PROPOSE-AND-REFINE: MULTI-TURN LLM ROUTING FOR LONG-HORIZON TASKS WITH VIRTUAL-ROLLOUT MCTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are increasingly deployed for long-horizon tasks, where inference costs accumulate over multiple turns and different turns may require different model capabilities. Unlike episode-level routing, multi-turn routing selects a model at each interaction step to better balance performance and cost under user preferences. However, learning effective multi-turn routers remains challenging because the long-term value of a routing action depends on future routing plans, and high-quality supervision is difficult to obtain under a limited real-environment rollout budget. We propose ProRe, a propose-and-refine multi-turn routing framework. At each state, a generative proposer samples routing plans composed of variable-duration macro-actions, and a plan-conditioned refiner ranks them by predicted terminal performance and cumulative cost under a user-specified preference, thereby incorporating future information into the current decision. To train ProRe efficiently, we introduce OnVir, an online data-collection strategy that replaces real-environment rollouts in Monte Carlo tree search (MCTS) with ProRe-generated virtual rollouts, discovering higher-quality routing trajectories under a fixed sampling budget. Extensive experiments show that ProRe and OnVir jointly improve routing performance across diverse long-horizon tasks and user preferences.

1 INTRODUCTION

Large language models (LLMs) are increasingly deployed in consumer and enterprise applications (Appel et al., 2025). However, their capabilities and inference costs vary substantially across tasks and queries due to differences in model scale and reasoning effort (Hu et al., 2024; Huang et al., 2026). Users may also hold heterogeneous preferences over the trade-off between performance and cost (Chen et al., 2023; Feng et al., 2025). This challenge is particularly salient for long-horizon tasks such as software engineering (Jimenez et al., 2024) and deep research (Phan et al., 2025), which require multi-step interactions and accumulate inference costs over time. Always invoking the strongest model can be prohibitively expensive, whereas relying exclusively on cheaper models may compromise performance. This trade-off motivates LLM routing, which selects models from a candidate pool according to user preferences (Zhang et al., 2026b; Ma et al., 2026).

Existing LLM routing methods primarily focus on episode-level routing, choosing one model from the initial query for the entire episode. They either directly predict a suitable model (Chen et al., 2024b; Zhang et al., 2025c) or maximize a preference-aware utility based on estimated performance and cost (Liu et al., 2026). Such static routing is inherently suboptimal for long-horizon tasks because different turns may require different model capabilities. Multi-turn routing instead adaptively selects a model at each interaction step (Zhang et al., 2026a). Existing approaches typically formulate this process as a Markov decision process (MDP), where each action selects a model at a given interaction step. They collect training trajectories using behavior policies such as Monte Carlo tree search (MCTS) (Browne et al., 2012) or heuristic strategies including single-model execution and random switching, and then train the router using imitation learning (Ross et al., 2011), offline Q-learning (Clifton & Laber, 2020), or outcome regression (Zhang et al., 2026b).

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

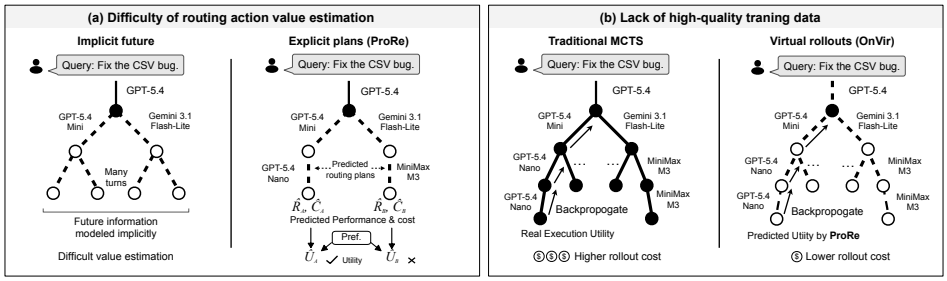

Figure 1: Challenges in multi-turn LLM routing. (a) Predicting long-horizon action values is difficult because existing routers consider future information only implicitly, whereas ProRe explicitly proposes complete future routing plans and evaluates them using predicted performance, cost, and user preference. (b) Traditional MCTS relies on costly real-environment rollouts to evaluate candidate branches, whereas OnVir replaces them with virtual rollouts, enabling cheaper exploration.

Nevertheless, as shown in Fig. 1, learning effective multi-turn routers poses two key challenges. First, estimating long-horizon action values is difficult because the value depends on both the current state and subsequent routing plans, complicating the performance–cost trade-off. Existing multi-turn routers capture this dependence only implicitly and do not explicitly consider long-horizon information when selecting the current action (Hafner et al., 2019; Ding et al., 2025). Second, acquiring high-quality supervision under a limited sampling budget is difficult. While search-based behavior policies such as MCTS can outperform heuristics, evaluating candidate branches requires costly rollouts to task completion. Because the search space grows exponentially with the task horizon, fixed-budget search offers limited coverage and may yield suboptimal training trajectories. This behavior–expert gap causes imitation learning and outcome regression to inherit the suboptimality of logged trajectories (Rajaraman et al., 2020), while offline Q-learning is additionally susceptible to out-of-distribution value-estimation errors that compound through bootstrapping (Qu et al., 2025).

To this end, we propose **Pro**pose-and-**Re**fine (ProRe), a multi-turn routing framework comprising a generative proposer and a plan-conditioned refiner. Given the current interaction state and a user-specified cost–performance preference, the proposer samples multiple complete routing plans. Inspired by action chunking (Zhao et al., 2023), each plan consists of variable-duration macro-actions that specify a model and the number of consecutive execution turns, reducing the frequency of routing decisions. The refiner ranks these plans by predicting terminal performance and cumulative inference cost, then combines them under the specified preference. ProRe executes the first macro-action of the highest-ranked plan and replans from the resulting state until task completion or budget exhaustion. This planning process thus explicitly incorporates long-horizon information by comparing possible future routing plans before acting. To collect high-quality training trajectories, we introduce OnVir, an online data-collection strategy that augments MCTS with virtual rollouts. OnVir bootstraps ProRe from heuristic single-model trajectories and then performs multi-round collection. In each round, MCTS selects and expands routing actions, after which the proposer generates a completion from the resulting leaf, and the refiner predicts its utility for backpropagation. This virtual rollout avoids costly real-environment rollouts while updating node statistics. After S_V simulations, OnVir samples the root macro-action from virtual visit counts and executes it in the real environment. The resulting real trajectories are added to a cumulative dataset for improving ProRe at the end of each round. By integrating MCTS with virtual rollouts generated and evaluated by the evolving ProRe, OnVir can discover higher-quality routing trajectories under a fixed environment-interaction budget. Extensive experiments across diverse tasks and user preferences show that ProRe and OnVir jointly improve routing performance and achieve favorable performance–cost trade-offs.

Our contributions are threefold.

- We introduce ProRe, a multi-turn routing framework that proposes and refines routing plans, enabling each routing action to be evaluated under explicit future information.
- We develop OnVir, an online data-collection strategy that augments MCTS with virtual rollouts generated and evaluated by an evolving ProRe, improving training-trajectory quality under a fixed environment-interaction budget.
- We conduct extensive experiments across diverse tasks and preferences, showing ProRe and OnVir jointly improve routing performance and produce higher-quality training data.

2 Related Work

Episode-level LLM Routing. Episode-level (single-turn) routing selects one LLM from a candidate pool at the start of an episode and keeps it fixed, conditioned on user preferences. Early work primarily maximizes task performance (Jiang et al., 2023; Zhan et al., 2026), whereas recent work increasingly emphasizes the trade-off between performance and inference cost as model capabilities diversify and costs rise (Zhang et al., 2023; Ding et al., 2024). Existing approaches generally fall into two categories. The first directly predicts a suitable model using a fine-tuned lightweight encoder such as BERT (Chen et al., 2024b; Srivatsa et al., 2024; Feng et al., 2025) or a fine-tuned LLM (Zhang et al., 2025a;c), or employs a model cascade (Chen et al., 2023; Dai et al., 2024). The second trains a query-specific performance and cost estimator for each model and selects the model that maximizes a user-specified utility (Mei et al., 2025; Liu et al., 2026; Cheng et al., 2024), enabling finer control over user preferences. ProRe integrates these paradigms through a two-stage design in which a proposer first generates multiple plausible routing plans to narrow the search space, and the refiner's performance and cost estimators then re-rank the candidates.

Multi-turn LLM Routing. LLMs are increasingly used for long-horizon tasks such as software engineering (Jimenez et al., 2024) and deep research (Phan et al., 2025), which involve multi-step interactions and accumulate inference costs over turns. Because different interaction steps may require different model capabilities, episode-level routing can be suboptimal. Multi-turn routing, therefore, adaptively selects a model at each interaction step (Zhan et al., 2026). Existing methods typically formulate routing as a Markov decision process (MDP), where each action selects a model at each step. They differ primarily in their behavior policies for trajectory collection and their learning objectives. For example, MTRouter (Zhang et al., 2026b) collects trajectories using heuristic strategies, including single-model execution and random switching, and trains a state–action outcome regressor using terminal utility as supervision. DialRouter (Zhang et al., 2026a) uses MCTS to search for high-utility trajectories and trains the router via imitation learning, whereas SeqRouter (Xu et al., 2026) adopts offline Q-learning. Despite their effectiveness, these methods still face challenges in estimating long-horizon action values and obtaining high-quality training trajectories under limited environment rollout budgets.

3 Method

As illustrated in Fig. 2, we propose Propose-and-Refine (ProRe), a routing framework with a generative plan proposer and a plan-conditioned refiner, and OnVir, an online data-collection strategy that augments MCTS with virtual rollouts to collect high-quality training trajectories. The remainder of this section is organized as follows: Sec. 3.1 formulates multi-turn routing, while Secs. 3.2 and 3.3 describe ProRe and OnVir, respectively.

3.1 Problem Formulation

We consider long-horizon tasks in which an agent interacts with an environment for at most L turns. Let $\mathcal{M} = \{m_1, \ldots, m_K\}$ denote a pool of LLMs with heterogeneous capabilities and inference costs. We index routing steps by t and environment turns by ℓ. At routing step t, the router observes $s_t = (x, h_t, \rho_t)$, where x is the task, h_t is the interaction history, and $\rho_t = (a_0, \ldots, a_{t-1})$ is the executed routing history. Following ACT (Zhao et al., 2023), we represent each routing decision as a variable-duration macro-action $a_t = (m_t, n_t)$, where $m_t \in \mathcal{M}$ and $n_t \in \{1, \ldots, N\}$, which reduces routing frequency while retaining flexible model switching. Let $\mathcal{A}$ denote the action space, with $|\mathcal{A}| = KN$. The selected model m_t is invoked for up to n_t consecutive environment turns, subject to task termination and the remaining turn budget. For simplicity, n_t also denotes the realized duration after truncation. Executing a_t induces the next routing state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$. A future routing plan from s_t is denoted by $\rho_t^+ = (a_t, \ldots, a_{T_t-1})$, containing $T_t - t$ macro-actions. Let $\lambda = (\lambda_p, \lambda_c)$ denote the user's performance–cost preference, with $\lambda_p, \lambda_c \geq 0$ and $\lambda_p + \lambda_c = 1$. For a completed trajectory τ with T_τ macro-actions and $L_\tau = \sum_{t=0}^{T_\tau - 1} n_t \leq L$ turns, we define

$$U_r(\tau; \lambda) = \lambda_p \, \mathrm{Norm}_R(R(\tau)) - \lambda_c \, \mathrm{Norm}_C(C_r(\tau)) \,, \tag{1}$$

where $R(\tau)$ is terminal task performance and $C_r(\tau) = \sum_{\ell=0}^{L_\tau - 1} c_{\ell,r}$ is total inference cost. The benchmark-specific operators Norm_R and Norm_C apply min–max normalization to performance

