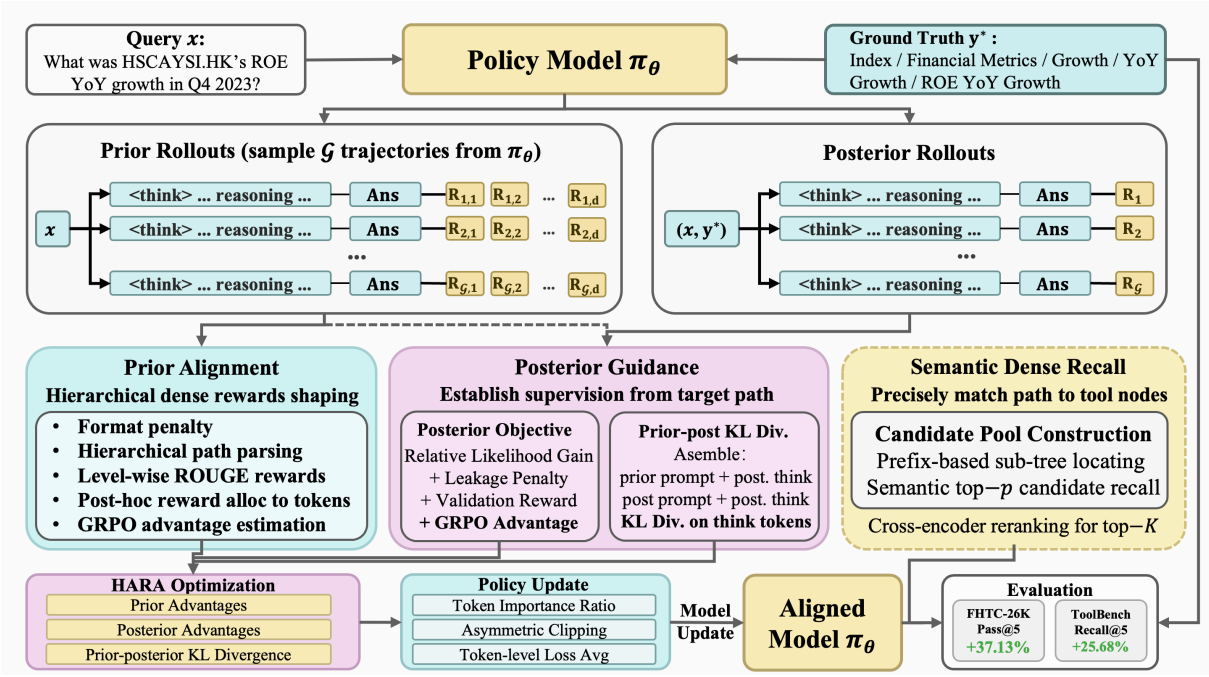

Figure 1: Overview of HARA. The model generates a hierarchical path, uses posterior rollouts and level-wise rewards during training, and uses the predicted path to retrieve tools at inference time.

The reward should say not only whether a tool is right, but also which routing decisions are already aligned with the reference path.

We introduce HARA, a hierarchy-aligned framework for training and using generative routers. As summarized in Figure 1, HARA couples three mechanisms. First, reference-answer-conditioned posterior rollouts provide high-signal exploration when prior rollouts rarely reach the correct branch (Lin et al., 2025); these rollouts are used only during training, so the deployed router remains a prior model conditioned only on the user query. Second, rewards are assigned at hierarchy-node boundaries, giving each routing decision a structured return while preserving downstream credit through reward-to-go. Third, the generated path is used as a semantic anchor for sub-tree localization, dense retrieval, and reranking (Karpukhin et al., 2020; Chen et al., 2024; Nogueira and Cho, 2019).

We evaluate HARA on FHTC-26K, a financial benchmark with 26K tools, and on ToolBench, a public benchmark of open-domain APIs. HARA improves Qwen3-8B Pass@5 on FHTC-26K from 33.10% to 70.23%; on ToolBench, it improves Recall@5 from 38.81% to 64.49%. Ablations show that posterior guidance, hierarchy-aware rewards, and path-constrained retrieval each contribute to the final system.

Our contributions are:

- We formulate large-catalog tool routing as hierarchical path generation followed by path-constrained retrieval.
- We introduce a cold-start-free RL recipe that uses reference-answer-conditioned posterior rollouts to explore useful branches without SFT warm-up.
- We design hierarchy-compatible path rewards that attach credit to node decisions and stabilize RL in a large tool tree.
- We show that aligning both training and retrieval with the hierarchy improves end-to-end routing on private financial and public tool benchmarks.

