STRUCTURING MOE EXPERT SELECTION FOR AGENTIC REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon LLM agents are frequently implemented using sparse mixture-of-experts (MoE) models, yet the co-design of agentic behavior and MoE structures remains underexplored. In this work, we comprehensively study the connections between agentic post-training and MoE expert selection. In off-the-shelf MoE models, we observe expert selection exhibits a specialized structure that naturally aligns with agentic trajectories. Specifically, expert routing overlaps more between turns where the agent performs semantically similar operations (e.g., READ, UPDATE) than between turns with differing operations. However, standard RL algorithms ignore this specialization, allowing the MoE routing to go uncontrolled during training, which empirically limit task performance and inference efficiency. To address this, we introduce a hierarchical routing control framework for agentic tasks. We explicitly encourage turn-level expert selections to align with agentic operations while regularizing token-level expert selections to maintain local consistency. To resolve stability issues that arise during post-training with the proposed methods, we further introduce an entropy-gated control mechanism. Overall, our routing control framework achieves over 10-point improvements in success rate on all evaluated benchmarks. These results demonstrate that agentic trajectory structure provides an effective signal for optimizing MoE capacity during RL post-training.

1 INTRODUCTION

Recent progress in long-horizon interactive agents has been driven by two complementary advances. First, external scaffolding built around LLMs enables complex agentic behaviors, such as multi-step reasoning and making external tool calls (Yao et al., 2023; Trivedi et al., 2024; Shepard & Salimans, 2026). Second, mixture-of-experts (MoE) architectures allow model capacity to scale efficiently by activating only a fraction of parameters per token (Liu et al., 2024). This sparse computation is particularly useful for agentic workloads, which generate long trajectories through repeated interactions with environments. Although long-horizon agents are frequently implemented using MoE architectures (Liu et al., 2025; Team et al., 2026; Zeng et al., 2026), the co-design of agentic frameworks and MoE routing remains underexplored. While current agentic post-training algorithms optimize the actor policy for downstream performance, they fail to account for how these updates affect expert selection in MoE models.

Agentic trajectories provide a natural structure for understanding this co-design. A trajectory repeatedly alternates between a *thinking* field, a *tool-use* field, and *environment feedback*. We encapsulate the three fields as one *turn* in an agentic trajectory. Semantically, a turn performs an identifiable operation, such as making tool calls that read, create, or update an object. We observe that the routers of MoE models already exhibit an operation-grouped specialization structure: adjacent tokens within one field tend to reuse experts, while turns with semantically similar operations tend to share similar expert distributions. However, this structure is overlooked in standard MoE-agnostic post-training. Standard RL does not encourage consistency between adjacent tokens or operation-grouped specialization, which limits both task performance and efficiency of MoE-based long-horizon agents, as demonstrated in Fig. 1.

In this work, we hypothesize that a well-structured expert selection mechanism is key to improving agentic task performance. Specifically, aligning expert routing with the semantic structure of the

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.

trajectory—by maintaining consistency within a *thinking* or *tool-use* field, and specializing experts for distinct operations—allows the model to utilize its expert capacity more effectively. To explicitly optimize this structure, we introduce a hierarchical routing control framework for expert selection during agentic RL. At the turn level, we maximize the mutual information (MI) between the router distribution, averaged across all tokens within a respective *thinking* or *tool-use* field in the turn, and the corresponding operation label for that turn. This encourages turns with the same semantic operation to share expert sets while separating the experts used for different operations. At the token level, we regularize adjacent expert selections to be locally consistent. The control is deliberately selective: it reinforces the previous token's expert set only when two adjacent routes are already similar, preserving necessary routing changes at semantic and field boundaries. Because the introduced auxiliary router control can destabilize RL training, we further introduce an entropy-gated mechanism that disables the routing-control gradients when the policy entropy encounters high fluctuations. This framework preserves the token-wise top-k router, and requires no architectural modification to existing MoE models. It can as well be combined with different policy-gradient algorithms such as PPO (Schulman et al., 2017), GRPO (Shao et al., 2024), LOOP (Chen et al., 2025), and GiGPO (Feng et al., 2026). Through evaluations on AppWorld (Trivedi et al., 2024) and AutomationBench (Shepard & Salimans, 2026), we show that the proposed routing control framework improves task performance when combined with various underlying RL algorithms, stabilizes MoE RL training, and increases inference efficiency. Overall, these results show that agentic trajectory structure provides a useful signal for organizing MoE expert selection during post-training.

We summarize the main contributions as follows:

- We systematically study the connection between agentic behavior and MoE expert selection. We identify operation-grouped expert specialization and temporal routing consistency as key factors to achieve better task performance and inference efficiency. These routing structures are overlooked by standard RL training algorithms.
- We propose a hierarchical routing control framework that combines operation-aware turn-level specialization with outlier-tolerant token-level consistency. We also identify the stability risk of directly controlling MoE routers and introduce an entropy-gated control mechanism. The method supports the standard top-k MoE router and is compatible with different RL algorithms.
- We conduct comprehensive experiments on AppWorld and AutomationBench. Our routing control framework consistently improves agentic RL with improved routing structures, over 10-point task performance improvement, long-term training stability, and better inference efficiency.

2 Preliminaries

2.1 Agentic Reinforcement Learning

We define agentic trajectories to contain several rounds of reasoning and tool use, in the same spirit as ReAct (Yao et al., 2023). For example, trajectories in AppWorld (Trivedi et al., 2024) and AutomationBench (Shepard & Salimans, 2026) involve reading information, creating an object, and updating an object in later turns. Concretely, at turn i, the policy $\pi_{\boldsymbol{\theta}}$ generates a *thinking field* y_i^{think} to perform textual reasoning for the task, followed by a *tool-use field* y_i^{tool} containing executable code or a tool call. The environment executes the tool use and returns feedback that guides subsequent turns. For a task x, this interaction produces an N-turn trajectory

$$\tau = \left[y_1^{\text{think}}; y_1^{\text{tool}}; v_1; \ldots; y_N^{\text{think}}; y_N^{\text{tool}}; v_N\right], \tag{2.1}$$

where v_i is the intermediate environment feedback.

Reinforcement learning (RL) uses an outcome reward $R(\tau)$ as the training signal. Specifically, let τ_i denote the task x together with all generated fields and environment feedback before turn i. The simplest policy gradient for maximizing the expected outcome reward is

$$\nabla_{\boldsymbol{\theta}} J_{\text{RL}}(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}(\cdot|x)}\left[R(\tau)\cdot\sum_{i=1}^{N}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}\left(y_i^{\text{think}}, y_i^{\text{tool}} \mid \tau_i\right)\right]. \tag{2.2}$$

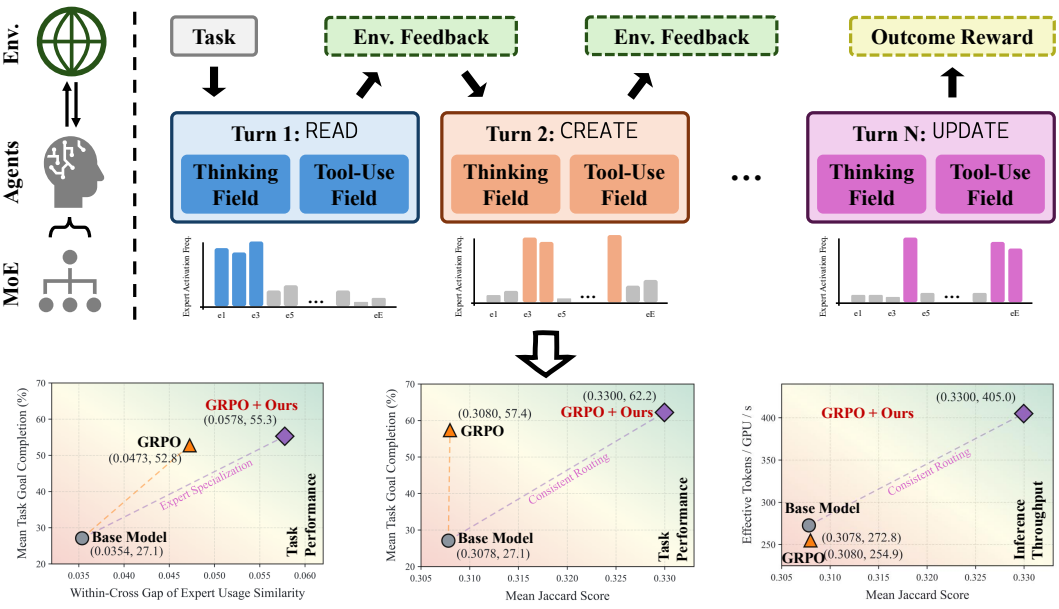


Figure 1: Overview of MoE routing control for agentic trajectories. **Top:** In each turn of interaction with the environment, agents generate *thinking* and *tool-use* fields, then receive *environment feedback*. Our routing control framework aims to align expert selection with agentic operations (e.g., `READ`, `CREATE`, and `UPDATE`) in each turn, assigning distinct expert groups to different operations and encouraging similar expert selections within each turn. **Bottom:** With `Qwen3-30B-A3B` on AppWorld, our routing control framework (when used with standard RL) unlocks the performance limits and raises the inference throughput, by improving expert specialization (larger with-cross gap of expert usage similarity in Appendix B.1) and consistency (higher Jaccard scores).

Environment feedback supplies context but contributes no policy log-probability term (Chen et al., 2025). Practical algorithms such as PPO (Schulman et al., 2017) or GRPO (Shao et al., 2024) replace the outcome reward with an advantage score, requiring further value estimation.

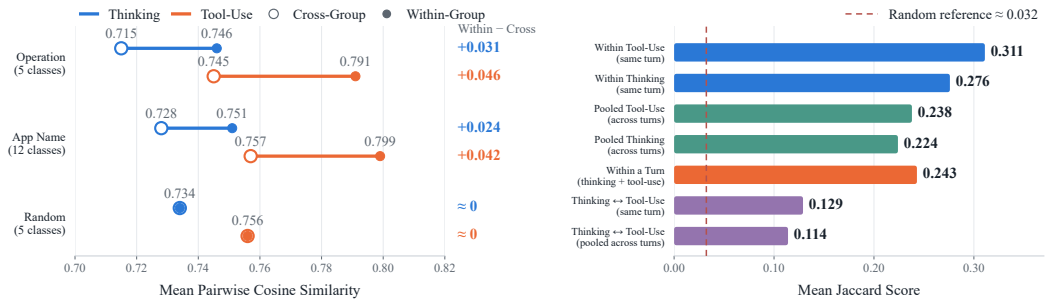
2.2 MoE Models: Efficiency and Expert Specialization

Generating the thinking and tool-use fields requires a sequence of token-level computations. An MoE policy performs these computations with sparse activation: each MoE layer contains M feed-forward experts, and a single-layer router selects only $k \ll M$ experts per token (Shazeer et al., 2017). Let $\mathcal{L}_{\text{MoE}}$ denote the set of MoE layers and t a token position in the interaction context. At layer $l \in \mathcal{L}_{\text{MoE}}$, the router maps the hidden representation $\mathbf{h}_{l,t}$ to expert logits and then computes a softmax distribution over all experts:

$$\mathbf{z}_{l,t} = W_l \mathbf{h}_{l,t}, \qquad \mathbf{p}_{l,t} = \text{softmax}(\mathbf{z}_{l,t}), \tag{2.3}$$

where W_l is the router's weight matrix and $p_{l,t}[e]$ is the probability assigned to expert e. The router selects the top-k experts, denoted by $\mathcal{S}_{l,t}$, and combines their output hidden states as a weighted sum using routing softmax probabilities. This sparse activation enables efficient parameter usage during inference (Yang et al., 2025; Qwen Team, 2026).

In agentic tasks, we study expert specialization (Dai et al., 2024; Bo et al., 2026; Wang et al., 2026c) through expert usage at both the token and turn levels. The expert-usage examples in Fig. 1 illustrate our framework's intended outcome: different operations preferentially use distinct expert groups. Standard RL optimizes task completion without explicitly optimizing the routing strategy.



(a) Turn-Level Specialization

(b) Token-Level Routing Similarity

Figure 2: Default routing statistics of `Qwen3-30B-A3B` on AppWorld. **Left:** Pairwise cosine similarity of token-level expert distributions; the difference between within group and cross-group cosine similarity values indicates a relatively higher expert overlap between tokens that share the same operation label. **Right:** Jaccard score between token-level expert sets is relatively higher within the same field and turn indicating more overlapping expert routing patterns.

3 METHOD

In this section, we introduce the proposed hierarchical routing control framework, namely turn-level control (Section 3.1) and token-level control (Section 3.2). We also discuss the MoE training stability issues induced by our framework and provide an effective solution in Section 3.3.

Before introducing the routing control method, we analyze the default routing statistics of off-the-shelf MoE models in Fig. 2 and Fig. 8. At the turn level, we try different grouping criteria beyond operations and compare within-group and cross-group expert-usage similarity. This similarity is computed across the expert distributions of token pairs. The gap between within- and cross-group similarity has been used to monitor the expert specialization of MoE models (Wang et al., 2026c). The results demonstrate that off-the-shelf MoE models naturally tend to use distinct expert groups for different operations. At the token level, we also observe that expert overlap for tokens within the same field and turn is relatively higher when compared to any cross-field or cross-turn scenario. Metrics used in the analysis are detailed in Appendix B.

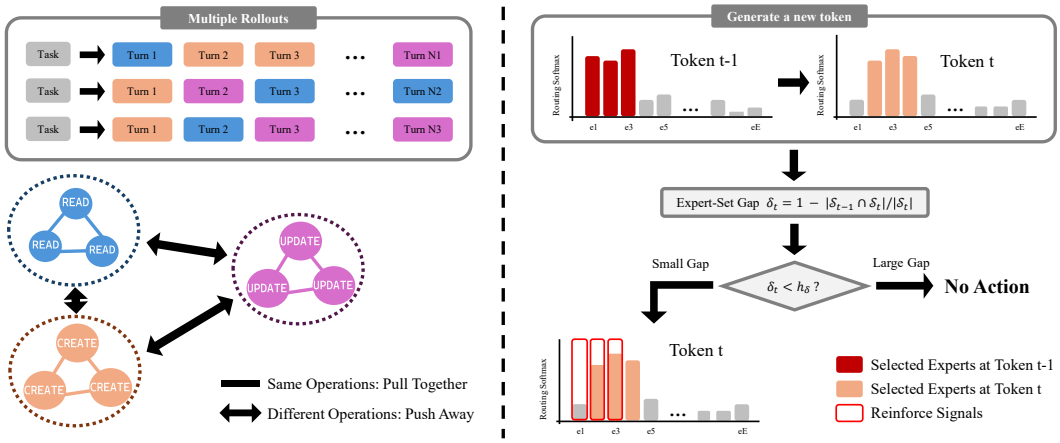
However, the default routing statistics are often perturbed during RL training. As shown in Section 4.3, RL injects noise into both token-level and turn-level routing statistics, which hinders task performance and inference efficiency. These issues motivate our explicit routing control framework to denoise expert selection.

3.1 TURN-LEVEL OPERATION-AWARE CONTROL

Agentic trajectories consist of multiple interactive turns involving different semantic operations. For example, agents may capture information, create objects, or update existing states in different turns. The tools/APIs actually used may differ but can be broadly categorized into a few semantic groups. We show the concrete rules for grouping operations in Appendix D.6.

Inspired by Bo et al. (2026), which uses mutual information (MI) between expert bins and modality to encourage expert specialization, we introduce MI-based regularization for turn-level routing control. For each MoE layer, we compute the following quantities separately for thinking and tool-use fields in a turn, omitting the layer and field-type indices for simplicity. Recalling Fig. 2(b), the aforementioned routing structure only applies to a group of thinking/tool-use fields, and is not observable for cross-thinking-tool-use scenario.