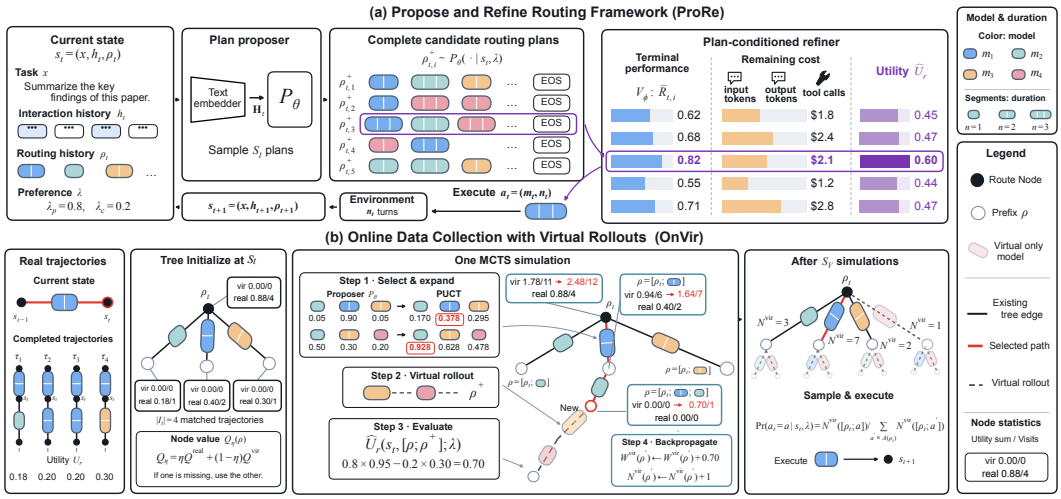

Figure 2: Overview of ProRe and OnVir. (a) At state s_t, the proposer samples S_I complete routing plans. The refiner predicts terminal performance and remaining input, output, and tool costs. ProRe then executes the first macro-action of the highest-utility plan. (b) At each state, OnVir initializes a prefix tree from completed trajectories, runs MCTS with selection, expansion, and virtual rollouts. The proposer generates a completion for the leaf, the refiner evaluates it, and the predicted utility is backpropagated. The next macro-action is sampled from virtual visit counts.

and log-transformed cost, respectively. The per-turn cost $c_{\ell,r}$ includes input- and output-token charges and, when applicable, tool-call charges; the prompt-cache hit rate r determines the effective input-token price. Our objective is to learn a preference-conditioned routing policy $\pi_\theta(a_t \mid s_t, \lambda)$ that maximizes expected utility, $\pi_\theta^\star \in \arg\max_{\pi_\theta} \mathbb{E}_{\tau\sim p_{\pi_\theta}(\tau|\lambda)}[U_r(\tau;\lambda)]$.

3.2 Propose-and-Refine Routing Framework

Estimating an action's long-horizon value requires accounting for subsequent routing plans ρ_t^+ beyond the current state s_t. ProRe instead generates complete routing plans and evaluates them by separately predicting terminal performance and inference cost. Full architectural details of these components are deferred to Appendix A.1.

We encode the task and interaction history with a text embedder, truncating the interaction history to at most L_h turns. The task and retained history are encoded independently and projected into a shared d-dimensional space, yielding a task token $\mathbf{x} \in \mathbb{R}^d$ and history tokens $\mathbf{h}_t \in \mathbb{R}^{L_h \times d}$. If fewer than L_h turns are observed, we zero-pad the history to length L_h. Let $\mathcal{V}_m = \mathcal{M} \cup \{\text{EOS}\}$ and $\mathcal{V}_n = \{1, \ldots, N\}$ denote the model and duration vocabularies, where EOS marks the end of a plan. Each macro-action is $a_j = (m_j, n_j)$ for routing step j, with $m_j \in \mathcal{M}$ and $n_j \in \mathcal{V}_n$. For an executed macro-action a_j with $j < t$, we represent it by the sum of learned model, duration, and positional embeddings, plus a projection of observed usage:

$$\mathbf{a}_j = \mathbf{m}_{m_j} + \mathbf{n}_{n_j} + \mathbf{e}_j^{\text{pos}} + g(\log(1 + \mathbf{u}_j)), \tag{2}$$

where $\mathbf{m}_{m_j}$ and $\mathbf{n}_{n_j}$ are learned embeddings for model m_j and duration n_j from $\mathcal{V}_m$ and $\mathcal{V}_n$, $\mathbf{e}_j^{\text{pos}}$ is a positional embedding, and $\mathbf{u}_j$ contains input- and output-token counts and, when applicable, billed tool-call counts (e.g., web search in our work). For future macro-actions, the usage term is omitted because usage is unobserved. Embedding the executed routing history yields $\boldsymbol{\rho}_t = (\mathbf{a}_0, \ldots, \mathbf{a}_{t-1})$. We further project the preference λ into a learned token $\mathbf{e}_\lambda \in \mathbb{R}^d$ and concatenate it with the task, interaction-history, and routing-history representations to obtain conditional information,$\mathbf{H}_t = [\mathbf{e}_\lambda; \mathbf{x}; \mathbf{h}_t; \boldsymbol{\rho}_t]$ for the subsequent proposer and refiner. A learned BOS token initializes autoregressive decoding but is not predicted.

Generative plan proposer. The proposer models an autoregressive distribution $P_\theta(\rho_t^+ \mid s_t, \lambda, \rho_t)$ over future routing plans. It uses a Transformer decoder with causal self-attention over the generated prefix and cross-attention to $\mathbf{H}_t$ to obtain the decoder state $\mathbf{z}_j$ at step j. Two softmax heads then predict the model and duration of each macro-action:

$$P_\theta(a_j = (m, n) \mid s_t, \lambda, a_{t:j-1}) = P_\theta^m(m \mid \mathbf{z}_j) P_\theta^n(n \mid \mathbf{z}_j). \tag{3}$$

The termination probability is $P_\theta^m(\text{EOS} \mid \mathbf{z}_j)$, and the duration prediction is ignored when EOS is selected. The probability of a complete plan factorizes as

$$P_\theta(\rho_t^+ \mid s_t, \lambda, \rho_t) = \prod_{j=t}^{T_t-1} P_\theta(a_j \mid s_t, \lambda, a_{t:j-1}) \times P_\theta(\text{EOS} \mid s_t, \lambda, a_{t:T_t-1}), \tag{4}$$

where decoding terminates upon EOS or turn budget is exhausted; in the latter case, the terminal EOS factor is set to one. Sampling from this distribution yields S_I candidate plans, $\mathcal{P}_t = \{\rho_{t,i}^+\}_{i=1}^{S_I}$.

Plan-conditioned refiner. For a candidate future routing plan $\rho_{t,i}^+$ for ρ_t at routing state s_t, the refiner scores it with a performance predictor V_ϕ and a cost predictor C_ψ. Let $\boldsymbol{\rho}_{t,i}^+$ denote the embeddings of the candidate plan $\rho_{t,i}^+$ by applying Eq. 2 without the usage term. For performance prediction, V_ϕ introduces a learned CLS token and incorporates future-plan information $\boldsymbol{\rho}_{t,i}^+$ and conditioning information $\mathbf{H}_t$ into it through a self-attention Transformer encoder that takes all of them as input. The resulting CLS representation $\mathbf{v}_{t,i}$ is then processed by a two-layer MLP with GELU activation to predict a normalized terminal performance $\hat{R}_{t,i}$, clipped to $[0, 1]$ during ranking.

For cost prediction, C_ψ first expands each macro-action (m, n) into n unit-duration actions $(m, 1)$, yielding an expanded plan $\widehat{\rho}_{t,i}^+$ of length $L_{t,i} = \sum_{j=t}^{T_{t,i}-1} n_j$, where $T_{t,i}$ is the number of macro-actions in $\rho_{t,i}^+$. It then employs a self-attention Transformer encoder over the conditioning information $\mathbf{H}_t$ and the expanded plan embeddings $\widehat{\boldsymbol{\rho}}_{t,i}^+$ to produce per-turn hidden states $\mathbf{C}_{t,i}$, where $\mathbf{C}_{t,i}[\ell]$ denotes the representation of future turn ℓ and $\tilde{m}_{t,i,\ell}$ is the model assigned to that turn. For each turn, an MLP predicts log-transformed usage for input tokens, output tokens, and optional tool calls, yielding the nonnegative estimates $\hat{\mathbf{u}}_{t,i,\ell} = (\hat{u}^{\text{in}}_{t,i,\ell}, \hat{u}^{\text{out}}_{t,i,\ell}, \hat{u}^{\text{tool}}_{t,i,\ell}) = \max\{\exp(\text{MLP}_C(\mathbf{C}_{t,i}[\ell])) - 1, 0\}$. This per-turn, token-count-based design has the potential to allow the refiner to evaluate under different cache-hit rates r. The predicted remaining cost is $\widehat{C}_r(s_t, \rho_{t,i}^+) = \sum_{\ell=0}^{L_{t,i}-1} \left(\hat{u}^{\text{in}}_{t,i,\ell} p^{\text{in}}_{\tilde{m}_{t,i,\ell}}(r) + \hat{u}^{\text{out}}_{t,i,\ell} p^{\text{out}}_{\tilde{m}_{t,i,\ell}} + \hat{u}^{\text{tool}}_{t,i,\ell} p^{\text{tool}} \right)$, where $p^{\text{in}}_m(r)$, p^{out}_m, and p^{tool} denote the effective input-token, output-token (model m), and per-call tool prices. We then combine predicted performance with incurred and predicted future costs by $\widehat{U}_r(s_t, \rho_{t,i}^+; \lambda) = \lambda_p \hat{R}_{t,i} - \lambda_c \,\text{Norm}_C\Big(C_r(\rho_t) + \widehat{C}_r(s_t, \rho_{t,i}^+)\Big)$, where $C_r(\rho_t)$ is the cost incurred by the executed routing history under cache-hit rate r. ProRe selects the highest-utility plan $\rho_t^{+,\star} = \arg\max_{\rho_{t,i}^+ \in \mathcal{P}_t} \widehat{U}_r(s_t, \rho_{t,i}^+; \lambda)$, executes its first macro-action, and replans after observing the next state. Repeating this process until termination obtains a complete routing trajectory.

3.3 Online Data Collection with Virtual Rollouts

Collecting high-utility routing trajectories is challenging because the plan space grows exponentially with the plan length T_t. MCTS improves exploration but requires costly environment rollouts. OnVir instead replaces environment rollouts with virtual ones: at a leaf, the proposer generates a plausible completion ρ_t^+ of the current plan ρ_t, and the refiner predicts its utility for backpropagation.

Online Collection. We collect data over Q rounds. For each preference λ, we bootstrap ProRe with single-model trajectories and denote the resulting dataset by $\mathcal{D}^{(0)} = \mathcal{D}_0$. At round $q \in \{1, \ldots, Q\}$, we train P_θ^q, V_ϕ^q, and C_ψ^q on $\mathcal{D}^{(q-1)}$ and freeze them during collection. We then collect B trajectories per task into D_q, and each completed one is added to D_q immediately, so later collections in the same round can reuse earlier outcomes, detailed in Appendix. B.4. After collection, we improve ProRe on all data collected up to round q, denoted as $\mathcal{D}^{(q)} = \mathcal{D}^{(q-1)} \cup D_q$. To collect a trajectory for task x at round q, at each routing state, OnVir builds a routing-plan prefix tree and selects a macro-action via MCTS with virtual rollouts. Next, we show how these two steps are used to obtain the complete trajectory using s_t as an example and omit the superscript q for brevity.