2 HARA

2.1 Hierarchical Tool Routing

HARA trains a generative router and uses its generated path to constrain retrieval, as shown in the path-generation and retrieval stages of Figure 1. Let $\mathcal{T}$ be a rooted tool tree. Each leaf tool t induces a path $p(t) = (l_1, \ldots, l_{K_t})$, where the depth K_t may vary across tools. For a query x, the training data provides a reference answer y_{ans} identifying a target leaf t^* and path $p^* = p(t^*)$.

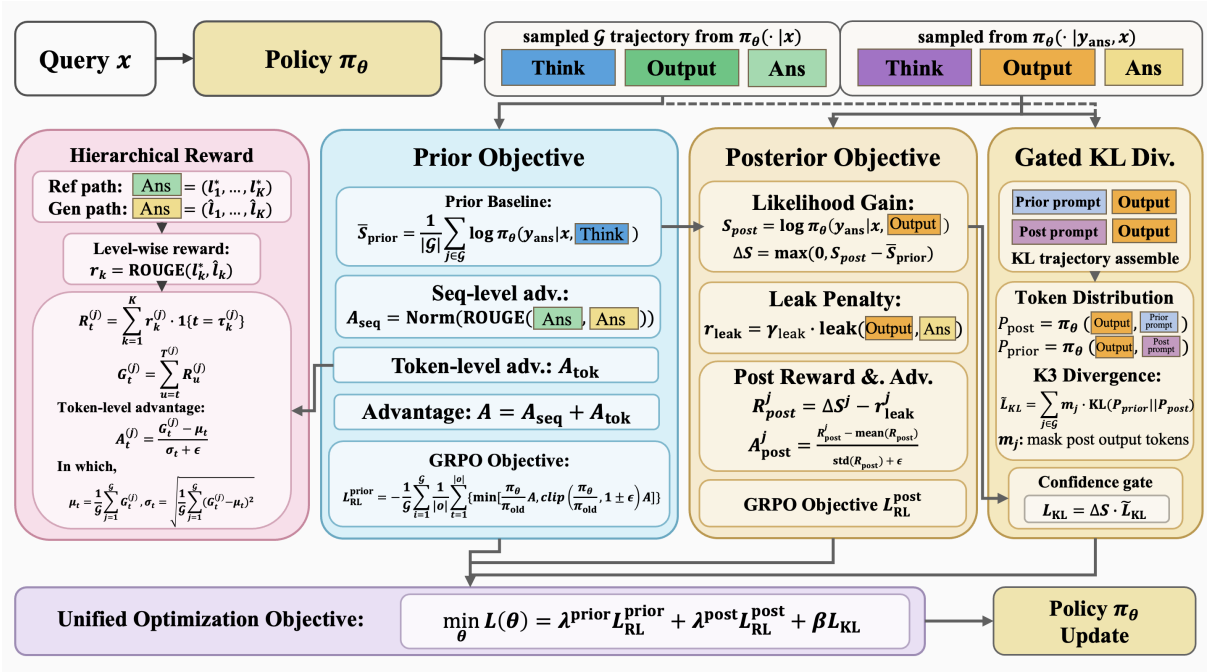

Figure 2: HARA optimization. Prior rollouts receive level-wise path rewards at hierarchy-node boundaries; posterior rollouts guide the prior only when they improve reference-answer likelihood without leaking the answer. The gated KL term transfers posterior knowledge to the prior policy by distilling reference-answer-conditioned token distributions.

The prior policy generates a routing output from the query,

$$o^{\text{prior}} \sim \pi_\theta(\cdot \mid x),$$

while the posterior policy is sampled only during training,

$$o^{\text{post}} \sim \pi_\theta(\cdot \mid x, y_{\text{ans}}).$$

A deterministic parser maps each output to a predicted path $\hat{p} = (\hat{l}_1, \ldots, \hat{l}_{\hat{K}})$. Malformed outputs receive zero path reward. At inference time, only the prior policy is used, and $\hat{p}$ localizes a candidate sub-tree before leaf-tool retrieval.

2.2 Cold-Start-Free Hierarchical RL

HARA is designed to train from the base policy rather than from an SFT warm start. This is the setting that makes hierarchical routing hard: early prior samples may contain reasonable semantic fragments, but they often miss the correct top-level branch and therefore receive little terminal credit. HARA avoids this cold-start bottleneck by sampling two kinds of trajectories in each update. Prior trajectories are conditioned only on the query and define the deployed behavior. Posterior trajectories are conditioned on the query and reference answer, and are used only as training-time exploration proposals.