Let $\mathcal{T}_i$ denote the token indices of the i-th turn in the trajectory. The turn-level expert selection distribution for i-th turn is defined as $\mathbf{q}_i(e) = \frac{1}{|\mathcal{T}_i|}\sum_{j\in\mathcal{T}_i}\mathbf{p}_j(e)$, where $\mathbf{p}_j(e)$ denotes the router expert selection probability for expert e at token j. A turn may contain multiple operations. For each operation o, define $y_{io} = \mathbb{I}[o \in \mathcal{O}_i]$, where $\mathcal{O}_i \subset \mathcal{O}$ is the turn's operation-label set. We



(a) Turn-Level Routing Control **(b)** Token-Level Routing Control

Figure 3: Overview of our hierarchical routing control framework. **Left:** Turn-level control aggregates router distributions within each field and maximizes the mutual information between the routing distribution and the turn's operation labels. **Right:** Token-level control measures the top-k expert-set gap between adjacent tokens and reinforces the preceding expert set only when this gap is small. The entropy gate is omitted here for clarity.

measure the association between expert identity E and operation presence Y_o through

$$\mathcal{I}(E; Y_o) = \mathbf{KL}\left(P_o(E, Y_o) \| P(E) \otimes P_o(Y_o)\right). \tag{3.1}$$

Across N eligible turns, we estimate the marginal and joint distributions of experts and operations:

$$\hat{P}(e) = \frac{1}{N}\sum_{i=1}^{N} q_i(e), \qquad \hat{P}_o(y) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[y_{io} = y],$$
$$\text{and } \hat{P}_a(e, y) = \frac{1}{N}\sum_{i=1}^{N} q_i(e)\mathbb{I}[y_{io} = y], \quad e \in \{e_1, e_2, ..., e_M\}, \quad y \in \{0, 1\}. \tag{3.2}$$

Then, the stabilized turn-level MI objective is

$$J_{\text{turn}} = \frac{1}{|\mathcal{O}|}\sum_{o\in\mathcal{O}}\sum_{e}\sum_{y\in\{0,1\}} \hat{P}_o(e, y) \cdot \text{clip}\left(\log \frac{\hat{P}_o(e, y) + \epsilon}{\hat{P}(e)\hat{P}_o(y) + \epsilon}, -h_c, h_c\right), \tag{3.3}$$

where $\mathcal{O}$ contains all possible operations. We use $\epsilon = 10^{-12}$ for numerical stability and $h_c = 20$ to bound the pointwise MI values used as control signals. Practically, we compute J_{turn} over all trajectories in a batch, and require at least 8 turns to construct the expert-operation association. No MI signal is applied when there are not enough turns.

Maximizing J_{turn} encourages turns with the same operation to favor similar experts and turns with different operations to develop distinct expert preferences. For each operation o, the log-ratio in Eq. 3.3 identifies experts that are over- or under-represented among turns with $Y_o = y_{io}$ relative to their overall usage. Turns containing the same operation receive a shared signal from its MI term, reinforcing experts associated with that operation and suppressing less associated experts. Turns without that operation receive complementary preferences, separating the two operation-conditioned mean distributions. Thus, the objective strengthens existing expert-operation associations and encourages consistency through shared expert preferences.

The relation to load balancing (Shazeer et al., 2017) follows from the underlying MI decomposition $I(E; Y_o) = H(E) - H(E \mid Y_o)$. The marginal-entropy term favors diverse overall expert usage, while the conditional-entropy term favors more concentrated usage within each operation state. Standard load-balancing losses encourage approximately uniform expert utilization without using

operation labels, while the MI objective additionally rewards operation-dependent usage. Therefore, J_{turn} complements load balancing by promoting operation specialization, but does not guarantee uniform expert utilization. We also empirically compare load-balancing loss with the proposed turn-level routing control in Appendix C.5.

3.2 Token-Level Local Consistency Control

RL tends to inject noise into token-level expert selection, increasing expert switching between adjacent tokens (empirical evidence is shown in Section 4.3). This can disrupt local routing consistency within a thinking or tool-use field and introduce overhead during trajectory generation. Yang et al. (2026) similarly motivates temporally consistent expert assignments for coherent agent behavior, using a switching penalty between consecutive environment steps. We comprehensively compare multiple variants of token-level routing control (comparison details in Appendix C) and determine that, for widely used MoE architectures such as Qwen's MoE models, a simple adjacent-token loss with a gap threshold is the most reliable choice.

As illustrated in Fig. 3(b), we reinforce the previous token's expert set only when the current selection is already similar to the previous token's. For adjacent token positions $t-1$ and t within the same field, we measure the expert-set gap for each MoE layer as $\delta_t = 1 - |\mathcal{S}_{t-1} \cap \mathcal{S}_t|/|\mathcal{S}_t|$. Given a threshold $h_\delta \in [0, 1]$, we encourage the current router to retain the previous token's expert selection by minimizing

$$\ell_t^{\text{token}} = -\frac{\mathbb{I}[\delta_t < h_\delta]}{|\mathcal{S}_{t-1}|} \sum_{e \in S_{t-1}} \log p_t[e]. \tag{3.4}$$

The previous token's router probabilities and the gate are treated as detached variables. Only the current token's probabilities are differentiable. The token-level objective $J_{\text{token}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t\in\mathcal{T}_i} \ell_t^{\text{token}}$ averages this per-token loss over all within-field adjacent token pairs. Pairs crossing field or turn boundaries are excluded.

The expert-set gap threshold h_δ makes this control selective. A small gap is treated as a local perturbation of an existing route, whereas a large gap may reflect a meaningful semantic transition and receives no consistency penalty. Applying the penalty to every pair could suppress such transitions and impose overly rigid routing.

This control reduces local routing noise while allowing expert selections to change when needed, complementing the operation-level specialization in Section 3.1. Empirically, it also improves rollout throughput and reduces rollout time (see Section 4.4 for results and Appendix E.3 for explanation).

3.3 Entropy-Gated Control for Stable MoE RL

The stability issues of MoE RL have been studied for single-turn tasks (Ma et al., 2026; Zhang et al., 2026). On multi-turn agentic tasks, our routing controls may also destabilize RL training. Small router score changes near the top-k boundary can alter expert selection, and strong auxiliary updates may interfere with policy optimization. Empirically, training collapse is accompanied by a sharp fluctuation in policy entropy (Wang et al., 2026b; Li et al., 2026a; Lochab et al., 2026), which motivates using it as a stability safeguard.

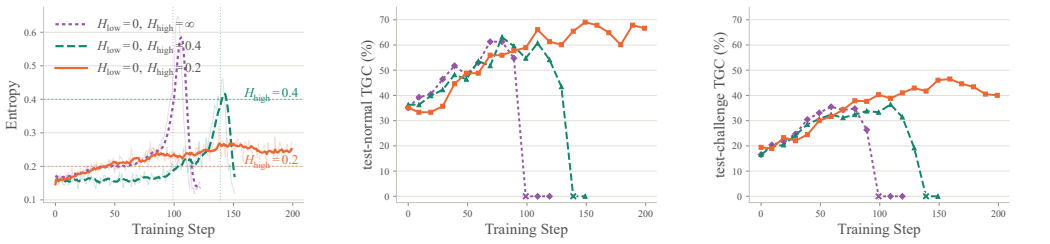
For a generated token position t, the policy entropy is defined as

$$\mathcal{H}_t = -\sum_{v \in \mathbb{V}} \pi_{\boldsymbol{\theta}}(v|\tau_{<t}) \cdot \log \pi_{\boldsymbol{\theta}}(v|\tau_{<t}), \tag{3.5}$$

where $\mathbb{V}$ is the vocabulary and $\tau_{<t}$ contains the generated tokens and environment feedback before position t. Let $\mathcal{H}$ denote its average over all thinking and tool-use tokens in the rollout batch at one RL step. We apply the routing control only when $H_{\text{low}} \le \mathcal{H} \le H_{\text{high}}$. Therefore, the entropy-gated objective is

$$J = J_{\text{RL}} + g \cdot (\lambda_{\text{turn}} J_{\text{turn}} + \lambda_{\text{token}} J_{\text{token}}), \tag{3.6}$$

where $g = \mathbb{I}[H_{\text{low}} \le \mathcal{H} \le H_{\text{high}}]$. When the gate closes, the routing-control gradients are detached while the standard RL updates continue. Routing control resumes when the monitored policy entropy $\mathcal{H}$ returns within the target range.



(a) Policy Entropy Curves **(b)** AppWorld Normal-Set Performance **(c)** AppWorld Challenge-Set Performance

Figure 4: Effect of entropy gating on RL with routing control. Crosses mark the first evaluation at which both splits reach zero TGC. Routing control alone induces training collapse, and RL training remains stable for all 200 steps when an appropriate entropy gate (target range $[0, 0.2]$) is applied.

To empirically show these stability issues and the effect of entropy gating, we compare training curves using the same routing control and different entropy gates in Fig. 4[1]. Without entropy gating, policy entropy spikes and both evaluation splits collapse to zero around step 100. A threshold $H_{\text{high}} = 0.4$ delays collapse to around step 140. With $H_{\text{high}} = 0.2$, training remains stable, retaining task performance on both splits. This supports the effectiveness of entropy gating.

4 Experiments

4.1 Setup

Benchmarks and metrics. We train on 90 AppWorld (Trivedi et al., 2024) tasks and 480 AutomationBench (Shepard & Salimans, 2026) public training tasks spanning six domains. For AutomationBench, we split the 600-sample public set into a training and a test subsets, each containing all six domains. AppWorld agents interact with applications through executable Python code. We report task goal completion (TGC), the percentage of fully completed tasks, and scenario goal completion (SGC), the percentage of scenarios with all constituent tasks completed, on `test-normal` and `test-challenge`. AutomationBench agents execute structured tool calls and receive partial-credit rewards based on the fraction of assertions passed; we report scores on withheld public subset, which are not used for training.

Models and training. We use the instruction-following versions of `Qwen3-30B-A3B-2507` for AppWorld and the more powerful `Qwen3.5-35B-A3B` for the harder AutomationBench, to guarantee initial success rates. The training and rollout generation of both models are colocated on 8 NVIDIA B200 GPUs. Within each baseline–control comparison, we keep the model, rewards, and RL hyperparameters fixed. We evaluate checkpoints periodically. The reported learning curves span 200 AppWorld steps and 100 AutomationBench steps.

Compared methods. We augment PPO (Schulman et al., 2017), GRPO (Shao et al., 2024), LOOP (Chen et al., 2025), and GiGPO (Feng et al., 2026) with our routing controls. Unless otherwise stated, "+ Ours" combines turn-level control, token-level control, and entropy gating (Section 3). We use GRPO on AppWorld for the routing and efficiency analysis.

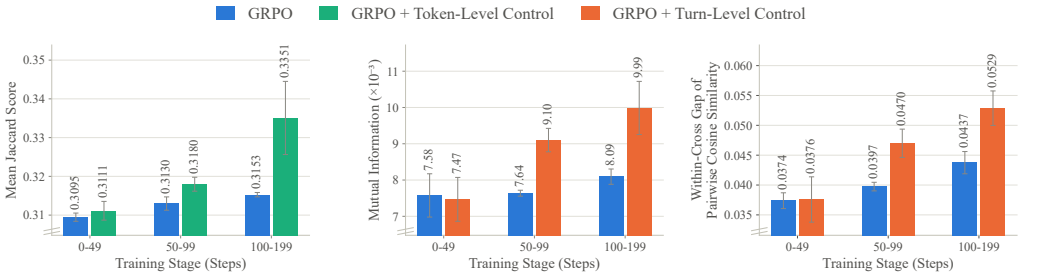
4.2 End-to-End Task Performance

Table 1 shows that our routing control framework improves all RL algorithms across the 2 benchmarks. Specifically, we observe up to 12.9-point improvements in AppWorld (Trivedi et al., 2024). For AutomationBench (Shepard & Salimans, 2026), our routing control improves LOOP (Chen et al., 2025) by 11.34 points. The reported numbers are from the best-performance checkpoints on AppWorld `dev` set and AutomationBench public set respectively. We notice the PPO (Schulman et al., 2017) exhibits inconsistent improvement on AppWorld, and assume this to be the outcome of

[1]We use $\lambda_{\text{turn}} = 2 \times 10^{-6}$ and $\lambda_{\text{token}} = 2 \times 10^{-7}$ here, as well as in the main experiments.

Table 1: Agentic task performance (in percentages). **Bold** and underline mark the highest and second-highest scores per column. Arrows show percentage-point changes from the corresponding baseline. Our routing control framework improve the agentic RL on both benchmarks, supporting diverse RL algorithms.

Method	AppWorld Qwen3-30B-A3B-2507				AutomationBench Qwen3.5-35B-A3B
	test-normal		test-challenge		
	TGC (%)	SGC (%)	TGC (%)	SGC (%)	
Base Model	35.1	14.3	18.7	7.2	22.04
PPO	63.1	37.5	35.5	16.5	54.56
+ Ours	60.7 (2.4 ↓)	42.9 (5.4 ↑)	38.6 (3.1 ↑)	20.9 (4.3 ↑)	57.39 (2.83 ↑)
GRPO	69.6	46.4	40.5	22.3	63.01
+ Ours	**75.0** (5.4 ↑)	55.4 (8.9 ↑)	49.2 (8.6 ↑)	**34.5** (12.2 ↑)	69.54 (6.53 ↑)
LOOP	64.9	46.4	37.4	21.6	55.50
+ Ours	73.2 (8.3 ↑)	**58.9** (12.5 ↑)	**50.4** (12.9 ↑)	30.9 (9.4 ↑)	66.84 (11.34 ↑)
GiGPO	52.4	26.8	32.9	17.3	70.01
+ Ours	57.7 (5.4 ↑)	39.3 (12.5 ↑)	37.4 (4.6 ↑)	18.7 (1.4 ↑)	**70.24** (0.23 ↑)



(a) Token-Level Control Effect **(b)** Turn-Level Control Effect **(c)** Turn-Level Expert Specialization

Figure 5: Routing statistics during AppWorld GRPO training. (a) Token-level control improve routing similarity. The numbers are averaged over all token pairs within fields. (b) Turn-level control increases the mutual information between expert selection and operation labels. (c) Turn-level control makes expert selections more separable by the operation labels. The reported numbers are averaged over all of the evaluation trajectories collected from the specified training step intervals. The error bars are the standard deviation of all data points.

the small set of trajectories for a single prompt. Under PPO, each prompt has far fewer turns than under group-based RL, which makes the calculation of mutual information unstable.

4.3 Changes in Routing Behavior

Fig. 5 examines the two controls separately. Under GRPO, the mean Jaccard similarity within each field stays relatively stable, while the token-level control increases the Jaccard score for 7.7% (Figure 5(a)), indicating improved local consistency. This improved routing consistency demonstrates the effectiveness of the token-level control.

Turn-level control strengthens the expert-operation alignment in later stages. We use the mutual information (MI) between turn-level expert distributions and operation labels (the same as the turn-level control objective) to quantify this dependence (Bo et al., 2026). Late-stage MI increases 23.5% (Figure 5(b)). Inspired by Wang et al. (2026c), we also quantify the expert specialization via the gap between within- and cross-group expert-usage similarity, as defined in Appendix B.1. This diagnostic measures whether tokens tend to use overlapped experts more when they are under the same operation label than different labels (Fig. 5(c)). The results support greater operation separability.

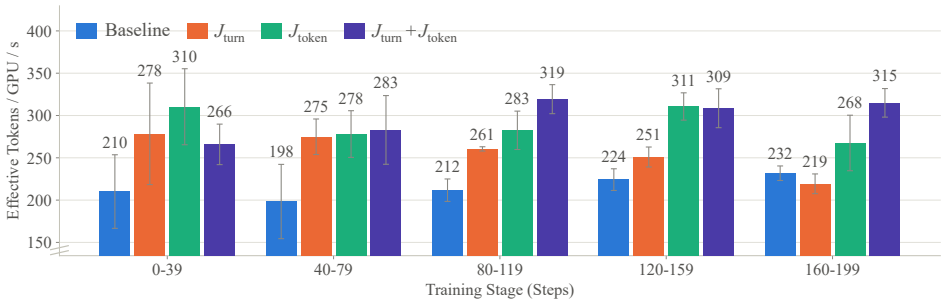


Figure 6: AppWorld rollout efficiency comparison under GRPO with turn-level control, token-level control, or both. Both controls improve inference throughput. The token-level control contributes the most to inference efficiency at the early training stage. The reported numbers are averaged over all of the evaluation trajectories collected from the specified training step intervals. The error bars are the standard deviations of all data points.

4.4 Inference Efficiency

Fig. 6 reports rollout wall time and effective generated tokens per GPU second. Each control improves both metrics across all three training stages. In steps 100–200, the combined method increases throughput from 224 to 316 tokens/GPU/s (+41.1%) and reduces wall time from 224 to 174 seconds (a 22.3% reduction). The combination achieves the highest late-stage throughput, whereas token-level control alone gives the shortest wall time (133 seconds). These end-to-end measurements demonstrate that improving the routing consistency has benefit on inference efficiency. We also have a detailed discussion about this efficiency benefit in Appendix E.3.

5 Related Works

RL for Interactive LLM Agents. Prior works improve long-horizon agents through better training objectives (Chen et al., 2025), credit assignment (Feng et al., 2026; Li et al., 2026b; Zhang et al., 2025), promoting RL exploration (Szot et al., 2026; Petrenko et al., 2026), and efficient environment construction and interaction (Li et al., 2025; Wang et al., 2026a). Our routing control framework is orthogonal to these advances and improves any agentic RL algorithm via MoE optimization.