Tree Initialization. At state s_t, OnVir builds a routing-plan prefix tree by taking the executed routing history ρ_t as the root node. We identify each node by its routing prefix ρ. Let ρ_t^k denote a node at depth k, formed by appending k future actions to ρ_t, i.e., $\rho_t^k = [\rho_t; a_t, \ldots, a_{t+k-1}]$. All nodes share s_t and differ only in their future routing actions. Each node ρ maintains real statistics $(N^{\text{real}}(\rho), W^{\text{real}}(\rho))$ and virtual statistics $(N^{\text{vir}}(\rho), W^{\text{vir}}(\rho))$, where N denotes the visit count and

W the cumulative utility. To initialize real statistics, we retrieve completed trajectories with the same task and preference (x, λ) from $\mathcal{D}^{(q-1)} \cup D_q$ that visited s_t. Let $\mathcal{I}_t$ be the set of records (τ, a) such that trajectory τ executed action a at s_t. For the root ρ_t, $N^{\text{real}}(\rho_t) = |\mathcal{I}_t|$ is the number of matched trajectories, and $W^{\text{real}}(\rho_t) = \sum_{(\tau,a)\in\mathcal{I}_t} U_r(\tau;\lambda)$ is their cumulative utility. For a first-level child $\rho_t^1 = [\rho_t; a]$, $N^{\text{real}}(\rho_t^1) = |\{(\tau, a') \in \mathcal{I}_t \mid a' = a\}|$ is the number of matched trajectories that executed action a at s_t, and $W^{\text{real}}(\rho_t^1) = \sum_{(\tau,a')\in\mathcal{I}_t,\, a'=a} U_r(\tau;\lambda)$ is their cumulative utility. Deeper nodes receive no real initialization, as the retrieved records provide evidence only for the action executed at s_t. All virtual statistics are initialized to zero.

MCTS with Virtual Rollouts. Given the initialized tree, OnVir runs S_V MCTS simulations. Each simulation consists of selection, expansion, a virtual rollout, and backpropagation. Following standard MCTS, Q denotes the mean value of a node. We define $Q^{\text{real}}(\rho) = W^{\text{real}}(\rho)/N^{\text{real}}(\rho)$ and $Q^{\text{vir}}(\rho) = W^{\text{vir}}(\rho)/N^{\text{vir}}(\rho)$, and combine them as $Q_\eta(\rho) = \eta Q^{\text{real}}(\rho) + (1-\eta)Q^{\text{vir}}(\rho)$ when both exist, otherwise using the available one. The total visit count is $N(\rho) = N^{\text{real}}(\rho) + N^{\text{vir}}(\rho)$.

During the selection stage, starting from the root ρ_t, the proposer supplies a prior $p_\rho(a) \propto P_\theta(a \mid s_t, \lambda, \rho)$, which is the next-action distribution conditioned on s_t, λ, and the already decoded prefix ρ, consistent with Eq. 4. This prior is normalized over executable macro-actions $\mathcal{A}(\rho) \subseteq \mathcal{A}$ that do not exceed the remaining turn budget. At each nonterminal node ρ, PUCT selects

$$a^* = \arg\max_{a\in\mathcal{A}(\rho)} \left[\widetilde{Q}_\eta(\rho, a) + c_{\text{puct}} p_\rho(a) \frac{\sqrt{1+N(\rho)}}{1+N([\rho;a])} \right], \tag{5}$$

where c_{puct} controls exploration and $\widetilde{Q}_\eta(\rho, a) = Q_\eta([\rho; a])$ for an existing child in the tree, while $\widetilde{Q}_\eta(\rho, a) = Q_\eta(\rho)$ for an unexpanded action with zero visits. Selection stops at an unexpanded action or when the turn budget is exhausted. In the former case, we create a child $[\rho; a]$ and initialize its statistics to zero, while in the latter, no expansion is performed.

From the resulting leaf ρ, the proposer samples a completion $\rho^+ \sim P_\theta(\cdot \mid s_t, \lambda, \rho)$ until EOS or the remaining-budget boundary. The refiner then predicts the utility $\widehat{U}_r(s_t, [\rho; \rho^+]; \lambda)$, which is backpropagated to every node ρ' on the selected path $\mathcal{P}$: $N^{\text{vir}}(\rho') \leftarrow N^{\text{vir}}(\rho') + 1$ and $W^{\text{vir}}(\rho') \leftarrow W^{\text{vir}}(\rho') + \widehat{U}_r(s_t, [\rho; \rho^+]; \lambda)$. This virtual rollout requires no candidate-model invocations, tool execution, or simulated dialogue transitions. The sampled completion is used only for leaf evaluation and is not inserted into the tree, and real statistics remain unchanged during virtual search.

After S_V simulations, OnVir samples the macro-action a_t under s_t as

$$\Pr(a_t = a \mid s_t, \lambda) = N^{\text{vir}}([\rho_t; a]) / \sum_{a'\in\mathcal{A}(\rho_t)} N^{\text{vir}}([\rho_t; a']), \tag{6}$$

where only current virtual visit counts are used, following Silver et al. (2016). If a completed trajectory in D_q already records the same state–action pair (s_t, a_t), we reuse its logged model outputs instead of invoking the model again. If the episode continues, search restarts at s_{t+1} from an empty-prefix root, and previous virtual rollouts are discarded while only real statistics supported by completed trajectories are reused. Once the episode terminates or exhausts the turn budget, the trajectory is added to D_q. Algorithm 1 and 2 in the Appendix summarize all procedures above.

For final evaluation, we train all components of ProRe on the full collected dataset $\mathcal{D}^{(Q)} = \mathcal{D}_0 \cup \bigcup_{i=1}^{Q} D_i$, where Q is the number of online rounds. At inference, for each routing state s_t, ProRe samples S_I future plans from the proposer and re-ranks them using the refiner, as described in Sec. 3.2. The detailed training procedure is provided in Appendix A.2 due to page limitations.

4 Experiments

4.1 Experimental Setting

Benchmarks and LLM Pool. Following Zhang et al. (2026b), we evaluate on three long-horizon benchmarks. ScienceWorld (SW) is a text-based environment where an agent executes procedural experiments via multi-step commands. Humanity's Last Exam (HLE) consists of challenging cross-domain questions that require multi-step reasoning and tool use, and agents may invoke

a paid web search tool. SWE-BENCH VERIFIED (SWE-V) requires resolving real GitHub issues within repository sandboxes. We set the maximum number of environment turns L to 50 for SW and SWE-V, and 30 for HLE. We split each benchmark into training, validation, and test sets. The test set contains in-domain (ID) and out-of-domain (OOD) splits, where OOD domains are disjoint from training and ID test domains. We construct the LLM pool $\mathcal{M}$ from seven widely used models spanning diverse capabilities and cost tiers, including DeepSeek V4 Flash, DeepSeek V4 Pro, GPT-5.4, GPT-5.4 Mini, GPT-5.4 Nano, Gemini 3.1 Flash-Lite, and MiniMax M3.

Baselines. We compare our method with heuristic methods, episode-level routers, and multi-turn routers. Heuristic methods include the cheapest, strongest, and adaptive single-model policies, each of which selects one LLM for the entire episode based on training-set results under cost, performance, or preference-adaptive criteria. Episode-level routers, including ColdStartRouter (Cold) (Liu et al., 2026), GraphRouter (Feng et al., 2025), IRT-Router (Song et al., 2025), and RouterDC (Chen et al., 2024b), select one model from the initial query and keep it fixed. For multi-turn routing, MTRouter (Zhang et al., 2026b) trains an outcome regressor and does not model user preferences. We further adapt it to MTPref, which uses the same architecture but regresses preference-conditioned utility. SeqRouter (Xu et al., 2026) is trained with offline Q-learning, and DialRouter (Zhang et al., 2026a) with imitation learning.

Implementation Details. All candidate LLMs are decoded at temperature 0. Following MTRouter, we set the prompt-cache hit rate to $r = 0$ unless otherwise specified. Following existing work (Feng et al., 2025; Liu et al., 2026), we consider three preference settings for $\lambda = (\lambda_p, \lambda_c)$, namely cost $(0.2, 0.8)$, balance $(0.5, 0.5)$, and performance $(0.8, 0.2)$. We use a frozen Qwen3-Embedding-0.6B encoder (Zhang et al., 2025b) as the text embedder. The maximum macro-action duration is $N = L/10$. For ProRe, the proposer samples $S_I = 8$ complete plans and retains the most recent $L_h = 8$ interaction turns. For OnVir, we use $Q = 3$ rounds with $B = 4$ real trajectories per instance per round, and $S_V = 128$ virtual rollouts with $c_{\text{puct}} = 1.5$ and $\eta = 0.5$ in fixed mode. We repeat the main experiments and primary ablations twice and report the mean. Episode-level and multi-turn baselines use the same training data as MTRouter, in which each of the seven LLMs is run for two full episodes per training instance, along with five additional random-switch trajectories that uniformly resample the model at each turn.

Additional details on the benchmarks, baselines, and implementation are provided in Appendix B.3.

4.2 MAIN RESULTS

Effectiveness of Our Method. We evaluate ProRe, trained on OnVir-collected trajectories, against heuristic, episode-level, and multi-turn routers trained on MTRouter data. Table 1 presents the comparison, and the results reveal three findings. (1) ProRe achieves the highest utility across all benchmarks and preference settings, exceeding the best competing method by 0.20, 0.18, and 0.11 in Overall on SW, HLE, and SWE-V, respectively. It also responds smoothly to preferences. As the performance weight increases from 0.2 to 0.8, both cost and performance increase on the in-domain split (e.g., on SW, the cost–performance pair shifts from $(0.31, 0.63)$ to $(0.36, 0.71)$), whereas several episode-level routers remain close to a fixed single-model policy. (2) Multi-turn routing alone does not necessarily outperform heuristic or episode-level routing. On SWE-V, the best multi-turn router, MTPref, achieves an Overall of 0.04, below the Adaptive router's 0.12. This suggests that implicit future modeling via imitation learning or Q-learning may be insufficient for long-horizon routing, especially with Q-learning, where bootstrapping can propagate value-estimation errors. (3) ProRe also generalizes better to out-of-domain tasks and remains best in each benchmark in OOD settings, though a gap between in-domain and out-of-domain performance persists. It highlights cross-task-family generalization as an open challenge for multi-turn routing.

Effectiveness of OnVir. We assess whether OnVir collects higher-quality training data through two experiments. First, we compare the best trajectory utility obtained by OnVir, ProRe, traditional MCTS, and the MTRouter strategy under comparable interaction budgets. All strategies start from the same seven-trajectory bootstrap set $\mathcal{D}_0$ in each benchmark. Here ProRe collects the same twelve real trajectories as OnVir but omits the search tree, drawing each trajectory independently from the proposer. Fig. 3 shows that OnVir achieves the highest mean best-trajectory utility on both SW and SWE-V with substantially fewer real trajectories. This indicates that virtual rollouts provide more informative search signals, whereas ProRe collapses toward a static utility due to the lack of

Table 1: Normalized cost, performance, and utility on SW, HLE, and SWE-V. We report results across three preference settings: cost, balance, and performance. ID Sum and OOD Sum denote the utility sums across the three preference settings in the in-domain and out-of-domain evaluations, respectively, and Overall is their average. For each dataset, the best and second-best values of U and sums are highlighted in **bold** and underline, respectively.