Posterior exploration without answer leakage. The posterior is useful because the reference answer often reveals which part of the hierarchy is relevant. However, a posterior rollout is not automatically a valid teaching signal: it may copy answer strings, overfit to the reference, or propose a path that cannot be recovered by the prior at test time. HARA therefore treats posterior samples as candidates for transfer rather than as gold demonstrations. Building on recent advances in answer-conditioned exploration (Lin et al., 2025), for a generated derivation z, we score how much it makes the reference answer predictable under the prior input:

$$S(z) = \log \pi_\theta(y_{\text{ans}} \mid x, z),$$

where the reference answer is scored as the target and is not included in the scoring context. The posterior gate is

$$\bar{S}_{\text{prior}} = \frac{1}{|\mathcal{G}|}\sum_{j\in\mathcal{G}} S(z^{\text{prior},j}),$$

$$\Delta S = \max(0, S(z^{\text{post}}) - \bar{S}_{\text{prior}}).$$

The margin compares a posterior rollout against the average prior rollout for the same query, corresponding to the posterior gate in Figure 2. If the

posterior is not more useful than what the prior already produces, it is not transferred. Posterior trajectories are rewarded by

$$r_{\text{post}} = \Delta S - \gamma_{\text{leak}} \ell_{\text{leak}}(o^{\text{post}}, y_{\text{ans}}),$$

where ℓ_{leak} penalizes direct answer copying, and γ_{leak} is a hyperparameter that controls the strength of this leakage penalty during training. Accepted posterior tokens are then distilled into the prior with a gated KL term. This gives HARA a way to discover useful branches before the prior can reach them by itself, while keeping inference identical to ordinary query-only routing.

Rewards aligned with hierarchy nodes. Posterior exploration alone does not solve credit assignment. A terminal reward can say whether a retrieved tool worked, but it cannot distinguish a nearly correct path from a completely wrong branch, which causes high variance and instability during RL. To stabilize training, we adopt the level-wise path reward structure introduced by (Anonymous, 2026), implementing it specifically as a reward design on the prior side. This scores routing decisions at the level where they occur. For each matched level $k \leq \min(\hat{K}, K^*)$, we compute

$$r_k = \text{sim}(\hat{l}_k, l_k^*),$$

where sim is ROUGE-L over normalized node labels in our implementation (Lin, 2004); unmatched levels receive zero reward. If τ_k is the decoding boundary of node $\hat{l}_k$, the token reward and return are

$$R_t = \sum_k r_k \, \mathbf{1}\{t = \tau_k\}, \quad G_t = \sum_{u=t}^{T} R_u.$$

This construction corresponds to the path-reward block in Figure 2, and makes the reward compatible with the action structure: correct high-level branches receive immediate credit, leaf mistakes do not erase prefix progress, and early branch errors still reduce the return for subsequent tokens. We use group-relative normalization over rollouts for the same query and optimize the policy with GRPO (Shao et al., 2024).

Joint objective. Figure 2 shows that HARA has three optimization blocks. The prior RL term updates the deployed query-only policy using the level-wise returns above. The posterior RL term trains the answer-conditioned branch with the leakage-aware reward r_{post}, so that it remains a useful explorer rather than a fixed teacher. The KL term couples the two prompt conditions on accepted posterior reasoning tokens: it transfers information discovered under the posterior prompt to the prior, while also discouraging the posterior branch from drifting too far from query-only behavior.

For a posterior reasoning trajectory z^{post}, let m_t indicate non-leaking reasoning tokens selected for KL alignment. We compare the prior-prompt and posterior-prompt token distributions on the same posterior reasoning prefix:

$$\begin{aligned} p_t^{\text{prior}} &= \pi_\theta(\cdot \mid x, z_{<t}^{\text{post}}), \\ p_t^{\text{post}} &= \pi_\theta(\cdot \mid x, y_{\text{ans}}, z_{<t}^{\text{post}}), \end{aligned}$$

where both distributions are produced by the current policy under different prompts. The gated KL loss is

$$\begin{aligned} \widetilde{\mathcal{L}}_{\text{KL}} &= \frac{1}{\sum_t m_t} \sum_t m_t \, \text{KL}(p_t^{\text{post}} \| p_t^{\text{prior}}), \\ \mathcal{L}_{\text{KL}} &= \Delta S \, \widetilde{\mathcal{L}}_{\text{KL}}. \end{aligned}$$

Thus posterior information affects the prior only when the posterior rollout improves the answer-likelihood margin, and the posterior branch is simultaneously regularized toward behavior that the prior can realize from the query alone. The final loss combines the three terms:

$$\mathcal{L} = \lambda^{\text{prior}} \mathcal{L}_{\text{RL}}^{\text{prior}} + \lambda^{\text{post}} \mathcal{L}_{\text{RL}}^{\text{post}} + \beta \mathcal{L}_{\text{KL}}.$$

Full estimator details are given in Appendix A.