MoE Routing Control. Optimizing the MoE expert selection can improve MoE RL (Ma et al., 2026; Ko et al., 2026). For the MoE optimization, prior works promote temporal consistency and expert reuse (Shen & Henderson, 2026; Zhu et al., 2026), and encourage specialization around task phases or modalities (Yang et al., 2026; Bo et al., 2026). Our framework combines these aims into a hierarchical routing control consisting turn-level specialization and token-level consistency, which improves task performance and inference efficiency simutenously.

6 Conclusion

We study how agentic post-training interacts with expert selection in mixture-of-experts (MoE) agents. We observe that agentic trajectories naturally contain structured expert usage patterns related to the operation labels of each turn, and that ignoring such structure induces suboptimal task performance and efficiency. To address this problem, we introduce a hierarchical routing control framework that combines operation-aware turn-level control with selective token-level routing control. We further use policy entropy to gate the auxiliary routing gradients, preventing the instability caused by routing control. Experiments on AppWorld and AutomationBench show that the proposed framework improves the task performance of multiple RL algorithms, induces the intended routing structure, stabilizes long-term training, and also improves inference efficiency. Overall, our results demonstrate that the structure of agentic trajectories provides a useful signal for organizing MoE usage during post-training.

AI USE STATEMENT

We used generative AI tools to improve the clarity and readability of the manuscript, assist in preparing tables and figures, and support code development. The authors take full responsibility for the final content of this work, including its text, tables, figures, and scientific claims.

REFERENCES

Zi-Hao Bo, Yaqian Li, Anzhou Hou, Rinyoichi Takezoe, Ertao Zhao, Tianxiang Pan, Jiale Yan, Mo Guang, and Kaiwen Long. Smoes: Soft modality-guided expert specialization in moe-vlms. *arXiv preprint arXiv:2604.23996*, 2026.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive llm agents. *arXiv preprint arXiv:2502.01600*, 2025.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1280–1297, 2024.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408, 2026.

DP Kingma and JL Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.

Dohwan Ko, Jinyoung Park, Seoung Choi, Sanghyeok Lee, Seohyun Lee, and Hyunwoo J Kim. Moe-grpo: Optimizing mixture-of-experts via reinforcement learning in vision-language models. *arXiv preprint arXiv:2603.24984*, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021.

Bolian Li, Yifan Wang, Yi Ding, Anamika Lochab, Ananth Grama, and Ruqi Zhang. Addressing performance saturation for llm rl via precise entropy curve control. *arXiv preprint arXiv:2604.26326*, 2026a.

Jiazheng Li, Yawei Wang, Qiaojing Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. Salt: Step-level advantage assignment for long-horizon agents via trajectory graph. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 4709–4725, 2026b.

Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. Simulating environments with reasoning models for agent training. *arXiv preprint arXiv:2511.01824*, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Anamika Lochab, Bolian Li, and Ruqi Zhang. Uniform-correct policy optimization: Breaking rlvr's indifference to diversity. *arXiv preprint arXiv:2605.00365*, 2026.

Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Fuli Luo, and Zhifang Sui. Stabilizing moe reinforcement learning by aligning training and inference routers. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ZMkfGplf4b.

Aleksei Petrenko, Ben Lipkin, Kevin Chen, Erik Wijmans, Marco Francis Cusumano-Towner, Raja Giryes, and Philipp Kraehenbuehl. Entropy-preserving reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Zeyu Shen and Peter Henderson. Temporally extended mixture-of-experts models. *arXiv preprint arXiv:2604.20156*, 2026.

Daniel Shepard and Robin Salimans. Automationbench. *arXiv preprint arXiv:2604.18934*, 2026.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Andrew Szot, Michael Kirchhof, Omar Attia, and Alexander Toshev. Expanding llm agent boundaries with strategy-guided exploration. *arXiv preprint arXiv:2603.02045*, 2026.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Ziwei Chai, Y Charles, HS Che, Cheng Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1529–1550, 2026a.

Shumin Wang, Yuexiang Xie, Wenhao Zhang, Yuchang Sun, Yanxi Chen, Yaliang Li, and Yanyong Zhang. On the entropy dynamics in reinforcement fine-tuning of large language models. In *Forty-third International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=Sj8NBI6aV0.

Xi Wang, Soufiane Hayou, and Eric Nalisnick. The myth of expert specialization in moes: Why routing reflects geometry, not necessarily domain expertise. *arXiv preprint arXiv:2604.09780*, 2026c.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.emnlp-demos.6/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shengtian Yang, Yu Li, Shuo He, Yewen Li, Qingpeng Cai, Peng Jiang, and Lei Feng. Phase-aware mixture of experts for agentic reinforcement learning. In *Forty-third International Conference on Machine Learning*, 2026.

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

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Di Zhang, Xun Wu, Shaohan Huang, Lingjie Jiang, Yaru Hao, Li Dong, Zewen Chi, Zhifang Sui, and Furu Wei. Towards stable and effective reinforcement learning for mixture-of-experts. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25446–25457, 2026.

Zijing Zhang, Ziyang Chen, Mingxiao Li, Zhaopeng Tu, and Xiaolong Li. Rlvmr: Reinforcement learning with verifiable meta-reasoning rewards for robust long-horizon agents. *arXiv preprint arXiv:2507.22844*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Xiongwei Zhu, Xiaojian Liao, Tianyang Jiang, Yusen Zhang, Liang Wang, and Limin Xiao. Remoe: Boosting expert reuse through router fine-tuning in memory-constrained moe LLM inference. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ylAhgNb2ak.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

CONTENTS

1 Introduction **1**

2 Preliminaries **2**

2.1 Agentic Reinforcement Learning 2

2.2 MoE Models: Efficiency and Expert Specialization 3

3 Method **4**

3.1 Turn-Level Operation-Aware Control 4

3.2 Token-Level Local Consistency Control 6

3.3 Entropy-Gated Control for Stable MoE RL 6

4 Experiments **7**

4.1 Setup 7

4.2 End-to-End Task Performance 7

4.3 Changes in Routing Behavior 8

4.4 Inference Efficiency 9

5 Related Works **9**

6 Conclusion **9**

A Detailed Related Works **15**

A.1 RL for Interactive LLM Agents 15

A.2 MoE Routing Control 15

B Metric Definitions **15**

B.1 Measuring Expert Specialization 16

B.1.1 Expert-Operation Classification 16

B.1.2 Similarity Gap 16

B.2 Measuring Temporal Routing Consistency 17

B.2.1 Jaccard Similarity 17

B.2.2 Expert Switches 18

B.3 Measuring Expert Load Balance 18

C Variant Routing Controls **18**

C.1 Routing Regularization: Field Consensus 18

C.2 Switch Regularization: Adjacent-Distribution Agreement 19

C.3 Outlier Regularization 19

C.4 Routing Reward and Routing Clipping 20

C.5 Load-Balancing Loss 20

D Technical Details **21**

D.1 MoE Model Architecture 21

D.2 Agentic RL Infrastructure . . . 21
D.3 Inference Settings . . . 21
D.4 Training Settings . . . 21
D.5 Injected Gradient Flow in RL Regularization . . . 22
D.6 Rubrics for Grouping Agentic Operations . . . 23
D.6.1 AppWorld Rubrics . . . 23
D.6.2 AutomationBench Rubrics . . . 24

E Additional Discussion **24**
E.1 Is It Possible to Manually Fix the Within-Field Routing? . . . 24
E.2 Is It Possible to Directly Control the Routers' Entropy? . . . 25
E.3 Why Does Routing Consistency Improve Efficiency? . . . 25

F Additional Experiments **26**
F.1 Ablation Study on Turn Labels . . . 26
F.2 Histogram of the Number of Shifted Experts . . . 27
F.3 Effect of R3 on Agentic RL . . . 28
F.4 Ablation Study on Entropy Loss . . . 28
F.5 Ablation Study on the Token-Level Control Threshold . . . 28
F.6 Hyperparameter Magnitude Search . . . 28
F.7 Ablation Study on Routing Control Components . . . 29
F.8 Ablation Study on Rollout Efficiency . . . 30

G Agentic Trajectory Examples **30**
G.1 AutomationBench Trajectory . . . 30

A Detailed Related Works

In this section, we discuss related work alongside the proposed method, including research on agentic RL (Appendix A.1) and MoE routing control (Appendix A.2).

A.1 RL for Interactive LLM Agents

Reinforcement learning enables LLM agents to improve through repeated interaction with stateful environments. LOOP (Chen et al., 2025) demonstrates this approach on long-horizon interactive tasks using a critic-free policy optimization objective. However, terminal outcome rewards provide limited guidance about the contribution of individual turns. GiGPO (Feng et al., 2026) derives step-level advantages by grouping actions at shared environment states, while SALT (Li et al., 2026b) constructs trajectory graphs to refine advantage assignment from outcome rewards. RLVMR (Zhang et al., 2025) provides additional process supervision through verifiable rewards for intermediate meta-reasoning behaviors.

Sparse rewards also make successful trajectories difficult to discover. SGE (Szot et al., 2026) guides exploration with high-level language strategies, while ADAPO (Petrenko et al., 2026) adaptively adjusts policy clipping to preserve entropy and output diversity. Other studies also address the cost of environment interaction. Simia-RL (Li et al., 2025) uses reasoning models to simulate environment feedback and rewards, reducing the engineering required to construct executable training environments. SAGE (Wang et al., 2026a) learns to generate and reuse a skill library, reducing interaction steps and generated tokens when solving related tasks. These approaches improve the learning signal, exploration, or interaction procedure. Our work studies how agentic RL changes expert selection within MoE models and uses the structure of turns to regularize their routing. This control complements the underlying policy-gradient estimator.

A.2 MoE Routing Control

Recent work studies routing as a source of both instability and adaptability during MoE post-training. Rollout Routing Replay (R3) (Ma et al., 2026) stabilizes RL by replaying rollout-time expert assignments during training, aligning the discrete computation paths used for sampling and optimization. MoE-GRPO (Ko et al., 2026) instead treats expert selection as a stochastic policy and uses reward-based updates to explore expert combinations in vision-language models. These methods address consistency between training and inference or reward-driven routing optimization, without explicitly organizing expert usage around the operations performed by an interactive agent.

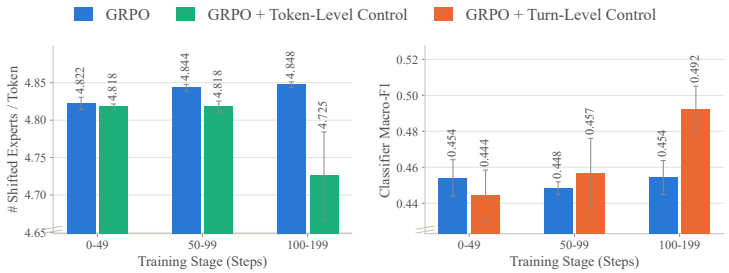
Temporal consistency provides another signal for routing control. Shen & Henderson (2026) introduce temporally extended MoE models that retain expert selections across multiple tokens and quantify routing persistence through expert switch rates. ReMoE (Zhu et al., 2026) fine-tunes native routers with temporal-locality losses to improve expert reuse under memory-constrained inference. Our token-level control shares the goal of local consistency, but selectively reinforces the preceding expert set only when adjacent selections are already similar within a thinking or tool-use field. This leaves larger routing changes unpenalized by the token-level loss and preserves the native top-k selection rule.

PA-MoE (Yang et al., 2026) connects routing specialization with agentic RL by learning phase-dependent expert selection and regularizing switching across environment steps. Its reported implementation uses added LoRA experts across transformer layers. SMoES (Bo et al., 2026) also promotes structured specialization, maximizing mutual information between soft modality scores and expert bins in vision-language models. Our turn-level objective instead uses operation labels extracted from generated tool-use fields to regularize the native expert distributions. It strengthens existing expert-operation associations without assigning operations to expert groups in advance. Our routing control framework combines this operation-level signal with selective token-level consistency to optimize the routing strategy during agentic RL.

B Metric Definitions

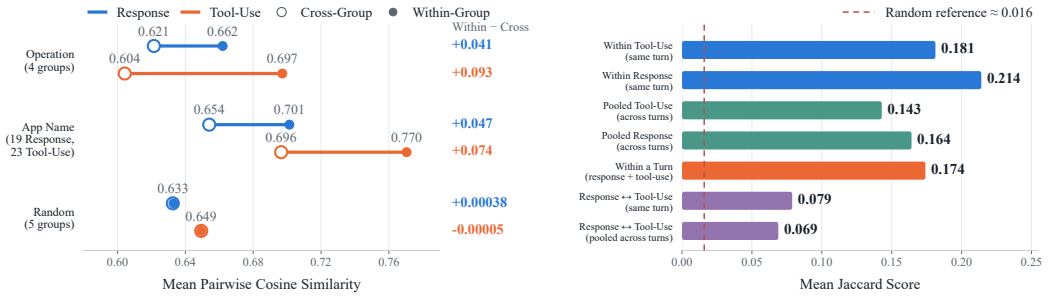
In this section, we define several evaluation metrics used in the empirical analysis. These metrics provide a foundation for understanding MoE model architectures and routing strategies. We show the specialization and consistency measurement along the RL training in Fig. 7.

Additionally, we show the default routing statistics for `Qwen3.5-35B-A3B` in Fig. 8 using our defined metrics in this section. The routing statistics for `Qwen3.5-35B-A3B` is similar to that of `Qwen3-30B-A3B`. Off-the-shelf MoE models naturally exhibit expert specialization and routing consistency.



(a) Token-Level Control Effect **(b)** Turn-Level Expert Specialization

Figure 7: Routing behavior changes during training. Measured by the expert switches and expert-operation classification, our routing control framework improves the routing consistency and expert specialization.



(a) Turn-Level Specialization **(b)** Token-Level Routing Similarity

Figure 8: Default routing statistics of `Qwen3.5-35B-A3B` on Automation. **Left:** Pairwise cosine similarity of token-level expert distributions; the difference between within group and cross-group cosine similarity values indicates a relatively higher expert overlap between tokens that share the same operation label. **Right:** Jaccard score between token-level expert sets is relatively higher within the same field and turn indicating more overlapping expert routing patterns.

B.1 Measuring Expert Specialization

B.1.1 Expert-Operation Classification

To directly show the association between expert selection and turn operations, we train a nearest-centroid classifier to predict turn labels based on input expert indices. We split the expert-operation pairs in the evaluation set into K subsets and use K-fold cross-validation to ensure fair scores. Then, we pool the held-out predictions from all K folds and compute

$$\mathbf{F1} = \frac{1}{|\mathcal{O}|} \sum_{o \in \mathcal{O}} \frac{2\mathrm{TP}_o}{2\mathrm{TP}_o + \mathrm{FP}_o + \mathrm{FN}_o}, \tag{B.1}$$

where $\mathcal{O}$ is the set of observed operation classes in the evaluation group, and the counts are computed from the pooled predictions. This metric measures operation discriminability within the evaluation collection.

B.1.2 Similarity Gap

To measure turn-level expert specialization in Fig. 2(a), we examine whether turns with the same operation label have more similar routing distributions than turns with different labels. Following Wang et al. (2026c), we compare expert-usage frequency vectors using cosine similarity. The gap between within- and across-group similarities measures routing specialization with respect to these labels.

We first summarize token-level expert selections into one routing distribution per field in each turn, then group fields by their turn labels. Within-group similarity averages over distinct fields in the same group, whereas across-group similarity averages over fields from two different groups. Both quantities can be computed exactly from sums of normalized routing distributions.

Turn-level routing distributions. For a fixed MoE layer and field type (thinking or tool use), let f index a nonempty field instance containing T_f tokens. Let $\boldsymbol{m}_t^f \in \{0,1\}^E$ indicate the k experts selected at token position t, where E is the number of experts and $\sum_{e=1}^{E} \boldsymbol{m}_t^f[e] = k$. The routing distribution of field f is

$$\boldsymbol{p}^f = \frac{1}{T_f k} \sum_{t=1}^{T_f} \boldsymbol{m}_t^f, \qquad \sum_{e=1}^{E} \boldsymbol{p}^f[e] = 1. \tag{B.2}$$

We compute the following quantities separately for each layer and field type, omitting these indices for clarity.