Data	Group	Method	In-Domain										OOD										Overall ↑
			Cost			Balance			Performance			Sum ↑	Cost			Balance			Performance			Sum ↑	
			C↓	P↑	U↑	C↓	P↑	U↑	C↓	P↑	U↑		C↓	P↑	U↑	C↓	P↑	U↑	C↓	P↑	U↑		
SW	Heuristic	Cheapest	0.33	0.50	-0.17	0.33	0.50	0.08	0.33	0.50	0.33	0.24	0.32	0.46	-0.16	0.32	0.46	0.07	0.32	0.46	0.30	0.21	0.23
		Strongest	0.44	0.63	-0.22	0.44	0.63	0.09	0.44	0.63	0.41	0.28	0.45	0.45	-0.27	0.45	0.45	0.00	0.45	0.45	0.27	0.00	0.14
		Adaptive	0.33	0.50	-0.17	0.44	0.63	0.09	0.44	0.63	0.41	0.33	0.32	0.46	-0.16	0.45	0.45	0.00	0.45	0.45	0.27	0.11	0.22
	Episode-level	ColdStart	0.33	0.55	-0.15	0.32	0.55	0.12	0.32	0.55	0.38	0.35	0.34	0.43	-0.19	0.34	0.43	0.04	0.35	0.43	0.27	0.12	0.24
		GraphRouter	0.33	0.50	-0.17	0.33	0.50	0.08	0.33	0.50	0.33	0.24	0.32	0.46	-0.16	0.32	0.46	0.07	0.32	0.46	0.30	0.21	0.23
		IRT-Router	0.33	0.55	-0.15	0.33	0.53	0.10	0.33	0.55	0.37	0.32	0.33	0.46	-0.17	0.34	0.43	0.04	0.35	0.43	0.27	0.14	0.23
		RouterDC	0.40	0.65	-0.19	0.42	0.59	0.09	0.45	0.61	0.40	0.30	0.39	0.47	-0.22	0.43	0.46	0.02	0.48	0.41	0.23	0.03	0.17
	Multi-turn	MTRouter	0.53	0.70	-0.28	0.53	0.70	0.09	0.53	0.70	0.45	0.26	0.57	0.55	-0.35	0.57	0.55	-0.01	0.57	0.55	0.32	-0.04	0.11
		MTPref	0.38	0.63	-0.18	0.39	0.59	0.10	0.40	0.63	0.42	0.34	0.38	0.49	-0.21	0.44	0.49	0.02	0.44	0.51	0.32	0.13	0.24
		SeqRouter	0.41	0.43	-0.24	0.42	0.58	0.08	0.49	0.57	0.36	0.20	0.44	0.40	-0.28	0.44	0.52	0.04	0.51	0.46	0.27	0.03	0.12
		DialRouter	0.45	0.58	-0.24	0.43	0.55	0.06	0.42	0.63	0.42	0.24	0.43	0.46	-0.25	0.43	0.51	0.04	0.42	0.50	0.32	0.11	0.18
		Ours	0.31	0.63	**-0.12**	0.35	0.66	**0.16**	0.36	0.71	**0.50**	**0.54**	0.34	0.60	**-0.15**	0.35	0.55	**0.10**	0.38	0.57	**0.38**	**0.33**	**0.44**
HLE	Heuristic	Cheapest	0.62	0.18	-0.46	0.62	0.18	-0.22	0.62	0.18	0.02	-0.66	0.60	0.20	-0.44	0.60	0.20	-0.20	0.60	0.20	0.04	-0.60	-0.63
		Strongest	0.69	0.30	-0.50	0.69	0.30	-0.20	0.69	0.30	**0.10**	-0.60	0.72	0.37	-0.50	0.72	0.37	-0.18	0.72	0.37	0.15	-0.53	-0.57
		Adaptive	0.62	0.18	-0.46	0.62	0.18	-0.22	0.69	0.30	**0.10**	-0.58	0.60	0.20	-0.44	0.60	0.20	-0.20	0.72	0.37	0.15	-0.49	-0.54
	Episode-level	ColdStart	0.49	0.08	-0.38	0.52	0.09	-0.22	0.53	0.10	-0.03	-0.63	0.56	0.09	-0.43	0.61	0.16	-0.22	0.59	0.14	-0.01	-0.66	-0.65
		GraphRouter	0.47	0.05	-0.37	0.47	0.05	-0.21	0.47	0.05	-0.06	-0.64	0.51	0.03	-0.40	0.51	0.03	-0.24	0.51	0.03	-0.07	-0.71	-0.68
		IRT-Router	0.47	0.06	-0.36	0.55	0.15	-0.20	0.62	0.23	0.06	-0.50	0.52	0.07	-0.40	0.58	0.18	-0.20	0.66	0.29	0.10	-0.50	-0.50
		RouterDC	0.51	0.10	-0.39	0.52	0.11	-0.21	0.52	0.10	-0.02	-0.62	0.61	0.14	-0.46	0.62	0.17	-0.23	0.63	0.13	-0.02	-0.71	-0.67
	Multi-turn	MTRouter	0.63	0.16	-0.47	0.63	0.16	-0.23	0.63	0.16	0.00	-0.70	0.64	0.21	-0.47	0.64	0.21	-0.22	0.64	0.21	0.04	-0.65	-0.68
		MTPref	0.47	0.08	-0.36	0.55	0.14	-0.20	0.59	0.17	0.02	-0.54	0.52	0.10	-0.40	0.59	0.17	-0.21	0.61	0.17	0.01	-0.60	-0.57
		SeqRouter	0.60	0.19	-0.45	0.64	0.16	-0.24	0.64	0.13	-0.02	-0.71	0.64	0.29	-0.46	0.70	0.35	-0.18	0.65	0.16	0.00	-0.64	-0.68
		DialRouter	0.54	0.11	-0.41	0.53	0.12	-0.20	0.52	0.08	-0.04	-0.65	0.56	0.20	-0.41	0.60	0.19	-0.21	0.58	0.14	0.00	-0.62	-0.64
		Ours	0.33	0.08	**-0.25**	0.47	0.14	**-0.17**	0.64	0.29	**0.10**	**-0.32**	0.44	0.08	**-0.33**	0.57	0.22	**-0.17**	0.70	0.41	**0.19**	**-0.31**	**-0.32**
SWE-V	Heuristic	Cheapest	0.44	0.52	-0.24	0.44	0.52	0.04	0.44	0.52	0.33	0.13	0.47	0.46	-0.28	0.47	0.46	-0.01	0.47	0.46	0.27	-0.02	0.06
		Strongest	0.65	0.62	-0.40	0.65	0.62	-0.02	0.65	0.62	0.37	-0.05	0.67	0.61	-0.41	0.67	0.61	-0.03	0.67	0.61	0.35	-0.09	-0.07
		Adaptive	0.44	0.52	-0.24	0.44	0.52	0.04	0.65	0.62	0.37	0.17	0.47	0.46	-0.28	0.47	0.46	-0.01	0.67	0.61	0.35	0.06	0.12
	Episode-level	ColdStart	0.44	0.52	-0.24	0.44	0.52	0.04	0.44	0.52	0.33	0.13	0.47	0.46	-0.28	0.47	0.46	-0.01	0.47	0.46	0.27	-0.02	0.06
		GraphRouter	0.44	0.52	-0.24	0.44	0.52	0.04	0.44	0.52	0.33	0.13	0.47	0.46	-0.28	0.47	0.46	-0.01	0.47	0.46	0.27	-0.02	0.06
		IRT-Router	0.44	0.52	-0.24	0.44	0.52	0.04	0.44	0.52	0.33	0.13	0.47	0.46	-0.28	0.47	0.46	-0.01	0.47	0.46	0.27	-0.02	0.06
		RouterDC	0.48	0.48	-0.29	0.51	0.54	0.01	0.52	0.56	0.34	0.06	0.51	0.45	-0.32	0.55	0.47	-0.04	0.56	0.49	0.28	-0.08	-0.01
	Multi-turn	MTRouter	0.60	0.62	-0.35	0.60	0.62	0.01	0.60	0.62	0.38	0.04	0.67	0.56	-0.42	0.67	0.56	-0.05	0.67	0.56	0.31	-0.16	-0.06
		MTPref	0.41	0.43	-0.24	0.49	0.50	0.01	0.58	0.54	0.32	0.09	0.46	0.37	-0.29	0.61	0.53	-0.04	0.63	0.55	0.31	-0.02	0.04
		SeqRouter	0.63	0.34	-0.43	0.63	0.43	-0.10	0.55	0.47	0.27	-0.26	0.63	0.24	-0.46	0.64	0.32	-0.16	0.58	0.53	0.31	-0.31	-0.29
		DialRouter	0.56	0.56	-0.34	0.55	0.46	-0.04	0.54	0.52	0.31	-0.07	0.56	0.63	-0.32	0.56	0.67	0.05	0.56	0.63	0.39	0.12	0.03
		Ours	0.43	0.59	**-0.23**	0.47	0.60	**0.06**	0.50	0.65	**0.42**	**0.25**	0.51	0.66	**-0.27**	0.51	0.64	**0.07**	0.60	0.66	**0.41**	**0.21**	**0.23**

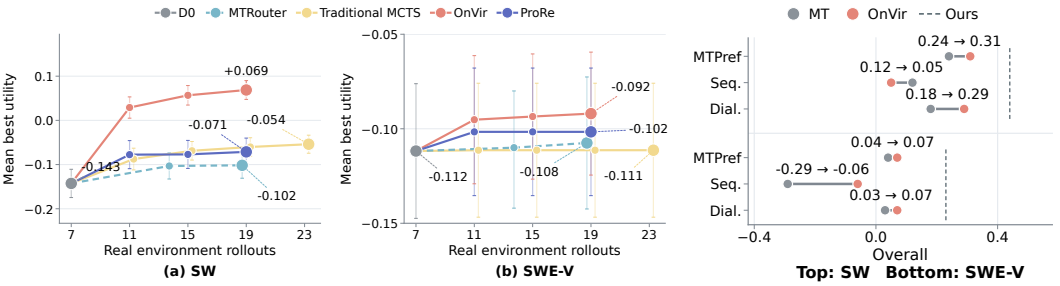

Figure 3: Mean best per-task utility versus the number of real-environment rollouts under the cost preference, across 50 training examples. Within each benchmark, all collection strategies start from the same $\mathcal{D}_0$. Vertical bars show 95% Wald confidence intervals. Markers are slightly offset along the x-axis to avoid overlap.

Figure 4: Overall of multi-turn routers trained on MTRouter data (MT) and OnVir data (OnVir). The dashed line is the performance of our method.

exploration, confirming the role of MCTS. Second, we train MTPref, SeqRouter, and DialRouter on OnVir-collected data and compare them with the same methods trained on MTRouter data. Fig. 4 shows their overall utility on SW and SWE-V, while Table 11 reports the per-preference utilities. The results show that OnVir data consistently improves overall utility, with SeqRouter on SW as the only exception, consistent with the instability of offline Q-learning in long-horizon routing, where value estimation collapses toward a cheap model due to bootstrapping errors. Overall, OnVir improves trajectory quality and provides useful supervision for downstream router learning.

4.3 Ablation Studies

Ablation of the Design of ProRe. Table 2 reports the mean utility over SW and SWE-V for each preference of various variants of ProRe. Per-benchmark results are given in Table 10. The results suggest the following: (1) Removing the macro-action design by setting the action duration to $N = 1$ does not substantially affect routing performance, demonstrating that its main role is to reduce the number of routing decisions. (2) Removing the refiner by letting the proposer greedily execute the

Table 2: Mean utility of ProRe variants across SW and SWE-V. Each preference is the average of SW and SWE-V. Sum is the sum of the three preferences in that row, and Overall averages the two sums. Best values are in **bold**.

Variant	In-Domain				OOD				Overall ↑
	Cost	Bal.	Perf.	Sum	Cost	Bal.	Perf.	Sum	
Ours	-0.18	0.11	**0.46**	0.39	**-0.21**	**0.09**	**0.40**	**0.28**	**0.34**
w/o Macro-action	**-0.15**	**0.15**	**0.46**	**0.46**	-0.22	0.08	0.34	0.20	0.33
w/o Refiner	-0.20	0.07	0.43	0.30	-0.23	0.07	0.35	0.19	0.25
w/o Proposer	-0.17	0.10	0.45	0.38	-0.31	0.05	0.38	0.12	0.25
w/o Plan Condition	-0.21	0.14	0.44	0.37	-0.22	0.08	0.36	0.22	0.30

Table 3: Mean in-domain utility of cache-rate variants across SW and SWE-V. Sum is the sum of the three preferences in that block. Best values in each column are in **bold**.