2.3 Path-Constrained Retrieval

Generated paths are interpretable but need not exactly match catalog strings. HARA therefore treats $\hat{p}$ as a semantic routing anchor, matching the inference-side flow in Figure 1. We first compare the generated high-level prefix with canonical catalog prefixes to localize candidate sub-trees. Within the localized pool, a dense bi-encoder retrieves leaf tools using the full generated path, and a cross-encoder reranks the top candidates:

$$\begin{aligned} S_{\text{leaf}}(t) &= \cos(E_d(\hat{p}), E_d(t)), \\ S_{\text{rank}}(t) &= E_c(\hat{p}, t). \end{aligned}$$

This retrieval stage lets the policy learn structural routing decisions without requiring exact reproduction of every leaf-tool string.

The retrieval design is also part of the training story. If inference used exact matching, a semantically correct path could fail because of minor label variation. If inference used only global dense retrieval, the model could ignore high-level branch structure and still recover some tools by semantic similarity. HARA uses a middle path: generation is responsible for selecting a plausible anchor, while dense retrieval and reranking handle paraphrase and catalog-string variation.

This alignment is important for stability. During training, level-wise rewards teach the model which prefixes are useful; during inference, those prefixes determine the candidate pool. As a result, a better prefix has a direct operational consequence: it narrows retrieval to a more relevant sub-tree. Conversely, a wrong branch is not hidden by a global retriever. This makes improvements in path similarity meaningful for final Pass@K, and it explains why HARA combines training-time hierarchy rewards with inference-time sub-tree retrieval rather than treating them as independent modules.

3 Experiments

Benchmarks. FHTC-26K (Anonymous, 2026) is a private financial benchmark with 26,314 variable-depth tool paths, 98,964 training queries, 400 validation queries, and 5,251 test queries. Each training query has a canonical target path. At evaluation time, a Top-K prediction is successful if any retrieved tool can fully answer the user query. ToolBench (Qin et al., 2024) evaluates transfer to public open-domain API hierarchies. Because ToolBench queries may have multiple valid tools, we report mean Recall@K over the annotated tool set.

Models and baselines. All trainable systems start from Qwen3-8B (Qwen Team, 2025). We compare the base model with exact edit-distance matching, the base model with HARA retrieval, SFT on reference paths, a RAVR-style posterior-guided variant (Lin et al., 2025), and the full HARA system. HARA is not initialized from the SFT model; it trains directly from the base policy, so the comparison to SFT tests whether hierarchy-aligned RL can replace a supervised cold-start stage rather than merely refine it. Retrieval uses BGE-M3 for dense recall and BGE-Reranker-v2-M3 for reranking (Chen et al., 2024; BAAI, 2024). Full training and retrieval hyperparameters are in Appendices B and C.

Metrics. On FHTC-26K, we report LLM-evaluated Pass@K using Qwen3-235B-Instruct as the judge under the LLM-as-a-Judge paradigm (Zheng et al., 2023). A query is counted as solved if at least one candidate in the Top-K set is judged sufficient. We use 10 independent judgments per candidate set and accept tools whose approval rate exceeds 0.66. On ToolBench, we report mean Recall@K against released annotations: for annotated answer-tool set $\mathcal{A}$ and retrieved Top-K set $\mathcal{C}_K$, $\text{Recall@}K = |\mathcal{C}_K \cap \mathcal{A}|/|\mathcal{A}|$, averaged over queries.

4 Results

4.1 End-to-End Routing

Table 1 reports end-to-end routing results. Exact matching is brittle: replacing edit distance with semantic retrieval raises FHTC-26K Pass@5 from 33.10% to 49.57%. SFT provides a strong supervised cold-start baseline at 65.07%, but HARA further improves Pass@5 to 70.23% while training directly from the base policy. This is the main empirical point: HARA does not rely on an SFT warm-up to make RL viable. Instead, posterior rollouts create early exploration signal and hierarchy-compatible rewards turn partial paths into stable credit.

The same pattern appears on ToolBench. HARA improves Recall@5 from 46.69% to 64.49%, despite being designed for hierarchical routing in the financial domain. The gain over SFT is smaller than on FHTC-26K, which is expected because ToolBench contains more heterogeneous public APIs and multiple acceptable tools per query. Still, the improvement suggests that HARA learns a reusable routing behavior: generate a plausible structured path, use it to narrow the candidate space, and recover the final tool with semantic retrieval.

4.2 Frontier Models and Scaling

Tables 2 and 3 address two separate questions. Table 2 asks whether larger untuned models solve hierarchical routing under the same retrieval pipeline. They still lag behind HARA, suggesting that hierarchical routing is not solved by scale alone. Larger models can often verbalize tool categories, but without training pressure aligned to the tree they still confuse branch decisions or generate paths that are hard to map back to catalog leaves.