Within- and across-group cosine similarity. Let $\{\mathcal{G}_1, \ldots, \mathcal{G}_M\}$ be the nonempty turn-label groups, where $\mathcal{G}_m$ contains fields whose turns carry label m, and write $n_m = |\mathcal{G}_m|$. Define the normalized field vector and its group sum as

$$\boldsymbol{u}^f = \frac{\boldsymbol{p}^f}{\|\boldsymbol{p}^f\|_2}, \qquad \boldsymbol{s}(\mathcal{G}_m) = \sum_{f \in \mathcal{G}_m} \boldsymbol{u}^f. \tag{B.3}$$

For $n_m \geq 2$, the within-group similarity excludes self-pairs:

$$\mathbf{WS}(\mathcal{G}_m) = \frac{1}{n_m(n_m-1)} \sum_{\substack{f,g \in \mathcal{G}_m \\ f \neq g}} (\boldsymbol{u}^f)^\top \boldsymbol{u}^g = \frac{\|\boldsymbol{s}(\mathcal{G}_m)\|_2^2 - n_m}{n_m(n_m-1)}. \tag{B.4}$$

For $m \neq n$, the across-group similarity is

$$\mathbf{CS}(\mathcal{G}_m, \mathcal{G}_n) = \frac{1}{n_m n_n} \sum_{f \in \mathcal{G}_m} \sum_{g \in \mathcal{G}_n} (\boldsymbol{u}^f)^\top \boldsymbol{u}^g = \frac{\boldsymbol{s}(\mathcal{G}_m)^\top \boldsymbol{s}(\mathcal{G}_n)}{n_m n_n}. \tag{B.5}$$

Lemma 1 (Exact computation from group sums)**.** *Given the E-dimensional field routing distributions, all within-group similarities for groups of size at least two and all across-group similarities can be computed exactly in*

$$O\left(E \sum_{m=1}^{M} n_m + EM^2\right) \tag{B.6}$$

time by computing and caching each group sum once. For equally sized groups with $n_m = n$, this reduces the time complexity from $O(EM^2n^2)$ for explicit pairwise computation to $O(E(Mn + M^2))$.

Proof. Since $\|\boldsymbol{u}^f\|_2 = 1$, expanding the squared group-sum norm gives

$$\|\boldsymbol{s}(\mathcal{G}_m)\|_2^2 = \sum_{f,g \in \mathcal{G}_m} (\boldsymbol{u}^f)^\top \boldsymbol{u}^g = n_m + \sum_{\substack{f,g \in \mathcal{G}_m \\ f \neq g}} (\boldsymbol{u}^f)^\top \boldsymbol{u}^g.$$

Subtracting the n_m self-pairs and dividing by $n_m(n_m-1)$ proves Eq. B.4. Similarly, bilinearity gives

$$\boldsymbol{s}(\mathcal{G}_m)^\top \boldsymbol{s}(\mathcal{G}_n) = \sum_{f \in \mathcal{G}_m} \sum_{g \in \mathcal{G}_n} (\boldsymbol{u}^f)^\top \boldsymbol{u}^g,$$

which proves Eq. B.5 after division by $n_m n_n$.

Normalizing the field distributions and constructing all cached group sums costs $O(E\sum_m n_m)$. Computing their M squared norms and $M(M-1)/2$ pairwise inner products costs $O(EM^2)$. In contrast, explicit comparison of all field pairs across M groups of size n costs $O(EM^2n^2)$. □

B.2 Measuring Temporal Routing Consistency

B.2.1 Jaccard Similarity

Let $\mathcal{S}_{l,t}$ be the set of experts selected by MoE layer l for token t. The token-level Jaccard similarity between token t_1 and token t_2 is defined as

$$\mathbf{J}(t_1, t_2) = \frac{|\mathcal{S}_{l,t_1} \cap \mathcal{S}_{l,t_2}|}{|\mathcal{S}_{l,t_1} \cup \mathcal{S}_{l,t_2}|}. \tag{B.7}$$

Then, for an agentic trajectory containing N turns, $\tau = \left[y_1^{\text{think}}; y_1^{\text{tool}}; v_1; \ldots; y_N^{\text{think}}; y_N^{\text{tool}}; v_N\right]$, the within-field Jaccard similarity is

$$\mathbf{J}_{\text{field}}(\tau) = \frac{1}{|\mathcal{L}_{\text{MoE}}| \cdot N} \sum_{l \in \mathcal{L}_{\text{MoE}}} \sum_{i=1}^{N} \left[\frac{1}{2\binom{|y_i^{\text{think}}|}{2}} \sum_{t_a, t_b \in y_i^{\text{think}}} \mathbf{J}(t_a, t_b) + \frac{1}{2\binom{|y_i^{\text{tool}}|}{2}} \sum_{t_a, t_b \in y_i^{\text{tool}}} \mathbf{J}(t_a, t_b) \right], \tag{B.8}$$

which is an average over all within-field token pairs. The comparative metrics in Fig. 2(b) are defined similarly, with different ways of constructing token pairs.

Additionally, suppose each token independently selects a uniformly random set of k experts from E available experts. The expected intersection and union sizes are k^2/E and $2k - k^2/E$, respectively. Their ratio gives the approximate reference

$$\mathbf{J}_{\text{ref}} = \frac{k^2/E}{2k - k^2/E} = \frac{k}{2E - k}. \tag{B.9}$$

For `Qwen3-30B-A3B`, where $E = 128$ and $k = 8$, this gives $\mathbf{J}_{\text{ref}} \approx 0.032$, matching the dashed reference in Fig. 2(b).

B.2.2 Expert Switches

Prior work also uses the number of expert switches per token to quantify routing consistency (Yang et al., 2026). We define the expert switch count over all within-field token pairs:

$$\mathbf{ES}(\tau) = \frac{1}{|\mathcal{L}_{\text{MoE}}| \cdot N} \sum_{l \in \mathcal{L}_{\text{MoE}}} \sum_{i=1}^{N} \left[\frac{1}{|y_i^{\text{think}}|} \sum_{t \in y_i^{\text{think}}} (k - |\mathcal{S}_{l,t} \cap \mathcal{S}_{l,t+1}|) + \frac{1}{|y_i^{\text{tool}}|} \sum_{t \in y_i^{\text{tool}}} (k - |\mathcal{S}_{l,t} \cap \mathcal{S}_{l,t+1}|) \right], \tag{B.10}$$

consistent with our hierarchical framework, in which routing consistency is measured only within each turn.

B.3 Measuring Expert Load Balance

Following Shazeer et al. (2017), we use the coefficient of variation (CV) to quantify expert-load imbalance. For a trajectory τ, let $\mathcal{T}_\tau$ contain its generated thinking and tool-use tokens, and let $\mathcal{S}_{\ell,t}$ denote the top-k experts selected at MoE layer ℓ for token t. The assignment count of expert e is

$$c_{l,e}(\tau) = \sum_{t \in \mathcal{T}_\tau} \mathbb{I}[e \in \mathcal{S}_{l,t}]\,. \tag{B.11}$$

We define the load CV as

$$\mathbf{CV}_l(\tau) = \frac{\text{std}_e(c_{l,e}(\tau))}{\text{mean}_e(c_{l,e}(\tau))}, \tag{B.12}$$

where both statistics are computed over all experts in the layer, including those with zero assignments. We compute the unsquared CV separately for each trajectory and MoE layer, then average these values. Lower CV indicates more balanced expert usage, with zero corresponding to equal assignment counts across experts.

C Variant Routing Controls

In this section, we introduce the routing control variants explored in our early research. Although these methods were less promising overall than our final framework in Section 3, they help explain which routing signals are useful for agentic RL. Table 2 compares their task performance on AppWorld (Trivedi et al., 2024), with the best-performing hyperparameters selected through a grid search ranging from 10^{-7} to 10^{-5}. We do not apply the entropy gate and main experiments' hyperparameters for our routing controls.

Shared notation. We retain the notation from Section 2.2: M experts, top-k selections $\mathcal{S}_{l,t}$, router logits $\mathbf{z}_{l,t}$, probabilities $\mathbf{p}_{l,t} = \text{softmax}(\mathbf{z}_{l,t})$, and MoE layers $l \in \mathcal{L}_{\text{MoE}}$. Let $\mathcal{T}$ contain the token positions of one generated thinking or tool-use field, omitting its turn and field-type indices. Extending the assignment counts in Appendix B.3, define

$$c_{l,e}(\mathcal{T}) = \sum_{t \in \mathcal{T}} \mathbb{I}[e \in \mathcal{S}_{l,t}], \qquad \mathcal{C}_l^{(K)}(\mathcal{T}) = \text{TopK}_e\big(c_{l,e}(\mathcal{T}), K\big). \tag{C.1}$$

Here, TopK returns expert indices, and $\mathcal{C}_l(\mathcal{T}) = \mathcal{C}_l^{(k)}(\mathcal{T})$ is the majority-voted consensus. Thus, "majority-voted" refers to the most frequently selected experts, without requiring selection by more than half the tokens. We compute consensus separately for each field in a turn. All adjacent-token expressions below require at least two tokens and exclude field and turn boundaries.

C.1 Routing Regularization: Field Consensus

`routing_reg` encourages every token in a field to favor its majority-voted experts. We monitor agreement with the uniform distribution over $\mathcal{C}_l(\mathcal{T})$ using

$$\mathcal{L}_{l,\mathcal{T}}^{\text{cons}} = -\frac{1}{k|\mathcal{T}|} \sum_{t \in \mathcal{T}} \sum_{e \in \mathcal{C}_l(\mathcal{T})} \log p_{l,t}[e]. \tag{C.2}$$

Table 2: Comparison of routing control variants on AppWorld, measured by task goal completion (TGC). The highest and second-highest scores in each column are bolded and underlined, respectively.

Method	**test-normal**	**test-challenge**
	TGC (%)	TGC (%)
Base Model	34.52	19.66
GRPO	66.70	38.80
+ `routing_reg`	70.24	43.17
+ `switch_reg` (Yang et al., 2026)	72.02	<u>46.28</u>
+ `outlier_reg`	0.0 (crashed)	0.0 (crashed)
+ `routing_reward`	73.21	41.25
+ `routing_clip`	<u>73.81</u>	45.08
+ `load_bal` (Shazeer et al., 2017)	**74.40**	45.56
+ Ours	72.62	**51.80**

The above field-wide target can be too restrictive: different tokens within one field may require different experts, which explains its suboptimal performance.

C.2 Switch Regularization: Adjacent-Distribution Agreement

We adapt the temporal-consistency penalty of PA-MoE (Yang et al., 2026), which penalizes expert changes between adjacent tokens at each native MoE layer.

`switch_reg` directly extends the expert switch count into a differentiable objective. We use the full router distributions:

$$\ell_{l,t}^{\text{switch}} = 1 - \langle p_{l,t-1}, p_{l,t} \rangle, \qquad \mathcal{L}_{l,\mathcal{T}}^{\text{switch}} = \frac{1}{|\mathcal{T}| - 1} \sum_{t:\, t-1,t \in \mathcal{T}} \ell_{l,t}^{\text{switch}}. \tag{C.3}$$

The inner product is the probability that independent expert draws agree. Its complement therefore favors shared, concentrated preferences. Writing $d_{l,t} = \langle p_{l,t-1}, p_{l,t} \rangle$, the pair derivatives are

$$\begin{aligned} \nabla_{z_{l,t}} \ell_{l,t}^{\text{switch}} &= p_{l,t} \odot \big(d_{l,t}\mathbf{1} - p_{l,t-1}\big), \\ \nabla_{z_{l,t-1}} \ell_{l,t}^{\text{switch}} &= p_{l,t-1} \odot \big(d_{l,t}\mathbf{1} - p_{l,t}\big), \end{aligned} \tag{C.4}$$

where $\mathbf{1}$ is the all-ones vector and $\odot$ denotes elementwise multiplication. Both endpoints receive gradients; the neighboring probabilities are not detached from the overall surrogate.

`switch_reg` can destabilize RL training by changing the full routing distribution. In particular, even identical adjacent distributions incur $1 - \|p\|_2^2$, so the surrogate also favors concentration and can substantially alter router entropy. This explanation for the instability is also consistent with the sensitivity discussed in Appendix E.2. Router entropy here is different from the next-token policy entropy used for gating in Section 3.3. The evaluated `switch_reg` run improves AppWorld test-challenge TGC to 46.28%, but remains below our final framework.

C.3 Outlier Regularization

`outlier_reg` suppresses selected experts outside a field's consensus expert set. For a configurable consensus size K, define

$$\mathcal{O}_{l,t} = \mathcal{S}_{l,t} \setminus \mathcal{C}_l^{(K)}(\mathcal{T}), \qquad m_{l,t}^{\text{outlier}} = \sum_{e \in \mathcal{O}_{l,t}} p_{l,t}[e]. \tag{C.5}$$

The design aims to reduce this selected outlier mass by applying probability-weighted downward signals to outlier logits. Only selected experts outside the target expert set are penalized; the remaining experts receive no direct penalty. We explored $K = 8$ and $K = 16$.

In our experiments, suppressing these selections perturbed routing too strongly and caused severe performance degradation; the run in Table 2 crashed. This result motivated reinforcing existing local agreement instead of directly penalizing departures from a field-wide consensus.

C.4 Routing Reward and Routing Clipping

These two variants incorporate routing consistency into the policy gradient objective (Eq. 2.3). From recorded expert selections, we compute the detached consensus-overlap score

$$\kappa_t = \frac{1}{|\mathcal{L}_{\text{MoE}}|} \sum_{l \in \mathcal{L}_{\text{MoE}}} \frac{|\mathcal{S}_{l,t} \cap \mathcal{C}_l(\mathcal{T})|}{k}. \tag{C.6}$$

We set $\kappa_t = 1$ outside the configured fields. Let $\widehat{A}_t$ be the existing advantage and $r_t(\theta) = \pi_\theta(y_t \mid \tau_{<t}) / \pi_{\theta_{\text{old}}}(y_t \mid \tau_{<t})$ the policy probability ratio, where $\tau_{<t}$ includes the task and interaction context before the generated token y_t. Define the clipped maximization objective

$$\mathcal{J}_{\text{clip}} = \mathbb{E}_t[\min\{r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}})\, A_t\}]\,. \tag{C.7}$$

The expectation follows the base policy loss's mask and reduction over generated tokens.

Routing reward. We augment the standard advantage with a routing inconsistency penalty:

$$\widetilde{A}_t = \widehat{A}_t - \lambda_{\text{reward}}(1 - \kappa_t). \tag{C.8}$$

Lower consistency decreases the token's advantage, while perfect agreement adds no penalty. Despite its name, `routing_reward` does not recompute terminal task rewards or return targets.

Routing clipping. Another way to add a routing inconsistency penalty to the RL objective is through clipping. We define a scaling factor based on routing inconsistency:

$$\alpha_t = 1 - \lambda_{\text{clip}}(1 - \kappa_t), \tag{C.9}$$

and apply this factor to the clipping bounds:

$$\epsilon'_{\text{low}} = \alpha_t \cdot \epsilon_{\text{low}}, \quad \epsilon'_{\text{high}} = \alpha_t \cdot \epsilon_{\text{high}}. \tag{C.10}$$

For $\lambda_{\text{clip}} \in [0, 1]$, inconsistent routing narrows the clipping interval around one, while $\kappa_t = 1$ preserves the baseline interval. This does not impose a hard bound on the actual policy update.

However, neither variant above effectively changed the routing statistics in our experiments. Because κ_t is computed from discrete recorded selections, neither provides a differentiated consistency signal to the current router. Ordinary policy-loss gradients can still reach router parameters through differentiable expert-combination weights, as discussed in Appendix D.5, but they do not explicitly reinforce the consensus experts. Thus, adjusting advantages or clipping bounds alone was insufficient for the intended routing control.

C.5 Load-Balancing Loss

We also evaluate the auxiliary load-balancing loss (Shazeer et al., 2017) implemented by the Megatron engine (Shoeybi et al., 2019). For a microbatch containing T tokens, define the token frequency $f_{l,e}$ and average expert probability $\bar{p}_{l,e}$ as

$$\mathcal{S}^{\text{aux}}_{l,t} = \text{TopK}_e(p_{l,t}[e], k), \quad f_{l,e} = \frac{1}{kT} \sum_{t=1}^{T} \mathbb{I}[e \in \mathcal{S}^{\text{aux}}_{l,t}], \quad \text{and } \bar{p}_{l,e} = \frac{1}{T} \sum_{t=1}^{T} p_{l,t}[e]. \tag{C.11}$$

The above statistics are recomputed from current router probabilities, rather than taken from replayed routes. The minimized loss is

$$\mathcal{L}^{\text{load-bal}} = \frac{\lambda_{\text{load-bal}}}{|\mathcal{L}_{\text{MoE}}| \cdot M} \sum_{l \in \mathcal{L}_{\text{MoE}}} \sum_{e=1}^{M} f_{l,e} \bar{p}_{l,e}. \tag{C.12}$$

The frequencies $f_{l,e}$ are detached from the gradient path. Differentiation through $\bar{p}_{l,e}$ discourages further allocation to heavily used experts and favors less-used experts. It applies to router input tokens beyond the generated thinking and tool-use fields.