Method	r=1.0				r=0.5			
	Cost	Bal.	Perf.	Sum	Cost	Bal.	Perf.	Sum
Repricing	**-0.08**	0.16	0.46	0.54	**-0.12**	0.12	0.44	0.44
Retrain	**-0.08**	**0.18**	0.46	**0.56**	-0.14	**0.14**	**0.47**	**0.47**
w/o Proposer	-0.12	**0.18**	0.44	0.50	-0.17	0.10	0.43	0.36
MTPref	-0.11	**0.18**	**0.47**	0.54	-0.15	0.12	0.41	0.38

highest-probability macro-action at each state, or removing the proposer by replacing its sampled plans with a single-model plan per LLM that remains fixed until EOS, both cause a substantial drop. This verifies the necessity of both components, where the refiner supplies a corrective signal, and the proposer supplies high-quality candidate routing plans. (3) Removing the future-plan condition ρ_t^+ from the refiner and conditioning only on the current state s_t lowers utility. This shows that incorporating explicit future information is important for cost and performance estimation, and that implicit future modeling is insufficient.

Cache-Rate Compatibility. We design the refiner's cost estimator to predict per-turn input and output token counts rather than monetary cost, so the same predictions can be repriced under different prompt-cache hit rates r. We vary $r \in \{0.5, 1.0\}$ and report mean in-domain utility on SW and SWE-V in Table 3, with per-benchmark results in Table 12. We compare three variants: Repricing uses the proposer trained at r=0 and prices token predictions at the target r during refiner ranking; Retrain relabels OnVir trajectories collected at r=0 and retrains the proposer, whereas MTPref relabels MTRouter data and is retrained for each r; w/o Proposer is the heuristic variant from the ablation above. Retrain achieves the highest summed utility, while Repricing remains competitive without retraining and outperforms MTPref at r=0.5. This shows that predicting token counts rather than monetary cost can effectively mitigate cache-rate variations. However, the gap between Repricing and MTPref vanishes at r=1.0, as expected, since OnVir data were collected at r=0 and relabeling cannot fully bridge the behavior–expert gap. These results highlight the importance of both data quality and stable cache rates for reliable downstream routing.

Number of Sampled Plans. We evaluate the effect of the number of sampled plans S_I from the proposer of ProRe in Fig. 5, using a 30-example subset of SW to reduce cost. The results show that utility is mainly sensitive to S_I under the performance preference, with in-domain utility peaking at $S_I = 16$ and OOD utility peaking earlier at $S_I = 4$. Balance and cost utilities also fluctuate across S_I, and a larger S_I does not yield consistent gains. These non-monotonic results suggest diminishing returns and potential noise from larger S_I, while the moderate default $S_I = 8$ remains reasonable.

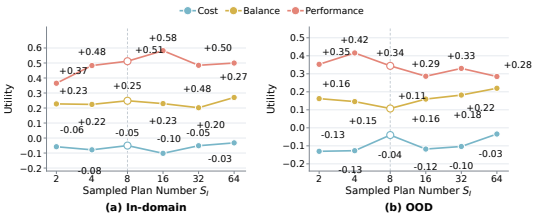

Figure 5: Utility versus the number of sampled plans S_I for subsequent plan refinement on SW.

5 Conclusion

Learning effective multi-turn routers poses two key challenges. First, estimating long-horizon action values that depend on subsequent routing plans is difficult. Second, acquiring high-quality supervision under a limited sampling budget is intractable, as costly rollouts and exponential search space hinder coverage. We address these challenges with ProRe, which samples complete future routing plans with variable-duration macro-actions and uses a plan-conditioned refiner to estimate terminal performance and cumulative cost before executing the first macro-action. We further introduce OnVir, which augments MCTS with virtual rollouts in which the proposer completes search prefixes and the refiner evaluates them to provide search signals, thereby avoiding costly real-environment rollouts. Across three long-horizon benchmarks and different user preferences, ProRe outperforms existing methods in routing performance and performance–cost trade-offs, while OnVir obtains higher-quality training trajectories under comparable budgets.

AI USE STATEMENT

In this work, we used generative AI tools to implement and debug the proposed method and the experimental pipeline, and to check the proofs. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology, generate synthetic data, assist with translation, support qualitative and thematic data analysis, or interpret results. Cleaning and reformatting datasets is not applicable to this work. Additionally, we used generative AI tools to create and edit the manuscript to improve readability. The remaining recommended disclosure tasks were not used or are not applicable.

We have reviewed all AI-assisted work. Code produced with AI assistance was inspected and tested by the authors. All AI-polished text was checked by the authors for factual accuracy and originality. Experimental code and results produced with AI assistance were reviewed by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not raise ethical concerns. It does not involve human subjects, personal data, or other issues covered by the Code of Ethics.

REPRODUCIBILITY STATEMENT

Due to time constraints at submission, we do not release the source code or the collected data with this paper. We will open-source both the code and the data upon acceptance. To help reproducibility, we provide detailed algorithms and experimental settings in the Appendix.

REFERENCES

Ruth Appel, Peter McCrory, Alex Tamkin, Michael Stern, Miles McCain, and Tyler Neylon. Anthropic economic index report: Uneven geographic and enterprise ai adoption, 2025. URL www.anthropic.com/research/anthropic-economic-index-september-2025-report.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation, 2024a.

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

Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. Routerdc: Query-based router by dual contrastive learning for assembling large language models. *Advances in Neural Information Processing Systems*, 37:66305–66328, 2024b.

Ke Cheng, Wen Hu, Zhi Wang, Peng Du, Jianguo Li, and Sheng Zhang. Enabling efficient batch serving for lmaas via generation length prediction. In *2024 IEEE International Conference on Web Services (ICWS)*, pp. 853–864. IEEE, 2024.

Jesse Clifton and Eric Laber. Q-learning: Theory and applications. *Annual Review of Statistics and Its Application*, 7(1):279–301, 2020.

Xiangxiang Dai, Jin Li, Xutong Liu, Anqi Yu, and John Lui. Cost-effective online multi-llm selection with versatile reward models. *arXiv preprint arXiv:2405.16587*, 2024.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*, 2024.

Jingtao Ding, Yunke Zhang, Yu Shang, Yuheng Zhang, Zefang Zong, Jie Feng, Yuan Yuan, Hongyuan Su, Nian Li, Nicholas Sukiennik, et al. Understanding world or predicting future? a comprehensive survey of world models. *ACM Computing Surveys*, 58(3):1–38, 2025.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. In *The Thirteenth International Conference on Learning Representations*, 2025.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. Routerbench: A benchmark for multi-llm routing system. *arXiv preprint arXiv:2403.12031*, 2024.

Zhehao Huang, Baijiong Lin, Jingyuan Zhang, Jingying Wang, Yuhang Liu, Ning Lu, Tao Li, and Xiaolin Huang. Vl-routerbench: A benchmark for vision-language model routing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9512–9523, 2026.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*, 2023.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, Yang Liu, and Yahui Zhou. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. *arXiv preprint arXiv:2507.01352*, 2025.

Hui Liu, Bin Zou, Kecheng Chen, Jie Liu, Wenya Wang, and Haoliang Li. Task-aware llm routing with multi-level task-profile-guided data synthesis for cold-start scenarios. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22047–22076, 2026.

Haoxuan Ma, Guannan Lai, and Han-Jia Ye. Mmr-bench: A comprehensive benchmark for multi-modal llm routing. *arXiv preprint arXiv:2601.17814*, 2026.

Kai Mei, Wujiang Xu, Shuhang Lin, and Yongfeng Zhang. Omnirouter: Budget and performance controllable multi-llm routing. *arXiv preprint arXiv:2502.20576*, 2025.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Yun Qu, Yuhang Jiang, Boyuan Wang, Yixiu Mao, Cheems Wang, Chang Liu, and Xiangyang Ji. Latent reward: Llm-empowered credit assignment in episodic reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 20095–20103, 2025.

Nived Rajaraman, Lin Yang, Jiantao Jiao, and Kannan Ramchandran. Toward the fundamental limits of imitation learning. *Advances in Neural Information Processing Systems*, 33:2914–2924, 2020.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Wei Song, Zhenya Huang, Cheng Cheng, Weibo Gao, Bihan Xu, GuanHao Zhao, Fei Wang, and Runze Wu. Irt-router: Effective and interpretable multi-llm routing via item response theory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15629–15644, 2025.

KV Srivatsa, Kaushal Kumar Maurya, and Ekaterina Kochmar. Harnessing the power of multiple minds: Lessons learned from llm routing. *arXiv preprint arXiv:2405.00467*, 2024.

Haoxiang Wang, Wei Xiong, Tengyang Xie, Han Zhao, and Tong Zhang. Interpretable preferences via multi-objective reward modeling and mixture-of-experts. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10582–10592, 2024.

Zhongling Xu, Shunan Zheng, and Wei Wang. Seqroute: Global budget-aware sequential llm routing via offline reinforcement learning. *arXiv preprint arXiv:2605.25424*, 2026.

Junfei Zhan, Haoxun Shen, Zheng Lin, and Tengjiao He. Prism: Privacy-aware routing for adaptive cloud–edge llm inference via semantic sketch collaboration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 28150–28158, 2026.

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. *arXiv preprint arXiv:2506.09033*, 2025a.

Jiarui Zhang, Xiangyu Liu, Yong Hu, Chaoyue Niu, Hang Zeng, Shaojie Tang, Fan Wu, and Guihai Chen. From myopic selection to long-horizon awareness: Sequential llm routing for multi-turn dialogue. *arXiv preprint arXiv:2604.12385*, 2026a.

Jieyu Zhang, Ranjay Krishna, Ahmed H Awadallah, and Chi Wang. Ecoassistant: Using llm assistant more affordably and accurately. *arXiv preprint arXiv:2310.03046*, 2023.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b.

Yi-Kai Zhang, De-Chuan Zhan, and Han-Jia Ye. Capability instruction tuning: A new paradigm for dynamic llm routing. *arXiv preprint arXiv:2502.17282*, 2025c.

Yiqun Zhang, Hao Li, Zihan Wang, Shi Feng, Xiaocui Yang, Daling Wang, Bo Zhang, Lei Bai, and Shuyue Hu. Mtrouter: Cost-aware multi-turn llm routing with history–model joint embeddings. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44206–44226, 2026b.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

APPENDIX OVERVIEW

The appendix is organized as follows: Appendix A provides further details of the proposed method. It describes the proposer and refiner architectures (Appendix A.1), the training procedure for online updates and final evaluation (Appendix A.2), and the inference and collection algorithms for ProRe and OnVir (Appendix A.3).

Appendix B reports the experimental setup. It describes the benchmarks, data splits, and the LLM pool with pricing (Appendix B.1), the episode-level and multi-turn baselines (Appendix B.2), the training and inference configurations (Appendix B.3), and trajectory replay during OnVir collection (Appendix B.4).

Appendix C presents additional experimental results, including supplementary ablations of the ProRe design and of training-data sources, the effects of the collection search budget, the mixing coefficient, and online learning, and a comparison of the learned performance head with pretrained reward models.

Appendix D discusses the limitations of the present study and directions for future work, including scaling the training data and the router, and integrating separately trained preferences into a single model.