Table 1: End-to-end routing results. All trainable methods start from Qwen3-8B. Except for edit-distance matching, systems use the same semantic retrieval pipeline.

Methods	FHTC-26K (%)			ToolBench (%)		
	Pass@1	Pass@3	Pass@5	Recall@1	Recall@3	Recall@5
Qwen3-8B + edit distance	16.99	27.60	33.10	25.19	35.26	38.81
Qwen3-8B + retrieval	32.39	43.60	49.57	27.05	40.69	46.69
Qwen3-8B + SFT	44.94	59.80	65.07	31.76	54.72	61.35
Qwen3-8B + RAVR	37.31	54.22	57.19	35.18	53.84	62.91
Qwen3-8B + HARA	**49.48**	**67.05**	**70.23**	**37.41**	**56.99**	**64.49**

Table 2: Frontier-model comparison on FHTC-26K. All models use the same retrieval pipeline without HARA training.

Model	Pass@5
Qwen3-Plus	45.02
DeepSeek-V3.2	50.40
MiniMax-M2.7	41.83

Table 3: Scaling HARA across Qwen3 base-model sizes on FHTC-26K.

Model	Base@1	Base@5	HARA@1	HARA@5
Qwen3-1.7B	23.52	43.50	25.50	45.86
Qwen3-4B	33.54	46.69	38.37	61.89
Qwen3-8B	32.39	49.57	49.48	70.23

Table 4: Grouped ablations on FHTC-26K. Rows are grouped by the controlled factor: reference systems, training signal, and inference-time path-to-tool mapping.

System	Pass@1	Pass@5
Reference systems		
Base + retrieval	32.39	49.57
SFT + retrieval	44.94	65.07
Training signal, same retrieval		
GRPO + retrieval	35.55	53.53
Posterior-only + retrieval	37.31	57.19
Inference mapping, same HARA policy		
HARA + edit distance	40.26	62.75
HARA + textual routing	42.85	63.17
HARA	**49.48**	**70.23**

Table 3 instead asks how HARA scales across Qwen3 base models. HARA improves each base model, with the largest gains at 8B. This pattern is consistent with the cold-start story: the method does not need a supervised warm-start policy, but it benefits from a stronger base model that can produce more plausible posterior reasoning under the reference answer. Better posterior reasoning gives HARA higher-quality exploration proposals, and level-wise rewards can then convert those proposals into larger prior-policy gains.

4.3 Ablation Studies

Table 4 separates three comparisons that were otherwise easy to conflate. The first block provides reference systems under the same semantic retrieval pipeline. SFT is a strong supervised cold-start baseline, but it still remains below full HARA. The second block controls the retrieval pipeline and changes only the training signal. GRPO with coarse rewards gives only modest gains over the base retriever, while posterior-only training improves exploration but remains below SFT. This suggests that answer-conditioned rollouts help only when their signal is grounded in the hierarchy. HARA performs best because posterior rollouts overcome the no-cold-start exploration problem and level-wise rewards stabilize credit assignment.

The third block fixes the HARA policy and changes only how generated paths are mapped back to tools. HARA + edit distance already improves over the base model, which means the learned policy generates more useful paths. However, edit distance still requires surface-form agreement with catalog strings. Textual routing is more flexible but remains global. Full HARA performs best because sub-tree-constrained semantic retrieval respects the same structure used during training: high-level generation narrows the candidate space, and dense retrieval resolves leaf-level paraphrase and formatting variation.

4.4 Training Dynamics

Figure 3 shows why the reward design matters for stable training. Posterior guidance improves