To understand the effectiveness of the operation-label-free load-balancing loss, we compare it with our routing control framework in Table 3. We set $\lambda_{\text{load-bal}} = 10^{-4}$ and use the same hyperparameters for our method as in the main experiments. The load balance (measured by the CV of expert load as defined in Appendix B.3) is improved, but the agentic task performance on `test-challenge` is not better until we apply the turn-level control. This demonstrates that operation labels are still required for MoE models to improve their hard task solving.

Table 3: Load balancing and task performance on AppWorld. Load-balancing loss effectively reduces the CV of expert load, but this does not translate to better empirical performance.

Method	**Expert-Load CV** ↓	**AppWorld TGC** (%) ↑	
		test-normal	test-challenge
Base Model	1.240	38.69	19.66
GRPO	1.241	69.64	46.28
+load-bal (Shazeer et al., 2017)	**1.155**	**74.40**	45.56
+token-level control & load-bal	1.321	69.05	43.17
+token-level & turn-level control (ours)	1.295	72.62	**51.80**

D Technical Details

In this section, we introduce the technical details of our experiments, particularly MoE model architecture (Appendix D.1), agentic RL infrastructure (Appendix D.2), hyperparameters (Appendices D.3 and D.4), gradient flow design (Appendix D.5), and operation-label detection (Appendix D.6).

D.1 MoE Model Architecture

We take `Qwen3-30B-A3B` as an example to introduce a few interesting facts about MoE model architecture:

Router module. The router module in each MoE layer is just a linear projection (`torch.nn.Linear`). `Qwen3-30B-A3B` does not even have a bias term. Therefore, the hidden states of the context are more decisive in choosing which experts to use (Wang et al., 2026c). Our proposed routing control framework actively changes the hidden states of each MoE layer, which contributes to its instability.

When does expert routing happen? Inference in MoE models can be divided into prefilling (processing prompts) and decoding (generating new tokens) stages. Expert routing occurs in both stages, and each token in the trajectories is assigned a set of experts. However, we discuss only expert routing during decoding, as multi-turn agentic trajectories pertain only to the decoding stage.

MoE decoding fusion. Production inference engines such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) typically fuse the routing operations (computing router scores + top-k sampling + activating selected experts) into one kernel. Breaking up this kernel will make the inference much less efficient. This also motivates our routing control design, updating the router weights rather than directly assigning expert indices to the routers.

D.2 Agentic RL Infrastructure

We adapt slime (Zhu et al., 2025) and extend its default rollout functions to support agents. Agentic trajectories typically contain multiple interactions between agents (generating tokens) and environments (executing code). The number of turns, the token count in each turn, and the execution time are highly variable, which makes standard batched inference impossible. We therefore isolate the decoding of each trajectory into a process, separate from the SGLang server processes. We attach an environment sandbox to each trajectory process, enabling parallel code execution. The high-level idea of the agentic RL rollout framework is summarized in Fig. 9.

D.3 Inference Settings

We summarize the full inference settings (particularly for evaluation) in Table 4, balancing inference speed and GPU memory capacity (we use B200 nodes[2] throughout the experiments).

D.4 Training Settings

For AppWorld (Trivedi et al., 2024), we train `Qwen3-30B-A3B-Instruct-2507`[3] for 200 steps. At each step, we sample 10 prompts and generate 16 rollout trajectories per prompt at temperature 1.0, yielding a

[2] https://www.nvidia.com/en-us/data-center/dgx-b200/.

[3] Multi-turn trajectories are very long, exceeding the context window of the older `Qwen3-30B-A3B`.

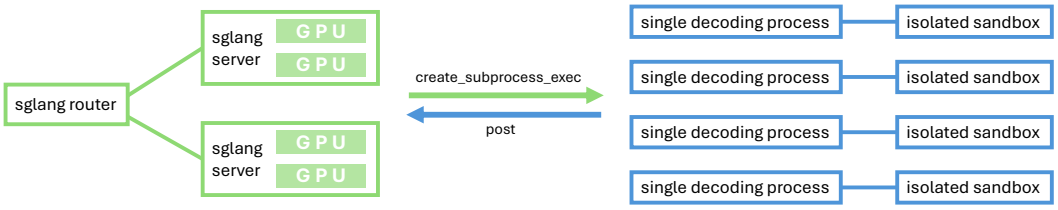


Figure 9: Rollout framework overview. Each trajectory is handled by a separate process with its sandbox for environment execution. SGLang servers generate tokens in response to HTTP requests from trajectory processes.

Table 4: Evaluation inference settings. Sampling parameters apply at each assistant turn. TP and EP denote tensor and expert parallelism. [a]AppWorld uses 160 concurrent trajectories, except for PPO, which uses 80. [b]AutomationBench uses 144, except for PPO, which uses 40. Concurrency limits are totals across both engines. The explicit context cutoff is checked before each generation turn; an unset cutoff does not remove model or server context limits.

Setting	**AppWorld**	**AutomationBench**
Model	`Qwen3-30B-A3B-Instruct-2507`	`Qwen3.5-35B-A3B`
Inference engine	SGLang	SGLang
Inference GPUs	8	8
Engine replicas	2	2
TP / EP per engine	4 / 4	4 / 4
Temperature	0.7	0.7
Nucleus sampling (p)	0.8	0.8
Top-k sampling	20	20
Evaluation trajectories per task	1	1
Thinking mode	Disabled	Disabled
Presence-penalty override	Not set	Not set
Maximum generated tokens per turn	1,500	2,048
Maximum interaction turns	50	50
Explicit context cutoff (tokens)	Not set	65,536
Maximum concurrent trajectories	160 / 80[a]	144 / 40[b]

nominal batch of 160 trajectories. To reduce waiting for slow rollouts, we terminate the remaining trajectories once every prompt has at least 12 completed trajectories and at least 90% of the full batch (144 trajectories) has completed; completion includes both successful and unsuccessful task outcomes. Aborted trajectories are masked out of the training loss and excluded from group reward normalization. Training trajectories are limited to 30 interaction turns, with at most 1,500 generated tokens per turn and a 3,000-token cap on each environment execution output. We optimize using Adam (Kingma & Ba, 2015) with a constant learning rate of 10^{-6}, gradient-norm clipping at 0.1, and policy-ratio clipping with $\epsilon = 0.2$; both the KL-penalty and entropy-bonus coefficients are set to zero. We evaluate every 10 training steps, allowing up to 50 interaction turns per evaluation trajectory.

D.5 Injected Gradient Flow in RL Regularization

The RL loss (Eq. 2.2) is computed from the output token probabilities of $\pi_{\boldsymbol{\theta}}$. Its gradients can reach router logits indirectly through differentiable expert-combination weights, but do not explicitly promote operation specialization or local routing consistency. We therefore inject auxiliary gradients directly at the router logits $\mathbf{z}_{l,t}$. This supplies explicit routing supervision alongside the standard RL update, without changing the current forward computation.

We introduce $\rho_{l,t}$ to help balance the magnitudes of the RL and routing-control gradients. Let $\mathbf{g}_{l,t}^{\text{in}}$ denote the gradient arriving at $\mathbf{z}_{l,t}$ before auxiliary injection at layer l. For a token position t in a turn $\mathcal{T}$, define

$$\rho_{l,t} = \frac{\|\mathbf{g}_{l,t}^{\text{in}}\|_2}{\max\left\{10^{-8},\, \max_{u\in\mathcal{T}} \|\mathbf{g}_{l,u}^{\text{in}}\|_2\right\}}. \tag{D.1}$$

This factor lies in $[0, 1]$: tokens with weaker incoming gradients receive smaller auxiliary updates, and tokens with zero incoming gradient receive none. The coefficients λ_{turn} and λ_{token} set the overall auxiliary strength, while $\rho_{l,t}$ adjusts it across tokens within each turn. Thus, $\rho_{l,t}$ supports balancing the two training signals but does not by itself enforce equal gradient magnitudes. We compute it once before injection at each layer and share it between both controls.

Using the turn-level objective J_{turn} from Section 3.1 and the token-level objective J_{token} from Section 3.2, the modified backward rule is

$$\mathbf{g}_{l,t}^{\text{out}} = \mathbf{g}_{l,t}^{\text{in}} - g_s\, \rho_{l,t} \left(\lambda_{\text{turn}} \nabla_{\mathbf{z}_{l,t}} J_{\text{turn}} + \lambda_{\text{token}} \nabla_{\mathbf{z}_{l,t}} J_{\text{token}}\right), \tag{D.2}$$

where g_s is the policy-entropy gate from Section 3.3. The auxiliary derivatives alter the gradient directions through the full router distribution $\mathbf{p}_{l,t}$. Scaling factors, gates, and discrete expert targets are treated as constants during the backward pass.

We apply this rule at every MoE layer, separately within generated thinking and tool-use fields. A custom autograd operation returns $\mathbf{z}_{l,t}$ from the forward pass and adds the auxiliary gradients in the backward pass. The resulting gradient propagates through $\mathbf{z}_{l,t} = W_l \mathbf{h}_{l,t}$ by the chain rule, updating both the router weights and the representations right before the router. We summarize the actual computation process of our routing control framework in Algorithm 1.

Algorithm 1 Details of Agentic RL with Routing Control

```
for each training step do
    Generate trajectories and extract labels and field boundaries.
    Compute the RL loss.
    for each MoE layer during backpropagation do
        Collect routing statistics.
        Compute scaling factors from incoming gradients.
        if entropy gate is open then
            Apply token control to similar adjacent routes within fields.
            Apply turn control.
            Scale and add the auxiliary updates.
        end if
        Continue backpropagation.
    end for
    Update model parameters.
end for
```

D.6 Rubrics for Grouping Agentic Operations

As a key component of the proposed routing-control framework, our rubric for detecting operation labels is simple, deterministic, and task-dependent. We apply hard-coded rules to the auxiliary information in each turn's code field. A turn may contain multiple operations; therefore, its label is the sorted set of unique operation categories detected in that turn. The resulting turn-level labels are assigned to both the thinking and tool-use fields.

D.6.1 AppWorld Rubrics

For AppWorld (Trivedi et al., 2024), we extract API calls of the form `apis.<app>.<api>` from the Python code generated at turn t. Let

$$\mathcal{C}_t^{\text{AW}} = \{(a_i, p_i)\}_{i=1}^{N_t}, \tag{D.3}$$

where a_i and p_i denote the application and API names of the i-th call, respectively. We map each call to one of five operation categories:

$$\mathcal{O}_{\text{AW}} = \{\textsc{Auth}, \textsc{Read}, \textsc{Create}, \textsc{Update}, \textsc{Delete}\}. \tag{D.4}$$

The operation classifier follows the ordered rules

$$g_{\mathrm{AW}}(a,p) = \begin{cases} \textsc{Auth}, & p \in \mathcal{P}_{\mathrm{auth}}, \\ \textsc{Read}, & \mathrm{prefix}(p) \in \mathcal{P}_{\mathrm{read}}, \\ h(\mu(a,p)), & \mu(a,p) \text{ is available}, \\ q(\mathrm{prefix}(p)), & \text{otherwise}, \end{cases} \tag{D.5}$$

where $\mathcal{P}_{\mathrm{auth}}$ contains authentication APIs such as `login`, `logout`, and `signup`; $\mathcal{P}_{\mathrm{read}}$ contains read-oriented prefixes such as `show`, `list`, `search`, `get`, and `download`; and $\mu(a,p)$ returns the HTTP method associated with the API. The HTTP-method mapping is

$$\begin{aligned} h(\texttt{GET}) &= \textsc{Read}, & h(\texttt{POST}) &= \textsc{Create}, \\ h(\texttt{PATCH/PUT}) &= \textsc{Update}, & h(\texttt{DELETE}) &= \textsc{Delete}. \end{aligned} \tag{D.6}$$

The read-prefix rule is applied before the HTTP-method rule because AppWorld implements several read-only, RPC-style APIs using `POST`. For example, `apis.splitwise.show_person_balance(...)` is labeled READ, despite being implemented as a `POST` endpoint.

The final multi-label annotation for turn t is

$$\mathrm{sort}\Big(\mathrm{unique}\Big\{g_{\mathrm{AW}}(a,p) \mid (a,p) \in \mathcal{C}_t^{\mathrm{AW}}\Big\}\Big). \tag{D.7}$$

A turn may have multiple operation labels. For example, a turn containing both `create_task(...)` and `delete_task(...)` gets {CREATE, DELETE}.

D.6.2 AUTOMATIONBENCH RUBRICS

For AutomationBench (Shepard & Salimans, 2026), operation detection is performed directly on the structured tool calls generated at turn t. Let

$$\mathcal{C}_t^{\mathrm{Auto}} = \{(n_i, x_i)\}_{i=1}^{N_t}, \tag{D.8}$$

where n_i is the tool name and x_i is its argument dictionary. We use the operation categories

$$\mathcal{O}_{\mathrm{Auto}} = \{\textsc{Search}, \textsc{Read}, \textsc{Create}, \textsc{Update}, \textsc{Delete}, \textsc{None}\}. \tag{D.9}$$

Unlike AppWorld, AutomationBench does not require an external API-to-method lookup: the HTTP method is explicitly included in each `api_fetch` call. We define

$$g_{\mathrm{Auto}}(n,x) = \begin{cases} \textsc{Search}, & n = \texttt{api_search}, \\ \textsc{Read}, & n = \texttt{api_fetch},\ x[\texttt{method}] \in \{\texttt{GET}, \texttt{HEAD}\}, \\ \textsc{Create}, & n = \texttt{api_fetch},\ x[\texttt{method}] = \texttt{POST}, \\ \textsc{Update}, & n = \texttt{api_fetch},\ x[\texttt{method}] \in \{\texttt{PATCH}, \texttt{PUT}\}, \\ \textsc{Delete}, & n = \texttt{api_fetch},\ x[\texttt{method}] = \texttt{DELETE}, \\ \textsc{None}, & \text{otherwise}. \end{cases} \tag{D.10}$$

Local formatting tools, such as `base64_encode`, and malformed or unsupported tool calls are not assigned an operation label.

The turn-level annotation is consequently

$$\mathrm{sort}\Big(\mathrm{unique}\Big\{g_{\mathrm{Auto}}(n,x) \mid (n,x) \in \mathcal{C}_t^{\mathrm{Auto}}\Big\}\Big). \tag{D.11}$$

A turn may have multiple operation labels. For example, a turn containing an `api_search` call, a `GET` request, and two `POST` requests receives {SEARCH, READ, CREATE}.

E ADDITIONAL DISCUSSION

In this section, we discuss a few interesting points about MoE routing control. These discussions complement our design choices in Section 3.

E.1 IS IT POSSIBLE TO MANUALLY FIX THE WITHIN-FIELD ROUTING?

It is true that production inference engines do not support manually setting expert selections (Appendix D.1), but it is still interesting to examine the effects of doing so. We use the naive inference function from HF Transformers (Wolf et al., 2020) and manually set the expert selection to be exactly the field-wise consensus set (Eq. C.1).

The results are shown in Table 5. We compute mean log-likelihoods for a set of example AppWorld trajectories using default and fixed routing strategies. We observe that enforcing fixed expert selection substantially degrades model likelihoods. Replacing token-specific routing with each field's majority-selected experts reduces mean token log-probability. These results demonstrate that hard routing constraints can induce a collapse in likelihood.

Table 5: Enforcing fixed expert routing substantially degrades trajectory likelihood. Generation log-likelihoods are computed from a set of example trajectories. Higher mean log-probability is better. $\Delta = 2 \cdot (\ell_{\text{fixed}} - \ell_{\text{default}})/(\ell_{\text{fixed}} + \ell_{\text{default}})$ is the relative decrease.

Trajectory	**Generation Log-Likelihood**		
	Default Routing	Fixed Routing	Δ (%)
positive ($R = 1$)	-0.2778	-1.5897	140.51
negative ($R = 0$)	-0.2795	-1.5292	138.19

E.2 Is It Possible to Directly Control the Routers' Entropy?

The stability of MoE RL is closely related to entropy, but the total action entropy from the token distributions is not enough to understand the MoE models' inference behavior. We also examine router entropy.

Entropy decomposition. For a fixed context x, consider a stochastic router that samples an expert $R \sim q_\phi(\cdot \mid x)$ and then the expert samples an action $A \sim \pi_\theta(\cdot \mid R, x)$. Here, $A \in \mathbb{V}$ is the next token. We distinguish three quantities: the total action entropy $H(A \mid x)$, the expected action entropy with the route fixed $\mathbb{E}_r H(A \mid R = r, x)$, and the router entropy $H(q_\phi(\cdot \mid x))$.