A DETAILS OF METHOD

A.1 IMPLEMENTATION OF PRORE

Proposer details. The proposer generates a future routing plan in an autoregressive manner. At decoding step j, let $\boldsymbol{\rho}^{+}_{t:j-1} = (\mathbf{a}_t, \ldots, \mathbf{a}_{j-1})$ denote the token embeddings of the already generated action prefix $(a_t, \ldots, a_{j-1})$. These embeddings are constructed as in Eq. 2 but without the usage term, since the corresponding macro-actions have not yet been executed. We prepend a learned BOS token and add positional information, obtaining the prefix embeddings $\mathbf{Y}_{t,j} = [\mathbf{e}_{\mathrm{BOS}}; \boldsymbol{\rho}^{+}_{t:j-1}]$. The decoder then conditions on $\mathbf{Y}_{t,j}$ and the state information $\mathbf{H}_t$ to produce a hidden state $\mathbf{z}_j$, which summarizes the generated prefix and the conditioning context and is used to predict the next macro-action a_j. For clarity, we illustrate the computation using a single decoder layer:

$$\mathbf{z}_j = \mathrm{MHA}(\mathbf{Q} = \mathrm{MHA}_{\mathrm{causal}}(\mathbf{Q} = \mathbf{Y}_{t,j}, \mathbf{K} = \mathbf{Y}_{t,j}, \mathbf{V} = \mathbf{Y}_{t,j}), \mathbf{K} = \mathbf{H}_t, \mathbf{V} = \mathbf{H}_t)_{\mathrm{last}}\,, \quad (7)$$

where MHA denotes multi-head attention, and $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ specify its query, key, and value inputs. The inner $\mathrm{MHA}_{\mathrm{causal}}$ applies causal self-attention over the prefix embeddings, and the outer MHA applies cross-attention to $\mathbf{H}_t$. The subscript last selects the representation of the final prefix token, yielding the decoder hidden state $\mathbf{z}_j$. Projections, residual connections, layer normalization, and feed-forward blocks are omitted for brevity.

Given $\mathbf{z}_j$, two linear softmax heads produce distributions over the model vocabulary $\mathcal{V}_m$ and the duration vocabulary $\mathcal{V}_n$, respectively. For a macro-action (m, n),

$$P_\theta(a_j = (m, n) \mid s_t, \lambda, a_{t:j-1}) = P^m_\theta(m \mid \mathbf{z}_j) P^n_\theta(n \mid \mathbf{z}_j). \quad (8)$$

The termination probability is $P^m_\theta(\mathrm{EOS} \mid \mathbf{z}_j)$, and the duration head is ignored when EOS is selected. The BOS token initializes autoregressive decoding but is never predicted.

Refiner details. The refiner comprises a performance predictor V_ϕ and a cost predictor C_ψ. For clarity, we similarly illustrate the computation using a single Transformer layer and omit other layers, residual connections, layer normalization, and feed-forward blocks. For a candidate future routing plan $\rho^{+}_{t,i} \in \mathcal{P}_t$, the performance predictor first forms $\mathbf{X}_{t,i} = [\mathbf{e}_{\mathrm{CLS}}; \mathbf{H}_t; \boldsymbol{\rho}^{+}_{t,i}]$, where $\mathbf{e}_{\mathrm{CLS}}$ is a learned token, $\mathbf{H}_t$ is the conditioning information, and $\boldsymbol{\rho}^{+}_{t,i}$ denotes the embeddings of the candidate plan. A self-attention Transformer encoder then computes:

$$\mathbf{v}_{t,i} = \mathrm{MHA}(\mathbf{Q} = \mathbf{X}_{t,i}, \mathbf{K} = \mathbf{X}_{t,i}, \mathbf{V} = \mathbf{X}_{t,i})[0], \quad (9)$$

where $[0]$ selects the CLS representation. A two-layer MLP with GELU activation maps $\mathbf{v}_{t,i}$ to a normalized terminal-performance estimate $\hat{R}_{t,i}$, which is clipped to $[0, 1]$ during ranking.

For cost prediction, C_ψ first expands each macro-action (m, n) into n unit-duration actions $(m, 1)$, yielding an expanded plan $\widehat{\rho}^{+}_{t,i}$ of length $L_{t,i} = \sum_{j=t}^{T_{t,i}-1} n_j$, where $T_{t,i}$ is the number of macro-actions in $\rho^{+}_{t,i}$. Let $\widehat{\boldsymbol{\rho}}^{+}_{t,i}$ denote the embeddings of the expanded plan with positional information, and let $\tilde{m}_{t,i,\ell}$ denote the model assigned to future turn ℓ. The cost encoder takes $\mathbf{X}^{C}_{t,i} = [\mathbf{e}_{\mathrm{CLS}}; \mathbf{H}_t; \widehat{\boldsymbol{\rho}}^{+}_{t,i}]$ as input and computes

$$\mathbf{C}_{t,i} = \mathrm{MHA}(\mathbf{Q} = \mathbf{X}^{C}_{t,i}, \mathbf{K} = \mathbf{X}^{C}_{t,i}, \mathbf{V} = \mathbf{X}^{C}_{t,i})_{\mathrm{future}}, \tag{10}$$

where the subscript future retains only the representations of the expanded future plan. For each future turn ℓ, an MLP predicts log-transformed usage for input tokens, output tokens, and optional tool calls, yielding the nonnegative estimates $\hat{\mathbf{u}}_{t,i,\ell} = (\hat{u}^{\mathrm{in}}_{t,i,\ell}, \hat{u}^{\mathrm{out}}_{t,i,\ell}, \hat{u}^{\mathrm{tool}}_{t,i,\ell}) = \max\{\exp(\mathrm{MLP}_C(\mathbf{C}_{t,i}[\ell])) - 1, 0\}$. Predicting usage before applying prices allows the refiner to evaluate under different cache-hit rates r. The predicted remaining cost is

$$\widehat{C}_r(s_t, \rho^{+}_{t,i}) = \sum_{\ell=0}^{L_{t,i}-1} \left(\hat{u}^{\mathrm{in}}_{t,i,\ell} p^{\mathrm{in}}_{\tilde{m}_{t,i,\ell}}(r) + \hat{u}^{\mathrm{out}}_{t,i,\ell} p^{\mathrm{out}}_{\tilde{m}_{t,i,\ell}} + \hat{u}^{\mathrm{tool}}_{t,i,\ell} p^{\mathrm{tool}} \right), \tag{11}$$

where $p^{\mathrm{in}}_m(r)$, p^{out}_m, and p^{tool} denote the effective input-token, output-token, and per-call tool prices of corresponding m. Finally, the predicted utility is $\widehat{U}_r(s_t, \rho^{+}_{t,i}; \lambda) = \lambda_p \hat{R}_{t,i} - \lambda_c \,\mathrm{Norm}_C(C_r(\rho_t) + \widehat{C}_r(s_t, \rho^{+}_{t,i}))$, where $C_r(\rho_t)$ is the cost incurred by the executed routing history under cache-hit rate r. ProRe ranks candidate plans by $\widehat{U}_r$ and selects the highest-utility one.

A.2 Details of Training

Data construction. For each routing decision at step t in every completed trajectory $\tau \in \mathcal{D}^{(q)}$, we create a training example from the context (s_t, λ, ρ_t), the observed future plan ρ^{+}_t, the normalized terminal performance $y = \mathrm{Norm}_R(R(\tau))$, where $R(\tau)$ is the terminal task performance of τ, and per-turn usage for input tokens, output tokens, and optional tool calls. For cost-predictor training, we expand ρ^{+}_t into unit-duration turns and record a usage vector $\mathbf{u}_{e,j}$ for each expanded turn j, with components for input-token, output-token, and tool-call usage if applicable (i.e., HLE benchmark).

Proposer training. For the proposer P_θ, we group examples by routing context (s_t, λ, ρ_t) and retain only groups containing at least two distinct observed future plans. From each retained group, we select the plan whose complete trajectory achieves the highest realized utility under λ, denoted by $\rho^{+,\star}_t$. Let $\mathcal{E}_{\mathrm{plan}}$ denote the resulting set of tuples $(s_t, \lambda, \rho_t, \rho^{+,\star}_t)$. For each selected plan, let $\bar{m}_j \in \mathcal{V}_m$ denote the target model symbol or EOS at prediction step j, and let $\bar{n}_j \in \mathcal{V}_n$ denote the target duration when $\bar{m}_j \neq \mathrm{EOS}$. With the decoder state $\mathbf{z}_j$ obtained under teacher forcing on the gold prefix, the proposer is trained by minimizing

$$\mathcal{L}_{\mathrm{plan}}(\theta) = -\mathbb{E}_{(s_t, \lambda, \rho_t, \rho^{+,\star}_t) \sim \mathcal{E}_{\mathrm{plan}}} \sum_{j=0}^{J} \left[\log P^{m}_\theta(\bar{m}_j \mid \mathbf{z}_j) + \mathbf{1}[\bar{m}_j \neq \mathrm{EOS}] \log P^{n}_\theta(\bar{n}_j \mid \mathbf{z}_j) \right]. \tag{12}$$

Refiner training. For the refiner, each real routing decision yields an example $(s_t, \lambda, \rho_t, \rho^{+}_t, y, \{\mathbf{u}_{e,\ell}\}_\ell)$. Let $\mathcal{E}_{\mathrm{ref}}$ denote the set of such examples. The performance predictor V_ϕ and the cost predictor C_ψ are trained independently. The performance predictor is trained by minimizing the squared error

$$\mathcal{L}_{\mathrm{perf}}(\phi) = \mathbb{E}_{(s_t, \lambda, \rho_t, \rho^{+}_t, y) \sim \mathcal{E}_{\mathrm{ref}}} \left[V_\phi(s_t, \lambda, \rho_t, \rho^{+}_t) - y \right]^2. \tag{13}$$

For the cost predictor, let ℓ index the expanded future turns of example e, and let $\mathbf{u}_{e,\ell}$ denote the observed usage vector. Let $\hat{\mathbf{z}}_{\psi,e,\ell} = \mathrm{MLP}_C(\mathbf{C}_e[\ell])$ denote the predicted log usage before the $\exp(\cdot)-1$ transformation. The cost loss is

$$\mathcal{L}_{\mathrm{cost}}(\psi) = \frac{1}{|\mathcal{E}_{\mathrm{ref}}|} \sum_{e \in \mathcal{E}_{\mathrm{ref}}} \sum_{\ell=0}^{L_e-1} \left\| \hat{\mathbf{z}}_{\psi,e,\ell} - \log(1 + \mathbf{u}_{e,\ell}) \right\|_2^2, \tag{14}$$

where the logarithm is applied element-wise. The refiner thus learns to score a given future routing plan according to a predefined performance–cost preference.

Online-training and Final Evaluation. After each collection round q, we optimize the proposer and refiner on $\mathcal{D}^{(q)}$ and use the updated parameters in the next OnVir round. The three components are optimized independently. For final evaluation, we train all components on the full collected dataset $\mathcal{D}^{(Q)} = \mathcal{D}_0 \cup \bigcup_{i=1}^{Q} D_i$, where Q is the number of online rounds. At inference time, for each routing state s_t, ProRe samples S_I future plans from the proposer and re-ranks them using the refiner, as described in Sec. 3.2.

A.3 Inference and Collection Algorithms

Algorithm 1 describes ProRe inference, and Algorithm 2 describes OnVir collection.