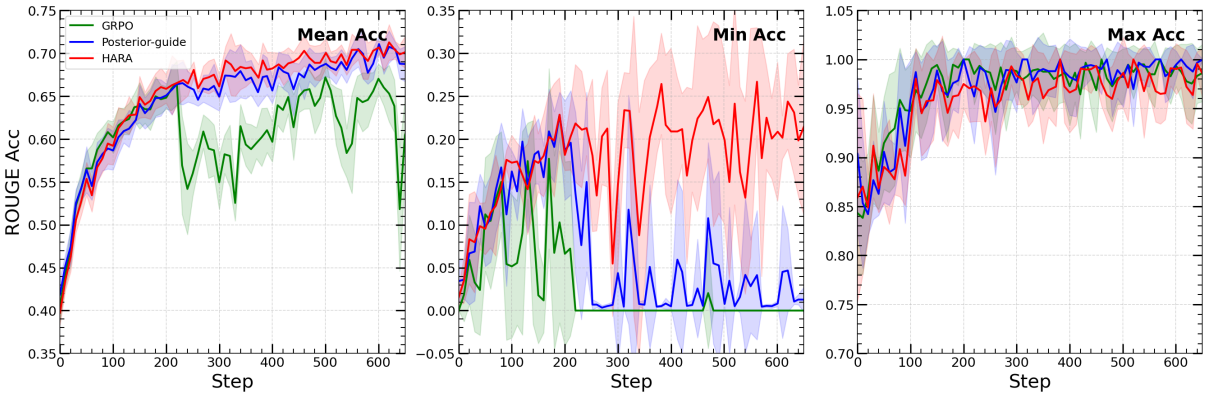

Figure 3: Training dynamics on FHTC-26K. HARA reaches higher and more stable path similarity than GRPO and posterior-only training, supporting the role of hierarchy-aware credit assignment.

early exploration because it gives the model access to answer-conditioned trajectories before the prior policy can reliably reach the correct branch. But when posterior guidance is used without hierarchy-aware credit assignment, training is less stable: useful posterior trajectories are not consistently converted into prior routing behavior. HARA maintains higher path similarity over training, supporting the claim that level-wise rewards turn posterior exploration into stable prior-policy improvement.

This distinction is important for large tool hierarchies. In a flat action space, a single success reward may be sufficient once exploration finds the right tool. In a tree, the model must learn a sequence of branch decisions, and different mistakes have different consequences. HARA's reward placement makes those consequences visible to the optimizer: correct prefixes remain useful, malformed or wrong-branch paths receive little return, and posterior information is transferred only when it improves answer likelihood without leakage. The resulting training curve is therefore not just a performance artifact; it is evidence that the reward is better adapted to hierarchical tool routing.

4.5 Why Does HARA Avoid Cold Start?

The main empirical contrast in Tables 1 and 4 is between three ways of obtaining a useful initial routing policy. SFT constructs one by imitation: the model learns to reproduce reference paths before any reinforcement learning signal is applied. This is effective, but it makes the router depend on a supervised warm-start stage and can over-emphasize surface-form path reproduction. Coarse GRPO takes the opposite approach and starts RL directly, but its reward is too sparse for a large hierarchy. HARA occupies a middle ground: it starts from the base policy like direct RL, but uses posterior trajectories to expose the model to plausible branches early in training.

This posterior signal is weaker than a demonstration. The posterior rollout is not copied as a gold path, and it is not used at inference time. It must first outperform the average prior rollout under the answer-likelihood margin, and it is penalized for answer leakage. This matters because answer-conditioned generation can otherwise teach shortcuts that disappear at test time. In HARA, posterior trajectories mainly serve to expand the support of exploration: they make it more likely that the optimizer observes branch names, intermediate categories, and reasoning fragments that are relevant to the reference answer. The prior policy still has to learn to produce those decisions from the query alone.

The level-wise reward then determines which parts of this expanded exploration are retained. A posterior trajectory may mention the right sector but choose the wrong financial statement, or select the right statement family but miss the final provider-specific API. A terminal reward treats these cases similarly if the final leaf is wrong. HARA instead gives credit at node boundaries, so the optimizer can preserve useful prefixes while correcting later decisions. This is the mechanism that turns no-cold-start exploration into stable training rather than noisy imitation.

The ablations support this interpretation. Posterior-only training improves over coarse GRPO, showing that answer-conditioned rollouts help exploration. But it remains far below full HARA, showing that exploration without

hierarchy-compatible credit assignment is insufficient. Conversely, HARA with weaker retrieval still improves over the base system, showing that the trained router itself is better aligned with the tool tree. The full system performs best when the same hierarchy is used in all three places: posterior exploration, reward assignment, and sub-tree-constrained retrieval.

4.6 Implications for Large Tool Catalogs

The no-cold-start property is especially useful when a tool catalog is large, proprietary, or frequently revised. In such settings, it is often feasible to collect query-answer pairs or identify the tool that produced an answer, but expensive to curate high-quality reasoning traces that teach the model how to navigate the hierarchy. A supervised warm-start policy also becomes stale when categories are renamed, providers are reorganized, or new leaf APIs are inserted. HARA reduces this dependency: the posterior uses available reference answers to generate exploration proposals, while the prior learns only the query-conditioned routing behavior that will be used at deployment.

The reward design also gives a more graceful failure mode than flat tool selection. A flat terminal reward treats the routing decision as all-or-nothing, so training can oscillate between superficially plausible tools with little indication of which branch choice caused the error. HARA exposes intermediate decisions. If the model selects the right market category but the wrong statement type, the prefix receives credit and the later node can still be corrected. If it chooses the wrong market category, the low return appears immediately at the branch where the error matters. This makes the optimization signal closer to the operational structure of a real catalog.