Lemma 2 (Entropy Decomposition)**.** *The total action entropy satisfies*

$$H(A \mid x) \le \min\Big\{\log|\mathbb{V}|,\ \mathbb{E}_{r\sim q_\phi(\cdot|x)} H(A \mid R = r, x) + H(q_\phi(\cdot \mid x))\Big\}.$$

Proof. The chain rule gives

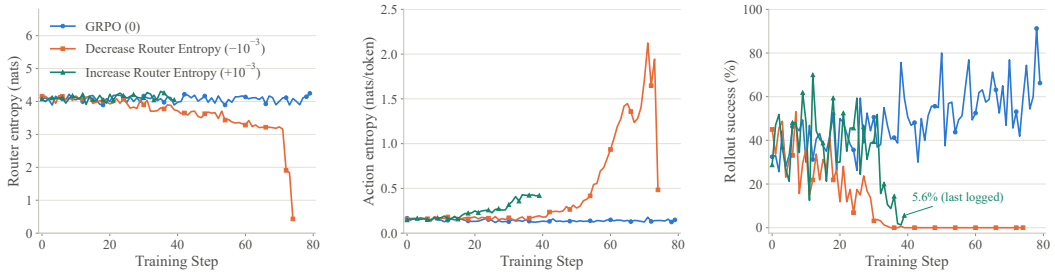
$$H(A \mid x) = \mathbb{E}_r H(A \mid R = r, x) + H(R \mid x) - H(R \mid A, x).$$

Since $H(R \mid A, x) \ge 0$, we can upper-bound the total action entropy by

$$\begin{aligned} H(A \mid x) &\le \mathbb{E}_r H(A \mid R = r, x) + H(R \mid x) \\ &= \mathbb{E}_r H(A \mid R = r, x) + H(q_\phi(\cdot \mid x)). \end{aligned}$$

Given the numerical entropy upper bound $\log|\mathbb{V}|$, the inequality holds. □

Failure cases of router entropy regularization. Fig. 10 compares training with and without direct regularization of router softmax entropy. The router entropy loss is similar to the standard total action entropy loss, and we set the coefficients to $\pm 10^{-3}$. We observe training collapse when perturbing the routers' entropy. Small router-score changes near the top-k boundary can replace active experts and disrupt subsequent hidden representations, which explains the explosion in total action entropy and training collapse.



(a) Router Entropy **(b)** Total Action Entropy **(c)** Training Rollout Success

Figure 10: Training curves for GRPO with direct router-entropy regularization. Direct entropy control causes an explosion in total action entropy and training collapse.

E.3 Why Does Routing Consistency Improve Efficiency?

As demonstrated in Section 4.4, encouraging field-wise routing consistency improves the efficiency of MoE models. However, this effect has not been fully understood yet. We propose a controlled analysis here to reveal the mechanism behind this efficiency improvement.

As inference engines such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) encapsulate routing in fused kernels, it is technically hard to manually set the expert selection during inference. Instead, we propose a simple weight matrix mixing trick to reduce the rank of the router's weight matrices. This low-rank mixing can effectively reduce the expert shifts. Specifically, we denote the router's weight matrix as $\mathbf{W} \in \mathbb{R}^{E \times H}$ and the hidden states as $\mathbf{h} \in \mathbb{R}^{H}$. The router's logit for expert e is computed as:

$$\text{logit}[e] = \mathbf{W}[e] \cdot \mathbf{h}. \tag{E.1}$$

To manually reduce expert shifts when decoding consecutive tokens, we aim to restrict expert selection to a smaller set. For example, we can force the router logits to be exactly the same for all experts, and then only the first 8 experts will be used. This approach is equivalent to reducing the rank of the routers' weight matrices. Specifically, we categorize experts into G groups according to their indices:

$$\underbrace{e_1, e_2, ..., e_K}_{\text{Group 1}}, \underbrace{e_{K+1}, e_{K+2}, ..., e_{2K}}_{\text{Group 2}}, ..., \underbrace{e_{(G-1)K+1}, e_{(G-1)K+2}, ..., e_{GK}}_{\text{Group } G}. \tag{E.2}$$

Then, for each group, its corresponding rows in the low-rank weight matrix are:

$$\mathbf{W}'[e] = \frac{1}{K} \sum_{e' \in \text{group}(e)} \mathbf{W}[e'], \tag{E.3}$$

where $\text{group}(e)$ is the group to which expert e belongs. This makes the rows within each group exactly the same, fixing any expert selection within a group to be its first expert. Then the new logits are computed as:

$$\text{logit}'[e] = \mathbf{W}'[e] \cdot \mathbf{h}. \tag{E.4}$$

By varying G, we can manually control the number of shifted experts.

Additionally, for small G, the number of experts in each group exceeds that in each EP rank, and thus only a portion of GPUs are utilized during inference. We address this bug by re-ranking experts within each bucket defined by group and EP rank:

$$\mathbf{W}'[e] = 0.5^{\text{tier}(e)} \cdot \frac{1}{K} \sum_{e' \in \text{group}(e)} \mathbf{W}[e'], \tag{E.5}$$

where $\text{tier}(e)$ is the order of expert e in its bucket defined by group and EP rank. Even with $G = 1$ (fixed 8 experts), we can now guarantee that all GPUs are utilized.

Based on the above router-collapsing procedure, we compare different routing consistency levels across various batch sizes and expert parallelism (EP) settings (Lepikhin et al., 2021). The results are shown in Fig. 11. First, whether EP is enabled does not influence inference efficiency. More importantly, the efficiency gain from more consistent routing benefits only inference with batch sizes larger than 1.

These observations support a batching-based explanation. For a decoding batch containing B tokens with top-k routing, the total number of token–expert assignments remains Bk. If these assignments are distributed across fewer active experts, the average number of tokens per active expert increases. This can improve expert-weight reuse and the efficiency of grouped expert computation. At batch size 1, each selected expert receives only one token, so this batching opportunity is absent. The observed batch-size dependence is consistent with this mechanism.

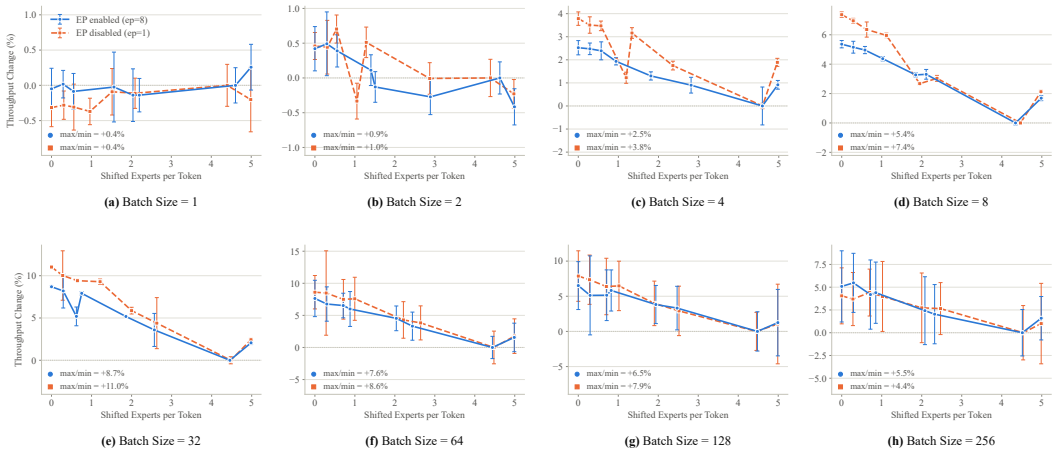
Our intervention is a synthetic diagnostic rather than a quality-preserving inference configuration. We therefore interpret Fig. 11 as evidence that routing structure can improve batched inference efficiency, which is exactly the real evaluation setting in our experiments.

F Additional Experiments

In this section, we present the full experimental results to complement Section 4 and help explain the effectiveness of each component in our framework as well as other applicable techniques.

F.1 Ablation Study on Turn Labels

As in Fig. 2(a), we also compare the full training curves with turn-level routing control based on different labeling schemes in Table 6. In addition to the operation labels used in the main experiments, we also compare fully random labels and application-name labels. We observe that operation labels provide the best labeling scheme for improving agentic RL performance.



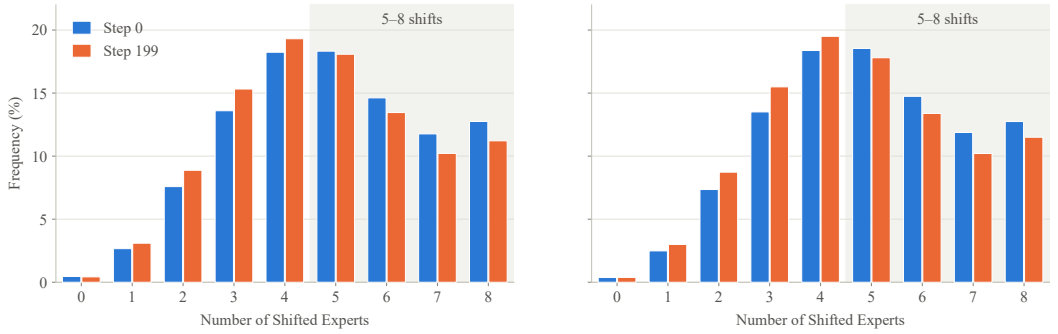
(a) Batch Size = 1 **(b)** Batch Size = 2 **(c)** Batch Size = 4 **(d)** Batch Size = 8

(e) Batch Size = 32 **(f)** Batch Size = 64 **(g)** Batch Size = 128 **(h)** Batch Size = 256

Figure 11: Controlled routing versus inference throughput. We manually collapse the routers' weight matrices to ensure consistent expert selection. The efficiency gain from consistent routing primarily comes from parallelism across a batch of trajectories decoded simultaneously.

Table 6: AppWorld TGC (%) at training step 199 for `Qwen3-30B-A3B-Instruct-2507` trained with GRPO and turn-level routing control ($\lambda_{\text{MI}} = 2 \times 10^{-6}$). All three runs share the same saved training settings apart from the labeling schemes. Labeling each turn by its operations yields the best performance.

Turn Label	# Classes	test-normal	test-challenge
Random	5	66.67	39.33
Application Name	12	66.07	40.53
Operation	5	**70.83**	**42.69**



(a) Routing Regularization **(b)** Switch Regularization

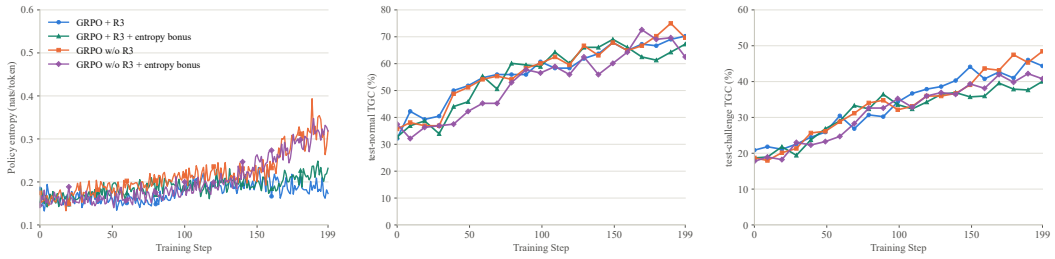
Figure 12: Histogram of expert shifts. After RL training with `routing_reg` (Appendix C.1) and `switch_reg` (Appendix C.2), high expert-shift counts consistently become less frequent.

F.2 Histogram of the Number of Shifted Experts

We visualize the histogram of the actual number of expert shifts in Fig. 12. We observe that, with appropriate routing controls, RL training effectively reduces expert shifts, and more tokens' expert selections are consistent with those of the preceding tokens.

F.3 Effect of R3 on Agentic RL

Rollout Routing Replay (R3) (Ma et al., 2026) is designed to enforce alignment between inference and training routing and prevent unintended routing drift. We evaluate its effectiveness for agentic RL tasks in our experiments. The results are shown in Fig. 13. Although removing R3 yields slightly better performance, it introduces instability into RL training. For example, without R3, the entropy curve may change sharply with a risk of explosion. Therefore, for better-controlled experiments, R3 should always be applied.

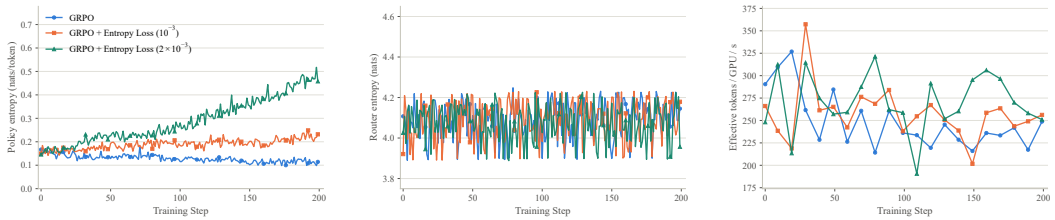


(a) Policy Entropy **(b)** AppWorld Normal-Set **(c)** AppWorld Challenge-Set

Figure 13: Comparison of training curves with R3. On agentic tasks, R3 can stabilize the entropy dynamics, preventing potential explosion. However, this stability does not translate to better performance due to the loss of RL exploration.

F.4 Ablation Study on Entropy Loss

As we observe that the inference efficiency improvement from our routing control framework often comes with higher total action entropy, we also investigate whether entropy alone can improve efficiency. In Fig. 14, we directly apply entropy loss to agentic RL and observe that changes in total action entropy do not translate into changes in routing behavior and have no observable effect on inference efficiency.



(a) Policy Entropy **(b)** Router Entropy **(c)** Evaluation Throughput

Figure 14: Comparison of training curves with entropy loss. We confirm that entropy loss has no effect on the routers' behavior or inference efficiency.

F.5 Ablation Study on the Token-Level Control Threshold

To determine the best threshold h_δ as used in Eq. 3.4, we compare different h_δ values in Table 7. We observe that $h_\delta = 0.5$ is the best value. Smaller thresholds reduce our framework to standard MoE-agnostic RL; larger thresholds significantly destabilize RL training.

F.6 Hyperparameter Magnitude Search

To empirically determine the best coefficient magnitudes for turn-level and token-level control, we compare a range of coefficients in Tables 8 and 9, respectively. For both control, we observe optimal magnitude ranges for achieving the best performance and avoiding training collapse. Larger magnitudes require the entropy gate introduced in Section 3.3 to prevent unstable router updates.

Table 7: Token-level routing control gap thresholds, evaluated using AppWorld TGC (%) with GRPO. Allowing too few expert shifts reduces our framework to standard RL; allowing too many expert shifts significantly destabilizes RL training.

Threshold	**Eval Step**	**test-normal**	**test-challenge**	**Collapse Step**
$0/8$	199	64.88	35.97	–
$2/8$	199	61.31	39.09	–
$4/8$	199	66.67	42.21	–
$6/8$	159	0.00	0.00	149
$8/8$	139	0.00	0.00	129

Table 8: Hyperparameter search for turn-level routing control. We apply only turn-level control to GRPO to determine the appropriate magnitude.

Setting	test-normal TGC (%)	test-challenge TGC (%)	Entropy	Tokens/GPU/s	Shifted Experts
$\lambda_{\text{turn}} = 0$ (GRPO)	66.67	38.85	0.1842	251.13	4.8393
$\lambda_{\text{turn}} = 10^{-3}$	0.00	0.00	0.2235	276.37	4.6192
$\lambda_{\text{turn}} = 10^{-4}$	0.60	0.96	0.1736	223.55	4.7340
$\lambda_{\text{turn}} = 10^{-5}$	63.69	36.93	0.1801	277.63	4.8370
$\lambda_{\text{turn}} = 10^{-6}$	63.69	41.73	0.1926	290.86	4.8425
$\lambda_{\text{turn}} = 10^{-7}$	68.45	41.49	0.1769	278.06	4.8247

Table 9: Hyperparameter search for token-level routing control. We apply only token-level control to GRPO to determine the appropriate magnitude. TGC is reported at step 199 when available and at the last evaluation otherwise: † denotes step 59 and ‡ denotes step 79.