Algorithm 1 ProRe: propose-and-refine inference

Require: Task x, preference λ, proposer P_θ, refiner (V_ϕ, C_ψ), candidate count S_I

Initialize $t \leftarrow 0$ and $s_t = (x, h_t, \rho_t)$ with empty interaction and routing histories
while task is unfinished and execution budget remains **do**
  Sample S_I complete candidate plans $\mathcal{P}_t = \{\rho_{t,i}^{+}\}_{i=1}^{S_I}$ from $P_\theta(\cdot \mid s_t, \lambda, \rho_t)$
  for each $\rho_{t,i}^{+} \in \mathcal{P}_t$ **do**
    Predict terminal performance and future usage with (V_ϕ, C_ψ); compute $\widehat{U}_r(s_t, \rho_{t,i}^{+}; \lambda)$ as in Sec. 3.2
  end for
  $\rho_t^{+,\star} \leftarrow \arg\max_{\rho_{t,i}^{+} \in \mathcal{P}_t} \widehat{U}_r(s_t, \rho_{t,i}^{+}; \lambda)$
  Execute the first macro-action of $\rho_t^{+,\star}$ in the real environment
  Append the observed transition and usage to the interaction history; update s_{t+1} and ρ_{t+1}
  $t \leftarrow t + 1$
end while
return Completed trajectory and observed outcome

Algorithm 2 OnVir: online collection with virtual rollouts

Require: Bootstrap trajectories $\mathcal{D}_0$, rounds Q, task–preference schedules $\{\mathcal{S}_q\}_{q=1}^{Q}$ with B episodes per task per round, virtual simulations per decision S_V, c_{puct}, and η

$\mathcal{D}^{(0)} \leftarrow \mathcal{D}_0$
for $q = 1, \ldots, Q$ **do**
  Train $(P_\theta^q, V_\phi^q, C_\psi^q)$ on $\mathcal{D}^{(q-1)}$ and freeze parameters during collection
  $D_q \leftarrow \emptyset$
  for each scheduled task–preference pair $(x, \lambda) \in \mathcal{S}_q$ **do**
    Initialize $t \leftarrow 0$, real state $s_t = (x, h_t, \rho_t)$, and trajectory $\tau \leftarrow \emptyset$
    while task is unfinished and execution budget remains **do**
      Set the root prefix to ρ_t; initialize its real statistics from completed trajectories in $\mathcal{D}^{(q-1)} \cup D_q$ with the same (x, λ, s_t)
      Initialize real statistics of each first-level child $[\rho_t; a]$ from matching trajectories that executed a at s_t; set deeper real statistics and all virtual statistics to zero
      for $b = 1, \ldots, S_V$ **do**
        Select and expand a prefix ρ from the root by PUCT in Eq. (5), using $p_\rho(a) \propto P_{\theta^q}(a \mid s_t, \lambda, \rho)$ normalized over executable actions; record path $\mathcal{P}$
        Sample completion $\rho^+ \sim P_{\theta^q}(\cdot \mid s_t, \lambda, \rho)$ until EOS or the remaining-budget boundary
        $\hat{u} \leftarrow \widehat{U}_r(s_t, [\rho; \rho^+]; \lambda)$ using (V_ϕ^q, C_ψ^q)
        for each node $\rho' \in \mathcal{P}$ **do**
          $N^{\mathrm{vir}}(\rho') \leftarrow N^{\mathrm{vir}}(\rho') + 1$; $W^{\mathrm{vir}}(\rho') \leftarrow W^{\mathrm{vir}}(\rho') + \hat{u}$
        end for
      end for
      Sample root action a_t with probability $N^{\mathrm{vir}}([\rho_t; a]) / \sum_{a' \in \mathcal{A}(\rho_t)} N^{\mathrm{vir}}([\rho_t; a'])$
      if D_q contains a completed trajectory with the same state–action pair (s_t, a_t) **then**
        Reuse its logged model outputs and observed transition
      else
        Execute a_t in the real environment
      end if
      Append the observed transition to τ and update s_{t+1} and ρ_{t+1}
      $t \leftarrow t + 1$; restart search at the new state
    end while
    $D_q \leftarrow D_q \cup \{\tau\}$ ▷ Add each completed episode immediately for reuse in later searches
  end for
  $\mathcal{D}^{(q)} \leftarrow \mathcal{D}^{(q-1)} \cup D_q$
end for
Train final $(P_\theta, V_\phi, C_\psi)$ on $\mathcal{D}^{(Q)}$ for evaluation
return Trained ProRe and cumulative real trajectories $\mathcal{D}^{(Q)}$

B EXPERIMENTAL DETAIL

B.1 BENCHMARKS AND LLM POOL

SCIENCEWORLD (SW) is a text-based scientific environment in which an agent executes procedural experiments via multi-step commands, with performance measured by the terminal environment score in $[-100, 100]$. HUMANITY'S LAST EXAM (HLE) contains challenging cross-domain questions requiring multi-step reasoning and tool use, where agents may invoke a paid web search tool, and performance is binary correctness. SWE-BENCH VERIFIED (SWE-V) requires resolving real GitHub issues within repository sandboxes, evaluated by the official test harness. We set the maximum number of environment turns L to 50 for SW and SWE-V, and 30 for HLE, with per-episode cost budgets of 2, 4, and 4 USD, respectively.

Following prior work (Zhang et al., 2026b), we construct in-domain (ID) and out-of-domain (OOD) splits by task domain, so that OOD evaluation tests transferability to held-out task families rather than to a random partition of near-duplicate instances. For SW, ID and OOD use disjoint task types, and ID variations are split into train/validation/test partitions with seed 42. For OOD, whereas prior work (Zhang et al., 2026b) used all remaining task types (over 300 instances), we sample a fixed subset with balanced domain coverage to reduce experimental cost. For HLE, we exclude multimodal questions and split academic categories into ID and OOD groups. For SWE-V, ID and OOD contain disjoint repositories. Statistics for each benchmark are summarized in Table 4.

Model pool and pricing. We construct the LLM pool $\mathcal{M}$ from seven widely used models spanning diverse capability and cost tiers: DeepSeek V4 Flash, DeepSeek V4 Pro, GPT-5.4, GPT-5.4 Mini, GPT-5.4 Nano, Gemini 3.1 Flash-Lite, and MiniMax M3. Table 5 reports the token prices used throughout the experiments. Input prices correspond to cache misses; cached-input prices are used only in the cache-rate experiments. HLE additionally charges Bocha web search at \$0.00444 per request. SW and SWE-V have no separately billed web-search component.

B.2 BASELINES

All methods use the same seven-model pool. Episode-level and multi-turn baselines are trained on the MTRouter data, which comprises two full-episode rollouts of each of the seven LLMs per training instance and five random-switch (roulette-style) trajectories. Episode-level routers encode only the initial query; unless noted, they use the frozen Qwen3-Embedding-0.6B (Zhang et al., 2025b) encoder shared with MTRouter. Learned routers are trained separately for each cost-performance preference λ, following Feng et al. (2025). We describe each baseline below.

Cheapest LLM. Selects the single model with the highest mean utility on the training data under a cost-first preference $(\lambda_p, \lambda_c) = (0.2, 0.8)$, and uses it for all evaluation preferences.

Strongest LLM. Selects the single model with the highest mean training utility under a performance-first preference $(\lambda_p, \lambda_c) = (0.8, 0.2)$, and uses it for all evaluation preferences.

Adaptive LLM. For each evaluation preference, select the single model with the highest mean training utility under that preference.

ColdStartRouter (Cold). Infers a hierarchical task-profile posterior from the query, predicts per-model performance and cost given that profile, and selects the model with the highest preference utility. Only the Gaussian query posterior, hierarchy latents, and task-conditioned decoders are trained; the query encoder is frozen (Liu et al., 2026).

GraphRouter. Constructs a query–LLM graph with utility and cost edge features, applies two GNN layers, and classifies the unique best LLM for the query. Routing is performed once from the initial query and kept fixed throughout the episode (Feng et al., 2025).

IRT-Router. Maps model ability and query discrimination/difficulty to predicted performance via an M-IRT 2PL head, adds a cost head for log-normalized cost, and selects the model maximizing $(1-\lambda)\hat{R} - \lambda\hat{C}$ on the initial query (Song et al., 2025).

RouterDC. Applies sample–LLM InfoNCE with top-K positives and bottom-K negatives defined by preference utility, together with a cluster contrastive loss. A query projection is trained on frozen

Table 4: Dataset statistics for each benchmark.

Benchmark	ID domains	OOD domains	Train	Val	ID test	OOD test
SW	13 task types	12 task types	218	72	74	72
HLE	6 categories	2 categories	400	100	300	300
SWE-V	8 repos	4 repos	200	100	100	100

Table 5: Token prices used for cost evaluation, in USD ($) per million tokens.

Model	Input	Cached input	Output
DeepSeek V4 Flash	0.140	0.0028	0.280
DeepSeek V4 Pro	0.435	0.003625	0.870
GPT-5.4	2.500	0.250	15.000
GPT-5.4 Mini	0.750	0.075	4.500
GPT-5.4 Nano	0.200	0.020	1.250
Gemini 3.1 Flash-Lite	0.250	0.025	1.500
MiniMax M3	0.300	0.060	1.200

embeddings, and the model with the highest predicted utility is used for the entire episode (Chen et al., 2024b).

MTRouter. Jointly embeds interaction history and candidate-model attributes and regresses the terminal task outcome of executing a model at the current turn. It does not take λ as input, and at test time greedily selects $\arg\max_a \hat{S}(h_t, a)$ (Zhang et al., 2026b).

MTPref. Follows the MTRouter protocol on the same single-model and random-switch trajectories, but replaces the unconditioned outcome target with preference utility $(1-\lambda)R - \lambda C$. The architecture is unchanged, and a separate regressor is trained for each λ (Zhang et al., 2026b).

SeqRouter. Trains a turn-level router by offline Q-learning on the official single-model and random-switch data, using Double DQN / fitted Q-iteration (Qu et al., 2025) with stepwise costs and a terminal performance bonus so that the undiscounted return equals preference utility. It acts by $\arg\max_a Q(h_t, a)$ (Xu et al., 2026).

DialRouter. Trains by imitation learning on the same offline data, where the expert action at each state is the observed model with the highest preference utility. A frozen BGE-M3 (Chen et al., 2024a) retriever fetches similar histories, and a gated classifier (Qwen2.5-0.5B, last three layers unfrozen) predicts the expert model; inference uses retrieval and classification only (Zhang et al., 2026a).

B.3 Implementation Details

Table 6 lists the training-split extrema used by Norm_R and Norm_C. SW performance uses the environment score range $[-100, 100]$, while HLE and SWE-V use binary outcomes in $[0, 1]$. Cost bounds are the minimum and maximum episode costs observed on the training split after repricing at the assumed prompt-cache hit rate r. The table reports the $r = 0$ bounds used in the main experiments and the cost extrema for cache-rate studies on SW and SWE-V at $r = 0.5$ and $r = 1.0$. Following MTRouter (Zhang et al., 2026b), we set the prompt-cache hit rate to $r = 0$ by default unless otherwise specified, as in the cache-rate compatibility study.

Before evaluating episode-level routers and heuristic methods, we evaluate each single-model policy by running that model for the full episode. Consequently, for router baselines that select a model solely from the initial query, we do not reroll the selected LLM. Instead, we reuse the previously collected single-model trajectory for that LLM on the same instance and report its terminal performance and cost, thereby reducing evaluation cost.

For ProRe, the proposer samples $S_I = 8$ complete plans and retains the most recent $L_h = 8$ interaction turns. The maximum macro-action duration is $N = L/10$, giving $N = 5$ on SW and SWE-V and $N = 3$ on HLE. ProRe is initialized from $\mathcal{D}_0$ and then updated with OnVir for $Q = 3$ rounds that collect $B = 4$ real trajectories per instance per round. Concretely, $\mathcal{D}_0$ contains one single-model episode per LLM, so each instance has $7 + 12 = 19$ episodes. Virtual rollout of OnVir uses a search budget of $S_V = 128$. Initialization uses 5 epochs with early stopping (minimum 3

Table 6: Min–max bounds used for performance and log-cost normalization. Performance bounds are repeated for each cache rate r. HLE is evaluated only at $r = 0$.

Benchmark	r	$\min R$	$\max R$	$\min C$ (USD)	$\max C$ (USD)
SW	0	-100	100	9.74×10^{-4}	1.590
SW	0.5	-100	100	5.74×10^{-4}	1.267
SW	1.0	-100	100	1.69×10^{-4}	0.945
HLE	0	0	1	1.85×10^{-4}	2.751
SWE-V	0	0	1	4.56×10^{-4}	3.639
SWE-V	0.5	0	1	9.66×10^{-4}	2.379
SWE-V	1.0	0	1	5.14×10^{-4}	1.233

Table 7: ProRe training and validation counts after the 8:2 task split, averaged over the three preferences. Episodes count complete task-execution trajectories. Proposer examples are states with at least two distinct future plans. Refiner examples are shared by the cost and performance heads.