5 Related Work

Tool routing and retrieval. Tool-augmented LLMs commonly select tools from flat candidate pools or retrieve APIs using query-tool similarity (Patil et al., 2023; Qin et al., 2024; Qu et al., 2024). These designs are effective when tool spaces are small or weakly structured, but they ignore the branch decisions in large tool trees. HARA instead learns an explicit path and uses that path to constrain retrieval, combining interpretability from generation with robustness from dense retrieval.

RL with structured feedback. RL has been used to improve LLM reasoning, tool use, and retrieval policies (Shao et al., 2024; DeepSeek-AI, 2025; Acikgoz et al., 2026), but sparse terminal rewards are difficult in large discrete action spaces and often motivate supervised cold-start stages. Answer-conditioned or posterior trajectories can improve exploration, while process-level rewards provide better credit assignment (Lin et al., 2025; Lightman et al., 2023). HARA combines these ideas for hierarchical tool routing: posterior rollouts make direct-from-base RL feasible, and level-wise path rewards make the optimization signal compatible with tree-structured tools.

6 Conclusion

We introduced HARA, a hierarchy-aligned framework for generative tool routing. HARA trains a router directly from the base model by using posterior-guided exploration to avoid an SFT cold-start stage and level-wise path rewards to stabilize credit assignment in a tool tree. At inference time, generated paths constrain semantic retrieval to localized sub-trees. Experiments on FHTC-26K and ToolBench show that aligning exploration, rewards, and retrieval with the hierarchy improves end-to-end routing over flat matching, SFT, and prior-only RL variants.

7 Limitations

HARA trades supervision cost for training-time computation and engineering complexity. Each update samples both prior and posterior rollout groups, roughly doubling the cost of autoregressive generation. During the optimization phase, computing the gated KL regularization requires evaluating the posterior reasoning trajectories under both prior and posterior prompt conditions. This additional forward passing substantially increases the overall computational overhead and peak memory usage. It also requires modest changes to mainstream GRPO pipelines, which usually assume one prompt and one rollout group per query. Our evaluation is currently limited to two datasets. Furthermore, to skip the cold-start phase, HARA trades bias for variance: posterior guidance reduces the high variance of cold-start exploration but introduces a localized bias, meaning the performance may not reach the upper bound of a SFT+RL paradigm initialized with high-quality human demonstrations.

References

Emre Can Acikgoz, Cheng Qian, Jonas Hübotter, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. 2026. Tool-r0: Self-evolving llm agents for tool-learning from zero data. *arXiv preprint arXiv:2602.21320*.

Anonymous. 2026. Navigating large hierarchical tool spaces via causally-decoupled advantage policy optimization. ACL ARR 2026 January Submission, OpenReview.

BAAI. 2024. Bge-reranker-v2-m3 model card. https://huggingface.co/BAAI/bge-reranker-v2-m3.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand. Association for Computational Linguistics.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yutao Bi, Yi Dai, Jiawei Sun, Meng Wang, and Wang Haofen. 2024. Retrieval-augmented generation for large language models: A survey. *SIGKDD Explorations*, 26(1):1–22.

Vladimir Karpukhin, Barlas Ouz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781. Association for Computational Linguistics.

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

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, Karl Cobbe, Ilya Sutskever, and John Schulman. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

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

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

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

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Tool learning with large language models: A survey. *arXiv preprint arXiv:2405.17935*.

Qwen Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *arXiv preprint arXiv:2306.05685*.

A Training Algorithm

Algorithm 1 summarizes the overall optimization procedure of HARA. In each iteration, we sample a batch of queries and their corresponding reference answers from the training set. For each query, the current policy π_θ generates a group of prior trajectories conditioned only on x and a group of posterior trajectories conditioned on (x, y_{ans}). The prior group is used to compute hierarchy-aware structured returns and token-level advantages, while the posterior group is used to compute the leakage-aware posterior reward and the confidence margin ΔS.

We then evaluate the gated posterior-to-prior KL regularization on posterior evaluable outputs.

When the posterior trajectory is more confident than the average prior baseline, the KL term is activated to transfer posterior information into the prior policy; otherwise, the KL term is suppressed. Finally, the prior RL loss, posterior RL loss, and gated KL regularization are aggregated, and the policy parameters θ are updated by gradient descent.