Setting	test-normal TGC (%)	test-challenge TGC (%)	Entropy	Tokens/GPU/s	Shifted Experts
$\lambda_{\text{token}} = 0$ (GRPO)	64.88	35.97	0.1628	305.34	4.8263
$\lambda_{\text{token}} = 10^{-4\dagger}$	0.00	0.00	0.2649	394.55	4.5584
$\lambda_{\text{token}} = 10^{-5\dagger}$	0.00	0.00	0.2639	364.24	4.6042
$\lambda_{\text{token}} = 10^{-6\ddagger}$	0.00	0.00	0.2010	295.37	4.7318
$\lambda_{\text{token}} = 10^{-7}$	66.67	42.21	0.1448	306.03	4.8191

F.7 Ablation Study on Routing Control Components

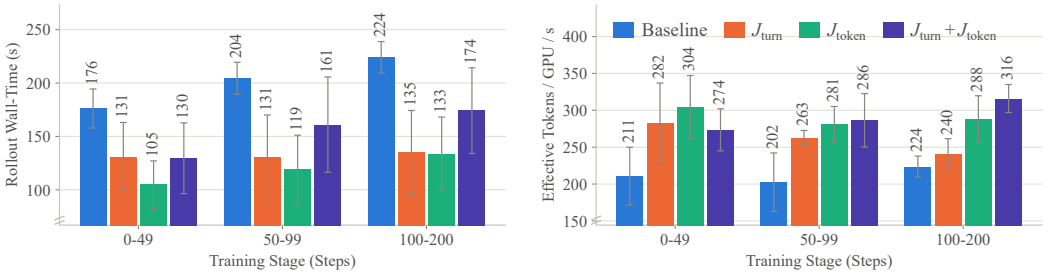
We compare the effect of each individual components of our routing control framwork in Table 10, evaluating the last checkpoints of 200-step training. Turn-level control contributes more to the task performance, and token-level control contributes more to the inference efficiency.

Table 10: Individual and combined effects of routing controls on AppWorld. Turn-level control primarily improves model capability on hard tasks, while token-level control primarily improves inference throughput.

Setting	**test-normal** TGC (%)	**test-challenge** TGC (%)	**Entropy**	**Tokens/GPU/s**	**Shifted Experts**
GRPO	66.67	38.85	0.1842	251.13	4.8393
+ Turn-Level Control	62.50	42.45	0.1943	**313.52**	4.8132
+ Token-Level Control	67.86	41.73	0.2053	277.74	**4.6676**
+ Both	**75.00**	**49.20**	**0.2331**	312.94	4.7416

F.8 Ablation Study on Rollout Efficiency

We also compare different measurement of rollout efficiency in Fig. 15, where we compare the actual rollout time. The conclusion persists that our routing control framework, especially the token-level control, significantly improves the rollout efficiency.



(a) Rollout Time **(b)** Inference Throughput

Figure 15: AppWorld rollout efficiency comparison under GRPO with turn-level control, token-level control, or both. (a) Both controls reduce the rollout time for a single trajectory. (b) Both controls improve inference throughput. The token-level control contributes the most to inference efficiency at the early training stage. The reported numbers are averaged over all of the evaluation trajectories collected from the specified training step intervals. The error bars are the standard deviation of all data points.

G Agentic Trajectory Examples

G.1 AutomationBench Trajectory

We show a successful trajectory on the AutomationBench task `operations.pipefy_gmail_vendor_approval` from `automation-public-eval`, rollout 0. The example contains 17 assistant turns, including the final response, and 35 tool calls. Calls issued together remain in the same turn, with feedback listed in the corresponding order. We retain the recorded response-field text, tool-use arguments, and final response; lengthy API-search outputs and irrelevant email fields are explicitly omitted. The heading "Thinking" displays the recorded response field; empty fields are marked explicitly. XML tool-call wrappers are removed and their arguments are displayed as JSON, retaining JSON-string parameters. JSON whitespace and Markdown formatting are adjusted for readability.

Task instruction (summary). Process unread vendor-review emails from `procurement@company.example.com`. Match the vendors to their Pipefy cards in `tbl_ops`, update each card's status and phase according to the review decision, mark the emails as read, and send confirmation emails to procurement. This is a summary from the trajectory and evaluation assertions; the original task prompt is not included in the log.

Model	`Qwen3.5-35B-A3B`	**Rollout index**	0
Domain	Operations	**Sample index**	53
Assistant turns	17	**Tool calls**	35
Recorded score	1.0	**Scored assertions passed**	10/10

Operation labels. Following Appendix D.6, `api_search` is labeled `Search`, whereas `api_fetch` is labeled by its HTTP method: `GET` as `Read` and `POST` as `Create`. Thus, the Pipefy field updates and phase moves, Gmail label changes, and email sends are all `Create` under this rule. The `base64_encode` calls in Turn 15 perform local formatting and receive the empty label set $\emptyset$; the final response in Turn 17 also has the empty label set. Each turn's label set applies to both its response and tool-use fields.

Outcome. The agent retrieves four vendor-review emails and resolves their corresponding Pipefy cards after an initially empty title lookup. It issues four status updates and four phase moves, marks the four messages as read, and sends four confirmation emails. The log records `task_completed_correctly` $= 1.0$, all 10 scored assertions passed, and zero tool errors. Three additional unscored negative assertions also pass, including

that the similarly named Northwind card `card_902` was not moved. Agent explanations and environment responses are reproduced as recorded.

Turn 1/17 {Search} **Agent**

Thinking. *No text recorded.*
Call 1: `api_search`.

```
{
  "query": "unread emails from procurement@company.example.com vendor reviews",
  "top_k": 10
}
```

Call 2: `api_search`.

```
{
  "query": "Pipefy tbl_ops cards update phase status",
  "top_k": 5
}
```

Turn 1/17 **Environment (excerpts)**

Call 1 (excerpt).

```
{
  "results": [
    {
      "id": "quickbooks.bill.create",
      "method": "POST",
      "description": "Enter a new bill from a vendor. Requires a vendor reference and at
   least one line item, which can be item-based or account-based.",
      "url": "https://quickbooks.api.intuit.com/v3/company/{companyId}/bill",
      ... [other schema fields omitted]
    },
    ... [9 other search results omitted]
  ],
  "count": 10
}
```

Call 2 (excerpt).

```
{
  "results": [
    {
      "id": "pipefy.cards.move",
      "method": "POST",
      "description": "Relocate a Pipefy card to another phase (column) within a pipe.",
      "url": "https://api.pipefy.com/graphql/cards/{cardId}/move",
      ... [other schema fields omitted]
    },
    {
      "id": "pipefy.cards.updateField",
      "method": "POST",
      "description": "Modify the value of a specific field on a Pipefy card.",
      "url": "https://api.pipefy.com/graphql/cards/{cardId}/fields:update",
      ... [other schema fields omitted]
    },
    ... [3 other search results omitted]
  ],
  "count": 5
}
```

Turn 2/17 {Search} **Agent**

Thinking. *No text recorded.*
Call 1: `api_search`.

```
{
  "query": "messages unread inbox",
  "top_k": 5
}
```

Turn 2/17 — **Environment (excerpt)**

Call 1 (excerpt).


```
{
  "results": [
    ... [1 other search result omitted],
    {
      "id": "gmail.users.messages.modify",
      "method": "POST",
      "description": "Add or remove labels on a specific message. Useful for archiving,
  toggling read/unread state, or applying custom labels.",
      "url": "https://gmail.googleapis.com/gmail/v1/users/{userId}/messages/{id}/modify
  ",
      ... [other schema fields omitted]
    },
    ... [1 other search result omitted],
    {
      "id": "gmail.users.messages.list",
      "method": "GET",
      "description": "Retrieve messages from the user's mailbox, optionally narrowed by
  label, sender, subject, or a search query.",
      "url": "https://gmail.googleapis.com/gmail/v1/users/{userId}/messages",
      ... [other schema fields omitted]
    },
    ... [1 other search result omitted]
  ],
  "count": 5
}
```


Turn 3/17 {Read} — **Agent**

Thinking. *No text recorded.*
Call 1: `api_fetch`.


```
{
  "method": "GET",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages",
  "params": "{\"q\": \"from:procurement@company.example.com is:unread vendor review\",
  \"maxResults\": 50}"
}
```


Turn 3/17 — **Environment**

Call 1.


```
{"messages": [{"id": "msg_rev_002", "threadId": "thr_rev_002"}, {"id": "msg_rev_003", "
  threadId": "thr_rev_003"}, {"id": "msg_rev_004", "threadId": "thr_rev_004"}, {"id": "
  msg_rev_005", "threadId": "thr_rev_005"}], "resultSizeEstimate": 4}
```


Turn 4/17 {Read} — **Agent**

Thinking. *No text recorded.*
Call 1: `api_fetch`.


```
{
  "method": "GET",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/msg_rev_002"
}
```


Call 2: `api_fetch`.


```
{
  "method": "GET",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/msg_rev_003"
}
```


Call 3: `api_fetch`.


```
{
  "method": "GET",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/msg_rev_004"
}
```


Call 4: `api_fetch`.


```
{
```

Turn 4/17 {Read} **Agent (continued)**


```
  "method": "GET",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/msg_rev_005"
}
```


Turn 4/17 **Environment (excerpts)**

Call 1 (excerpt).


```
{
  "id": "msg_rev_002",
  "from": "procurement@company.example.com",
  "to": ["ops@company.example.com"],
  "subject": "Vendor Review - Summit",
  "body_plain": "Summit review complete. Decision: Approved.",
  "label_ids": ["INBOX", "UNREAD"],
  ... [other message fields omitted]
}
```


Call 2 (excerpt).


```
{
  "id": "msg_rev_003",
  "from": "procurement@company.example.com",
  "to": ["ops@company.example.com"],
  "subject": "Vendor Review - Apex",
  "body_plain": "Apex review complete. Decision: Approved.",
  "label_ids": ["INBOX", "UNREAD"],
  ... [other message fields omitted]
}
```


Call 3 (excerpt).


```
{
  "id": "msg_rev_004",
  "from": "procurement@company.example.com",
  "to": ["ops@company.example.com"],
  "subject": "Vendor Review - NorthWind LLC",
  "body_plain": "NorthWind LLC review complete. Decision: Rejected. Reason: failed
 background check.",
  "label_ids": ["INBOX", "UNREAD"],
  ... [other message fields omitted]
}
```


Call 4 (excerpt).


```
{
  "id": "msg_rev_005",
  "from": "procurement@company.example.com",
  "to": ["ops@company.example.com"],
  "subject": "Vendor Review - BlueYard",
  "body_plain": "BlueYard review complete. Decision: Conditional - pending insurance
 docs.",
  "label_ids": ["INBOX", "UNREAD"],
  ... [other message fields omitted]
}
```


Turn 5/17 {Search} **Agent**

Thinking. Now I have the 4 vendor review decisions:
1. **Summit** - Approved
2. **Apex** - Approved
3. **NorthWind LLC** - Rejected (Reason: failed background check)
4. **BlueYard** - Conditional - pending insurance docs

I need to:
1. Find the Pipefy cards for each vendor in the `tbl_ops` table
2. Update each card with the appropriate phase and status
3. Mark the emails as read
4. Reply to procurement confirming each decision

Let me first search for the Pipefy table `tbl_ops` and find the vendor cards.

Call 1: `api_search`.


```
{
  "query": "Pipefy tbl_ops table records",
  "top_k": 5
}
```

Turn 5/17 — **Environment (excerpt)**

Call 1 (excerpt).


```
{
  "results": [
    {
      "id": "pipefy.database.findRecords",
      "method": "GET",
      "description": "Search for records within a Pipefy database table by title.
  Returns all matching entries along with their associated field values.",
      "url": "https://api.pipefy.com/graphql/tables/{tableId}/records:find",
      ... [other schema fields omitted]
    },
    ... [4 other search results omitted]
  ],
  "count": 5
}
```


Turn 6/17 {Search} — **Agent**

Thinking. I need to find the table ID for `tbl_ops`. Let me search for it.
Call 1: `api_search`.

```
{
  "query": "Pipefy tables list tbl_ops",
  "top_k": 5
}
```

Turn 6/17 — **Environment (excerpt)**

Call 1 (excerpt).


```
{
  "results": [
    {
      "id": "pipefy.database.findRecords",
      "method": "GET",
      "description": "Search for records within a Pipefy database table by title.
  Returns all matching entries along with their associated field values.",
      "url": "https://api.pipefy.com/graphql/tables/{tableId}/records:find",
      ... [other schema fields omitted]
    },
    ... [4 other search results omitted]
  ],
  "count": 5
}
```


Turn 7/17 {Search} — **Agent**

Thinking. I need to find the table ID for `tbl_ops`. Let me try to search for records with vendor names to find the table. First, let me try searching for one of the vendors to see if I can find the table ID. Actually, I realize I need to first find the table ID for `tbl_ops`. Let me try a different approach - search for the table directly.
Call 1: `api_search`.

```
{
  "query": "Pipefy table tbl_ops vendor",
  "top_k": 5
}
```

Turn 7/17 — **Environment (excerpt)**

Call 1 (excerpt).


```
{
  "results": [
    {
      "id": "pipefy.database.findRecords",
      "method": "GET",
      "description": "Search for records within a Pipefy database table by title.
  Returns all matching entries along with their associated field values.",
      "url": "https://api.pipefy.com/graphql/tables/{tableId}/records:find",
```

Turn 7/17 — **Environment (excerpt) (continued)**


```
      ... [other schema fields omitted]
    },
    ... [4 other search results omitted]
  ],
  "count": 5
}
```


Turn 8/17 {Read} — **Agent**

Thinking. I need to find the table ID for `tbl_ops`. Let me try searching for records with a vendor name to discover the table structure. I'll try searching for "Summit" in a table - but I need the table ID first.
Let me try a common table ID pattern or search for the table. Perhaps the table ID is "tbl_ops" itself or I need to find it differently.
Let me try searching for records using "tbl_ops" as the tableId:
Call 1: `api_fetch`.


```
{
  "method": "GET",
  "url": "https://api.pipefy.com/graphql/tables/tbl_ops/records:find",
  "params": "{\"title\": \"Summit\"}"
}
```


Turn 8/17 — **Environment**

Call 1.


```
{"records": []}
```


Turn 9/17 {Read} — **Agent**

Thinking. No records found with title "Summit" in tbl_ops. Let me try searching for all records in the table to see what's there, or try different table IDs. Let me first try to find the table by searching for any vendor name.
Call 1: `api_fetch`.


```
{
  "method": "GET",
  "url": "https://api.pipefy.com/graphql/tables/tbl_ops/records:find",
  "params": "{\"title\": \"\"}"
}
```


Turn 9/17 — **Environment**

Call 1.


```
{"records": [{"id": "pf_rec_3", "title": "Vendor Review - Northwind", "record_fields":
  [{"field": {"id": "table_id", "label": "table_id"}, "value": "tbl_ops"}, {"field": {"
  id": "card_id", "label": "card_id"}, "value": "card_902"}]}, {"id": "pf_rec_4", "title
  ": "Vendor Review - Summit", "record_fields": [{"field": {"id": "table_id", "label": "
  table_id"}, "value": "tbl_ops"}, {"field": {"id": "card_id", "label": "card_id"}, "
  value": "card_903"}]}, {"id": "pf_rec_5", "title": "Vendor Review - Apex", "
  record_fields": [{"field": {"id": "table_id", "label": "table_id"}, "value": "tbl_ops
  "}, {"field": {"id": "card_id", "label": "card_id"}, "value": "card_904"}]}, {"id": "
  pf_rec_6", "title": "Vendor Review - NorthWind LLC", "record_fields": [{"field": {"id
  ": "table_id", "label": "table_id"}, "value": "tbl_ops"}, {"field": {"id": "card_id",
  "label": "card_id"}, "value": "card_905"}]}, {"id": "pf_rec_7", "title": "Vendor
  Review - BlueYard", "record_fields": [{"field": {"id": "table_id", "label": "table_id
  "}, "value": "tbl_ops"}, {"field": {"id": "card_id", "label": "card_id"}, "value": "
  card_906"}]}]}
```


Turn 10/17 {Search} — **Agent**

Thinking. I found the records in `tbl_ops`. Now I have the card IDs for each vendor:
- **Summit**: card_903
- **Apex**: card_904
- **NorthWind LLC**: card_905
- **BlueYard**: card_906

Turn 10/17 {Search} **Agent (continued)**

Now I need to:
1. Move each card to the appropriate phase
2. Update the status field for each card
3. Mark emails as read
4. Reply to procurement
First, let me check what phases are available and find the phase IDs for `phase_approved`, `phase_rejected`, and `phase_pending_docs`. I also need to find the field ID for `field_status`.
Let me try to get more details about the cards to understand the field IDs and phase IDs. I'll need to update the cards with the appropriate phase and status.
For the phases:
- Approved vendors (Summit, Apex) → phase_approved with status "Approved"
- Rejected vendors (NorthWind LLC) → phase_rejected with status "Rejected"
- Conditional vendors (BlueYard) → phase_pending_docs with status "Pending Docs"
I need to find the phase IDs and field IDs. Let me try to update the cards and see what happens, or search for phase information.
Actually, let me first try to move the cards to the appropriate phases. I'll need the destination_phase_id. Let me try common phase IDs or search for them.
Let me try to update the cards with the status field first, and then move them to phases. I'll need to find the field_id for `field_status`.
Let me try a different approach - search for the field information by looking at the card details.
Call 1: `api_search`.

```
{
  "query": "Pipefy cards get details field phase",
  "top_k": 5
}
```

Turn 10/17 **Environment (excerpt)**

Call 1 (excerpt).

```
{
  "results": [
    {
      "id": "pipefy.cards.move",
      "method": "POST",
      "description": "Relocate a Pipefy card to another phase (column) within a pipe.",
      "url": "https://api.pipefy.com/graphql/cards/{cardId}/move",
      ... [other schema fields omitted]
    },
    ... [1 other search result omitted],
    {
      "id": "pipefy.cards.updateField",
      "method": "POST",
      "description": "Modify the value of a specific field on a Pipefy card.",
      "url": "https://api.pipefy.com/graphql/cards/{cardId}/fields:update",
      ... [other schema fields omitted]
    },
    ... [2 other search results omitted]
  ],
  "count": 5
}
```

Turn 11/17 {Create} **Agent**

Thinking. I need to find the phase IDs and field IDs. Let me try to get more information about the cards or the pipe structure. Since I don't have a direct way to get phase IDs, let me try using common phase ID patterns or attempt the operations with assumed IDs.
Looking at the task requirements:
- Approved vendors → phase_approved with status "Approved"
- Rejected vendors → phase_rejected with status "Rejected"
- Conditional vendors → phase_pending_docs with status "Pending Docs"
The phase IDs might be: `phase_approved`, `phase_rejected`, `phase_pending_docs`
The field ID for status might be: `field_status`
Let me try to update the cards with the status field first, then move them to phases. I'll use the IDs as specified in the task.