Algorithm 1 HARA Optimization

Require: Policy π_θ, training set $\mathcal{D}$, group size $\mathcal{G}$, hyperparameters $\lambda^{\text{prior}}, \lambda^{\text{post}}, \beta$

while not converged **do**
  Sample a batch of query-answer pairs $(x, y_{\text{ans}}) \sim \mathcal{D}$
  for each (x, y_{ans}) in the batch **do**
    Sample $\mathcal{G}$ prior trajectories $(z^{\text{prior}}_{\text{think}}, o^{\text{prior}}) \sim \pi_\theta(\cdot \mid x)$
    Sample $\mathcal{G}$ posterior trajectories $(z^{\text{post}}_{\text{think}}, o^{\text{post}}) \sim \pi_\theta(\cdot \mid x, y_{\text{ans}})$
    Parse y^{prior} into hierarchical paths and compute level-wise rewards r^{prior}_k
    Compute prior token rewards R^{prior}_t, reward-to-go G^{prior}_t, and token-level advantages $A^{\text{prior}}_{\text{tok},t}$
    Compute prior baseline score $\bar{s}_{\text{prior}}$
    Compute posterior leakage penalty r_{leak} for o^{post}
    Compute posterior confidence score S_{post} and gating margin $\Delta S = \max(0, S_{\text{post}} - \bar{S}_{\text{prior}})$
    Compute posterior reward $r_{\text{post}} = \Delta S - r_{\text{leak}}$
    Compute posterior-to-prior KL term $\widetilde{\mathcal{L}}_{\text{KL}}$ on post-thought tokens of o^{post}
    Apply confidence-aware gating: $\mathcal{L}_{\text{KL}} = \Delta S \cdot \widetilde{\mathcal{L}}_{\text{KL}}$
  end for
  Compute prior loss $\mathcal{L}^{\text{prior}}_{\text{RL}}$ from prior trajectories
  Compute posterior loss $\mathcal{L}^{\text{post}}_{\text{RL}}$ from posterior trajectories
  $\mathcal{L}(\theta) = \lambda^{\text{prior}}\mathcal{L}^{\text{prior}}_{\text{RL}} + \lambda^{\text{post}}\mathcal{L}^{\text{post}}_{\text{RL}} + \beta\mathcal{L}_{\text{KL}}$
  Update θ by minimizing $\mathcal{L}(\theta)$
end while

B Training Parameters

This section summarizes the key hyperparameter settings used during reinforcement learning. HARA is trained on a single compute node with 8 NVIDIA H100 (80GB) GPUs. To balance the high-throughput generation stage and the memory-intensive optimization stage, we decouple hardware allocation across inference and training. Specifically, 4 GPUs are dedicated to the vLLM engine for actor rollout generation, while the remaining 4 GPUs are used by the Megatron-LM backend for actor optimization and reference-model computation, with tensor parallelism set to 2. All models are run in bfloat16 precision.

During rollout generation, we sample $G = 8$ trajectories per query with decoding temperature 1.0 and top-$p = 0.9$. The maximum prompt length and maximum response length are both set to 3072 tokens. For optimization, we use the AdamW optimizer with a peak learning rate of 1×10^{-6}, 20 warmup steps, and zero weight decay. Training is conducted for 50 epochs with a per-device batch size of 8 and 8 gradient accumulation steps. To stabilize RL in the large hierarchical routing space, the KL penalty coefficient β is set to 0.001, and we perform only 1 PPO epoch per iteration to avoid overfitting to the current rollout buffer. The policy clipping range is set to 0.2, while advantages and rewards are clipped at 2.0 and 10, respectively, to suppress extreme updates and reduce the risk of policy collapse.

C Retrieval Parameters

To map the generated routing path to the large action space of 26,314 tools, we use a three-stage retrieval pipeline based on the BGE-M3 bi-encoder and the BGE-Reranker-v2-M3 cross-encoder. The pipeline first extracts the top two levels of the generated path for prefix-based sub-tree localization. If an exact prefix match is found, the candidate pool is restricted to the corresponding sub-tree. Otherwise, the system falls back to dense prefix retrieval, selecting the top-5 most semantically similar prefix branches using softmax-normalized scores with nucleus threshold $P = 0.95$ and temperature $\tau = 1.0$. If no candidate pool is obtained after this step, the retrieval space reverts to the full catalog of 26,314 tools.

Within the resulting candidate pool, the bi-encoder performs coarse-grained recall by computing cosine similarity between the full generated path and candidate tool representations, and returns the top-50 tools. The maximum sequence length is set to 256 and the embedding batch size is 64. The cross-encoder then performs fine-grained

reranking over these top-50 candidates, scoring semantic interactions between the generated path and each candidate tool description with a reranker batch size of 64, and produces the final top-5 list.

All retrieval embeddings are precomputed in fp16 precision and cached with SHA-1 verification. For the final Pass@K evaluation on FHTC-26K, we use Qwen3-235B-Instruct as the LLM judge through vLLM with 8-way tensor parallelism and bfloat16 precision. The judge is run with a relevance threshold of 0.66 and decoding temperature 0.3 to ensure stable and reproducible evaluation.