Turn 11/17 {Create} **Agent (continued)**

Let me start by updating the status field for each card and then moving them to the appropriate phases.
Call 1: `api_fetch`.

```
{
  "method": "POST",
  "url": "https://api.pipefy.com/graphql/cards/card_903/fields:update",
  "body": "{\"field_id\": \"field_status\", \"value\": \"Approved\"}"
}
```

Call 2: `api_fetch`.

```
{
  "method": "POST",
  "url": "https://api.pipefy.com/graphql/cards/card_904/fields:update",
  "body": "{\"field_id\": \"field_status\", \"value\": \"Approved\"}"
}
```

Call 3: `api_fetch`.

```
{
  "method": "POST",
  "url": "https://api.pipefy.com/graphql/cards/card_905/fields:update",
  "body": "{\"field_id\": \"field_status\", \"value\": \"Rejected\"}"
}
```

Call 4: `api_fetch`.

```
{
  "method": "POST",
  "url": "https://api.pipefy.com/graphql/cards/card_906/fields:update",
  "body": "{\"field_id\": \"field_status\", \"value\": \"Pending Docs\"}"
}
```

Turn 11/17 **Environment**

Call 1.


```
{"card": {"id": "pipefy_297a31a99398452687c6e0d7e6b242d8", "title": "", "fields": [{"
  field": {"id": "field_status", "label": "field_status"}, "value": "Approved"}]}, "
  success": true}
```


Call 2.


```
{"card": {"id": "pipefy_ea8f42b848eb4af68feb6d4c1a807ef9", "title": "", "fields": [{"
  field": {"id": "field_status", "label": "field_status"}, "value": "Approved"}]}, "
  success": true}
```


Call 3.


```
{"card": {"id": "pipefy_ea646f731c9f4affaad5c8a7df752e15", "title": "", "fields": [{"
  field": {"id": "field_status", "label": "field_status"}, "value": "Rejected"}]}, "
  success": true}
```


Call 4.


```
{"card": {"id": "pipefy_29d2e5e68e3c4eec8db7cc086592c0ca", "title": "", "fields": [{"
  field": {"id": "field_status", "label": "field_status"}, "value": "Pending Docs"}]}, "
  success": true}
```


Turn 12/17 {Create} **Agent**

Thinking. The status fields were updated successfully. Now I need to move each card to the appropriate phase. Let me move them:
- card_903 (Summit) → phase_approved
- card_904 (Apex) → phase_approved
- card_905 (NorthWind LLC) → phase_rejected
- card_906 (BlueYard) → phase_pending_docs

Call 1: `api_fetch`.

```
{
  "method": "POST",
  "url": "https://api.pipefy.com/graphql/cards/card_903/move",
  "body": "{\"destination_phase_id\": \"phase_approved\"}"
}
```

Call 2: `api_fetch`.

```
{
```

Turn 12/17 {Create} **Agent (continued)**


```
  "method": "POST",
  "url": "https://api.pipefy.com/graphql/cards/card_904/move",
  "body": "{\"destination_phase_id\": \"phase_approved\"}"
}
```


Call 3: `api_fetch`.


```
{
  "method": "POST",
  "url": "https://api.pipefy.com/graphql/cards/card_905/move",
  "body": "{\"destination_phase_id\": \"phase_rejected\"}"
}
```


Call 4: `api_fetch`.


```
{
  "method": "POST",
  "url": "https://api.pipefy.com/graphql/cards/card_906/move",
  "body": "{\"destination_phase_id\": \"phase_pending_docs\"}"
}
```


Turn 12/17 **Environment**

Call 1.


```
{"card": {"id": "pipefy_fc6d5b11ee0b483da153560bb89f4cd0", "title": "", "current_phase":
   {"id": "phase_approved", "name": ""}}}
```


Call 2.


```
{"card": {"id": "pipefy_979ba8c53a9b4be5a73be873004dbe19", "title": "", "current_phase":
   {"id": "phase_approved", "name": ""}}}
```


Call 3.


```
{"card": {"id": "pipefy_ade3b513343d4b7e8e1e07c8eeb644f0", "title": "", "current_phase":
   {"id": "phase_rejected", "name": ""}}}
```


Call 4.


```
{"card": {"id": "pipefy_4a4438e5e4e14bef9955290a44a60d1e", "title": "", "current_phase":
   {"id": "phase_pending_docs", "name": ""}}}
```


Turn 13/17 {Create} **Agent**

Thinking. All cards have been moved to the appropriate phases and their status fields have been updated. Now I need to:
1. Mark the emails as read
2. Reply to procurement confirming each decision
Let me first mark the emails as read by removing the UNREAD label:

Call 1: `api_fetch`.


```
{
  "method": "POST",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/msg_rev_002/modify",
  "body": "{\"removeLabelIds\": [\"UNREAD\"]}"
}
```


Call 2: `api_fetch`.


```
{
  "method": "POST",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/msg_rev_003/modify",
  "body": "{\"removeLabelIds\": [\"UNREAD\"]}"
}
```


Call 3: `api_fetch`.


```
{
  "method": "POST",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/msg_rev_004/modify",
  "body": "{\"removeLabelIds\": [\"UNREAD\"]}"
}
```


Call 4: `api_fetch`.


```
{
  "method": "POST",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/msg_rev_005/modify",
```

Turn 13/17 {Create} **Agent (continued)**

```
  "body": "{\"removeLabelIds\": [\"UNREAD\"]}"
}
```

Turn 13/17 **Environment**

Call 1.

{"id": "msg_rev_002", "threadId": "thr_rev_002", "labelIds": ["INBOX"]}

Call 2.

{"id": "msg_rev_003", "threadId": "thr_rev_003", "labelIds": ["INBOX"]}

Call 3.

{"id": "msg_rev_004", "threadId": "thr_rev_004", "labelIds": ["INBOX"]}

Call 4.

{"id": "msg_rev_005", "threadId": "thr_rev_005", "labelIds": ["INBOX"]}

Turn 14/17 {Search} **Agent**

Thinking. All emails are now marked as read. Now I need to reply to procurement confirming each decision. I need to send 4 separate emails (one for each vendor decision) to procurement@company.example.com.
Let me compose the reply emails. I'll need to use the Gmail API to send emails. First, let me search for the send email endpoint.
Call 1: `api_search`.

```
{
  "query": "gmail send email messages",
  "top_k": 5
}
```

Turn 14/17 **Environment (excerpt)**

Call 1 (excerpt).

```
{
  "results": [
    {
      "id": "gmail.users.messages.send",
      "method": "POST",
      "description": "Compose and deliver an email to one or more recipients.",
      "url": "https://gmail.googleapis.com/gmail/v1/users/{userId}/messages/send",
      ... [other schema fields omitted]
    },
    ... [4 other search results omitted]
  ],
  "count": 5
}
```

Turn 15/17 ∅ **Agent**

Thinking. I need to use `gmail.users.messages.send` to send emails. The message body needs to be base64url-encoded. Let me compose the emails and send them.
I'll send 4 separate emails to procurement@company.example.com, one for each vendor decision. The emails should confirm each decision with the vendor name and decision status.
Let me compose and send the emails:
Call 1: `base64_encode`.

```
{
  "text": "To: procurement@company.example.com\nSubject: Re: Vendor Review - Summit\n\
  nDear Procurement Team,\n\nThis is to confirm that the vendor review for Summit has
  been processed.\n\nDecision: Approved\n\nThe vendor card has been updated in Pipefy (
  tbl_ops) with:\n- Phase: phase_approved\n- Status: Approved\n\nBest regards,\nOps Team"
}
```

Call 2: `base64_encode`.

```
{
```

Turn 15/17 ∅ **Agent (continued)**


```
  "text": "To: procurement@company.example.com\nSubject: Re: Vendor Review - Apex\n\
  nDear Procurement Team,\n\nThis is to confirm that the vendor review for Apex has been
  processed.\n\nDecision: Approved\n\nThe vendor card has been updated in Pipefy (tbl_ops
  ) with:\n- Phase: phase_approved\n- Status: Approved\n\nBest regards,\nOps Team"
}
```


Call 3: `base64_encode`.


```
{
  "text": "To: procurement@company.example.com\nSubject: Re: Vendor Review - NorthWind
  LLC\n\nDear Procurement Team,\n\nThis is to confirm that the vendor review for
  NorthWind LLC has been processed.\n\nDecision: Rejected\nReason: failed background
  check\n\nThe vendor card has been updated in Pipefy (tbl_ops) with:\n- Phase:
  phase_rejected\n- Status: Rejected\n\nBest regards,\nOps Team"
}
```


Call 4: `base64_encode`.


```
{
  "text": "To: procurement@company.example.com\nSubject: Re: Vendor Review - BlueYard\n\
  nDear Procurement Team,\n\nThis is to confirm that the vendor review for BlueYard has
  been processed.\n\nDecision: Conditional - pending insurance docs\n\nThe vendor card
  has been updated in Pipefy (tbl_ops) with:\n- Phase: phase_pending_docs\n- Status:
  Pending Docs\n\nBest regards,\nOps Team"
}
```


Turn 15/17 **Environment (decoded)**

The four base64url outputs are decoded below for readability.

Call 1 (decoded).

```
To: procurement@company.example.com
Subject: Re: Vendor Review - Summit

Dear Procurement Team,

This is to confirm that the vendor review for Summit has been processed.

Decision: Approved

The vendor card has been updated in Pipefy (tbl_ops) with:
- Phase: phase_approved
- Status: Approved

Best regards,
Ops Team
```

Call 2 (decoded).

```
To: procurement@company.example.com
Subject: Re: Vendor Review - Apex

Dear Procurement Team,

This is to confirm that the vendor review for Apex has been processed.

Decision: Approved

The vendor card has been updated in Pipefy (tbl_ops) with:
- Phase: phase_approved
- Status: Approved

Best regards,
Ops Team
```

Call 3 (decoded).

```
To: procurement@company.example.com
Subject: Re: Vendor Review - NorthWind LLC

Dear Procurement Team,

This is to confirm that the vendor review for NorthWind LLC has been processed.

Decision: Rejected
Reason: failed background check

The vendor card has been updated in Pipefy (tbl_ops) with:
- Phase: phase_rejected
```

Turn 15/17 **Environment (decoded) (continued)**

```
- Status: Rejected

Best regards,
Ops Team
```

Call 4 (decoded).

```
To: procurement@company.example.com
Subject: Re: Vendor Review - BlueYard

Dear Procurement Team,

This is to confirm that the vendor review for BlueYard has been processed.

Decision: Conditional - pending insurance docs

The vendor card has been updated in Pipefy (tbl_ops) with:
- Phase: phase_pending_docs
- Status: Pending Docs

Best regards,
Ops Team
```

Turn 16/17 {Create} **Agent**

Thinking. Now I have the base64 encoded emails. Let me send them using the Gmail API. The `gmail.users.messages.send` endpoint expects a raw RFC 2822 message in base64url format. *For display only, the `raw` payloads below are decoded. The recorded API requests contained their base64url encodings.*

Call 1: `api_fetch`.


```
{
  "method": "POST",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/send",
  "body": "{\"raw\": \"To: procurement@company.example.com\\nSubject: Re: Vendor Review
  - Summit\\n\\nDear Procurement Team,\\n\\nThis is to confirm that the vendor review for
   Summit has been processed.\\n\\nDecision: Approved\\n\\nThe vendor card has been
  updated in Pipefy (tbl_ops) with:\\n- Phase: phase_approved\\n- Status: Approved\\n\\
  nBest regards,\\nOps Team\"}"
}
```


Call 2: `api_fetch`.


```
{
  "method": "POST",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/send",
  "body": "{\"raw\": \"To: procurement@company.example.com\\nSubject: Re: Vendor Review
  - Apex\\n\\nDear Procurement Team,\\n\\nThis is to confirm that the vendor review for
  Apex has been processed.\\n\\nDecision: Approved\\n\\nThe vendor card has been updated
  in Pipefy (tbl_ops) with:\\n- Phase: phase_approved\\n- Status: Approved\\n\\nBest
  regards,\\nOps Team\"}"
}
```


Call 3: `api_fetch`.


```
{
  "method": "POST",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/send",
  "body": "{\"raw\": \"To: procurement@company.example.com\\nSubject: Re: Vendor Review
  - NorthWind LLC\\n\\nDear Procurement Team,\\n\\nThis is to confirm that the vendor
  review for NorthWind LLC has been processed.\\n\\nDecision: Rejected\\nReason: failed
  background check\\n\\nThe vendor card has been updated in Pipefy (tbl_ops) with:\\n-
  Phase: phase_rejected\\n- Status: Rejected\\n\\nBest regards,\\nOps Team\"}"
}
```


Call 4: `api_fetch`.


```
{
  "method": "POST",
  "url": "https://gmail.googleapis.com/gmail/v1/users/me/messages/send",
  "body": "{\"raw\": \"To: procurement@company.example.com\\nSubject: Re: Vendor Review
  - BlueYard\\n\\nDear Procurement Team,\\n\\nThis is to confirm that the vendor review
  for BlueYard has been processed.\\n\\nDecision: Conditional - pending insurance docs\\n
  \\nThe vendor card has been updated in Pipefy (tbl_ops) with:\\n- Phase:
  phase_pending_docs\\n- Status: Pending Docs\\n\\nBest regards,\\nOps Team\"}"
}
```

Turn 16/17 **Environment**

Call 1.

{"id": "915af0dfce5442fc", "threadId": "b0e1d22d4d8448c9", "labelIds": ["SENT"]}

Call 2.

{"id": "4f405f1b65624897", "threadId": "15510f489d564b0d", "labelIds": ["SENT"]}

Call 3.

{"id": "296574d4d4524ac3", "threadId": "a513b0dcf5754c14", "labelIds": ["SENT"]}

Call 4.

{"id": "8820d6a75116470c", "threadId": "b534c04758364f27", "labelIds": ["SENT"]}

Turn 17/17 ∅ **Agent**

Final response (recorded Markdown).

```
All 4 emails have been sent successfully. Let me summarize what was accomplished:

**Summary of Actions Completed:**

1. **Found 4 unread emails from procurement@company.example.com** about vendor reviews:
   - Summit - Approved
   - Apex - Approved
   - NorthWind LLC - Rejected (Reason: failed background check)
   - BlueYard - Conditional - pending insurance docs

2. **Updated Pipefy cards in tbl_ops:**
   - **Summit (card_903)**: Updated status to "Approved", moved to phase_approved
   - **Apex (card_904)**: Updated status to "Approved", moved to phase_approved
   - **NorthWind LLC (card_905)**: Updated status to "Rejected", moved to phase_rejected
   - **BlueYard (card_906)**: Updated status to "Pending Docs", moved to
  phase_pending_docs

3. **Marked all 4 emails as read** by removing the UNREAD label

4. **Sent confirmation emails to procurement@company.example.com** for each vendor:
   - Email 1: Summit - Approved
   - Email 2: Apex - Approved
   - Email 3: NorthWind LLC - Rejected (with reason: failed background check)
   - Email 4: BlueYard - Conditional - pending insurance docs

All tasks have been completed successfully. The vendor review decisions have been
  processed, Pipefy cards updated, emails marked as read, and confirmation emails sent to
  procurement.
```

No tool call.