Benchmark	Split	Tasks	Episodes	Proposer examples	Refiner examples
SW	Train	175	3325	1438	22197
SW	Val	43	817	374	6119
HLE	Train	312	5928	1797	40069
HLE	Val	88	1672	434	11479
SWE-V	Train	158	3001	956	12820
SWE-V	Val	42	798	254	3354

epochs, patience 2), and each subsequent online round trains for 1 epoch under the same rule. For MCTS action selection, we set $c_{\text{puct}} = 1.5$ and $\eta = 0.5$ in fixed mode.

The text embedder is a frozen Qwen3-Embedding-0.6B (Zhang et al., 2025b) encoder with last-token pooling. Each task or history segment is truncated to 2048 tokens before encoding. The resulting 1024-dimensional embeddings are projected into a $d = 512$ hidden space. The proposer is a 2-layer Transformer decoder with 8 heads. Both refiner encoders use the same depth and width, with dropout 0.1. We train with AdamW at learning rate 1×10^{-4}, weight decay 1×10^{-4}, gradient clipping 1.0, and batch size 64.

For all implementations, we follow MTRouter[1] (Zhang et al., 2026b). Although the original paper does not report SWE-V results, its official repository includes the corresponding implementation. For episode-level and multi-turn routers, we use the training and validation splits described in Appendix B.1. For ProRe and OnVir, we further split the original training tasks into training and validation subsets at an 8:2 ratio by task ID, ensuring that all trajectories from the same task remain in the same subset. The held-out validation set is used for ablation studies and checkpoint selection. The proposer and refiner use different training-example construction methods, described in Appendix A.2. Table 7 reports the average counts over the three preferences. Following Feng et al. (2025); Liu et al. (2026), the three preferences are trained separately on the same task split. All experiments are conducted on eight NVIDIA L40 GPUs, with a total experimental cost of approximately 3000 USD. For the DeepSeek-V4 series, we use the preview version.

Following Feng et al. (2025); Liu et al. (2026), the three preferences are trained separately on the same task split. All experiments are conducted using eight NVIDIA L40 GPUs. The total experimental cost is approximately 3000 USD. For the DeepSeek-V4 series, we use the preview version.

B.4 Trajectory Replay during Collection

As described in Sec. 3.3, each completed trajectory is saved into D_q immediately, so later collections in the same round can reuse earlier outcomes. Concretely, the collector indexes all previously executed prefixes, including the $\mathcal{D}_0$ single-model trajectories and trajectories from earlier online

[1] https://github.com/ZhangYiqun018/MTRouter

Table 8: OnVir replay during collection. Online model steps count environment turns with `source=online`. Replayed steps match a stored prefix and issue no new model call. $\mathcal{D}_0$ and earlier rounds partition the replayed steps.

Benchmark	Online steps	Replayed	From $\mathcal{D}_0$	From earlier rounds
SW	218,720	110,238 (50.4%)	79,432	30,806
SWE-V	100,020	41,041 (41.0%)	33,799	7,242
HLE	261,200	70,952 (27.2%)	54,321	16,631
Total	579,940	222,231 (38.3%)	167,552	54,679

Table 9: Replay on the same 50 tasks at $(\lambda_p, \lambda_c) = (0.2, 0.8)$ in the main experimental results in the main body. OnVir counts online collection steps. Traditional MCTS counts the 16 live rollouts per task.

Benchmark	Collector	New steps	Replayed steps	Replay rate
SW	OnVir	7,843	11,297	59.0%
	Traditional MCTS	17,637	450	2.5%
SWE-V	OnVir	4,307	3,043	41.4%
	Traditional MCTS	10,662	450	4.0%

rounds. When the macro-action selected at s_t matches a stored continuation for the same model, OnVir replays the stored model outputs instead of invoking a new model call.

Replay must place the new episode back on the historical trajectory. On SW and HLE, the environment state is the dialogue, so replay just appends the stored messages. On SWE-V, actions modify the workspace. We propose to replay the extracted command sequence that produced the matched state, run those commands in the current Docker code, and restore the stored transcript so that the following state matches the historical prefix.

Table 8 reports replay rates across the nine official OnVir runs. A replayed step is one whose `recovered_from` field names the source trajectory. Sources beginning with `d0:` are historical baseline trajectories; the remaining identifiers are prefixes executed by search in earlier online rounds. The results show that across all three benchmarks, 38.3% of online model steps are replayed. However, such a replay is much weaker for traditional real-rollout MCTS. For the same 50 training tasks at preference $(\lambda_p, \lambda_c) = (0.2, 0.8)$ as in the main experiments of our main paper, even with 16 live rollouts per task, the search tree remains shallow. Consequently, a rollout restores very little of the trajectory and must execute nearly the entire episode from scratch. As shown in Table 9, OnVir replays 59.0% of model steps on SW and 41.4% on SWE-V, whereas traditional MCTS replays only 2.5% and 4.0%. Overall, these results demonstrate the data-efficiency advantage of OnVir.

Table 10: Utility of ProRe variants on SW and SWE-V. Cost, Bal., and Perf. denote cost-first, balance, and performance-first preferences. ID/OOD Sum aggregates the three preferences, and Overall averages the two sums. Best U (and Sum) in each dataset block is in **bold**. Ours matches Table 1.

Dataset	Variant	In-Domain				OOD				Overall ↑
		Cost	Bal.	Perf.	Sum	Cost	Bal.	Perf.	Sum	
SW	Ours	-0.12	0.16	0.50	0.54	**-0.15**	0.10	0.38	**0.33**	**0.44**
	w/o Macro-action	**-0.11**	**0.18**	**0.52**	**0.59**	**-0.15**	0.10	0.33	0.28	**0.44**
	w/o Refiner	-0.14	0.14	0.50	0.50	-0.18	0.09	**0.39**	0.30	0.40
	w/o Proposer	**-0.11**	0.13	0.50	0.52	-0.28	0.10	**0.39**	0.21	0.37
	w/o Plan Condition	-0.14	**0.18**	0.50	0.54	-0.16	0.10	0.33	0.27	0.41
	w/o Rerank	-0.13	0.15	0.48	0.50	-0.18	0.10	0.33	0.25	0.38
	Joint λ	-0.14	0.16	0.48	0.50	-0.16	0.10	0.36	0.30	0.40
SWE-V	Ours	-0.23	0.06	**0.42**	0.25	**-0.27**	**0.07**	**0.41**	**0.21**	**0.23**
	w/o Macro-action	**-0.18**	**0.11**	0.41	**0.34**	-0.29	0.06	0.35	0.12	**0.23**
	w/o Refiner	-0.26	0.00	0.36	0.10	-0.28	0.04	0.30	0.06	0.08
	w/o Proposer	-0.23	0.06	0.40	0.23	-0.34	-0.01	0.36	0.01	0.12
	w/o Plan Condition	-0.27	0.09	0.38	0.20	-0.29	0.06	0.38	0.15	0.18
	w/o Rerank	-0.25	0.03	0.33	0.11	-0.29	0.00	0.32	0.03	0.07
	Joint λ	-0.30	0.07	0.41	0.18	-0.30	0.06	**0.41**	0.17	0.18

Table 11: Utility of multi-turn routers trained on MTRouter data and OnVir-collected data. ID/OOD Sum aggregates the three preferences, and Overall averages the two sums. Within each benchmark and data source, the best value in each column is in **bold**.

Dataset	Data	Method	In-Domain				OOD				Overall ↑
			Cost	Bal.	Perf.	Sum	Cost	Bal.	Perf.	Sum	
SW	MT	MTPref	-0.18	0.10	**0.42**	0.34	-0.21	0.02	0.32	0.13	0.24
		Ours	**-0.13**	**0.11**	**0.42**	**0.40**	**-0.18**	**0.09**	**0.39**	**0.30**	**0.35**
	OnVir	MTPref	-0.19	0.15	0.44	0.40	-0.21	**0.10**	0.32	0.21	0.31
		Ours	**-0.12**	**0.16**	**0.50**	**0.54**	**-0.15**	**0.10**	**0.38**	**0.33**	**0.44**
SWE-V	MT	MTPref	**-0.24**	**0.01**	0.32	**0.09**	**-0.29**	-0.04	0.31	-0.02	**0.04**
		Ours	-0.36	-0.02	**0.35**	-0.03	-0.35	**0.03**	**0.40**	**0.08**	0.03
	OnVir	MTPref	-0.28	0.03	0.36	0.11	-0.31	-0.03	0.36	0.02	0.07
		Ours	**-0.23**	**0.06**	**0.42**	**0.25**	**-0.27**	**0.07**	**0.41**	**0.21**	**0.23**

C Additional Experimental Results

Supplementary Ablation of the Design of ProRe. Beyond the ablations in the main text, we evaluate two further variants in Table 10. The results reveal two findings. First, replacing re-ranking with per-state MCTS, which searches for the best macro-action at every state, lowers utility, with a larger drop on SWE-V. For example, the Overall utility decreases from 0.44 to 0.38 on SW and from 0.23 to 0.07 on SWE-V. This indicates that the proposer already concentrates on a few high-quality plans, while MCTS introduces many additional candidates that interfere with ranking rather than improving it. Second, training a single checkpoint on all three preferences causes a mild drop. For example, the Overall utility decreases from 0.44 to 0.40 on SW and from 0.23 to 0.18 on SWE-V, suggesting that different preferences may conflict within a shared model or that the training data may be insufficient. We leave scaling joint preference training on large-scale data to future work.

Ablation of Different Data Sources. To study how MTRouter data affects our method, we additionally train ProRe on MTRouter data, supplementing the OnVir study in Sec. 4.2. Table 11 shows that on the same data source, ProRe exceeds MTPref on SW under both sources, raising Overall from 0.24 to 0.35 on MTRouter data and from 0.31 to 0.44 on OnVir data. On SWE-V with MTRouter data, the two methods are close, with MTPref attaining an Overall of 0.04 and ProRe attaining 0.03. These comparisons support the effectiveness of ProRe when training data are held fixed. For a fixed method, training on OnVir-collected data improves Overall over training on MTRouter data for both MTPref and ProRe. Online collection, therefore, yields higher-quality trajectories and provides supervision that is closer to the expert routing policy than MTRouter data.

Table 12: In-domain utility of cache-rate variants. Dataset Sum aggregates the three preferences; Overall averages the two dataset sums. Best Overall in each r block is in **bold**. Absolute utilities are not comparable across r.

r	Method	SW				SWE-V				Overall Sum ↑
		Cost	Bal.	Perf.	Sum	Cost	Bal.	Perf.	Sum	
1.0	Repricing	-0.05	0.16	0.49	0.60	-0.11	0.16	0.43	0.48	0.54
	Retrain	-0.05	0.19	0.47	0.61	-0.10	0.16	0.45	0.51	**0.56**
	w/o Proposer	-0.08	0.20	0.43	0.55	-0.15	0.15	0.45	0.45	0.50
	MTPref	-0.08	0.19	0.46	0.57	-0.14	0.16	0.47	0.49	0.53
0.5	Repricing	-0.10	0.16	0.44	0.50	-0.13	0.08	0.44	0.39	0.45
	Retrain	-0.10	0.17	0.44	0.51	-0.18	0.11	0.50	0.43	**0.47**
	w/o Proposer	-0.12	0.14	0.43	0.45	-0.21	0.05	0.43	0.27	0.36
	MTPref	-0.09	0.17	0.44	0.52	-0.20	0.06	0.38	0.24	0.38

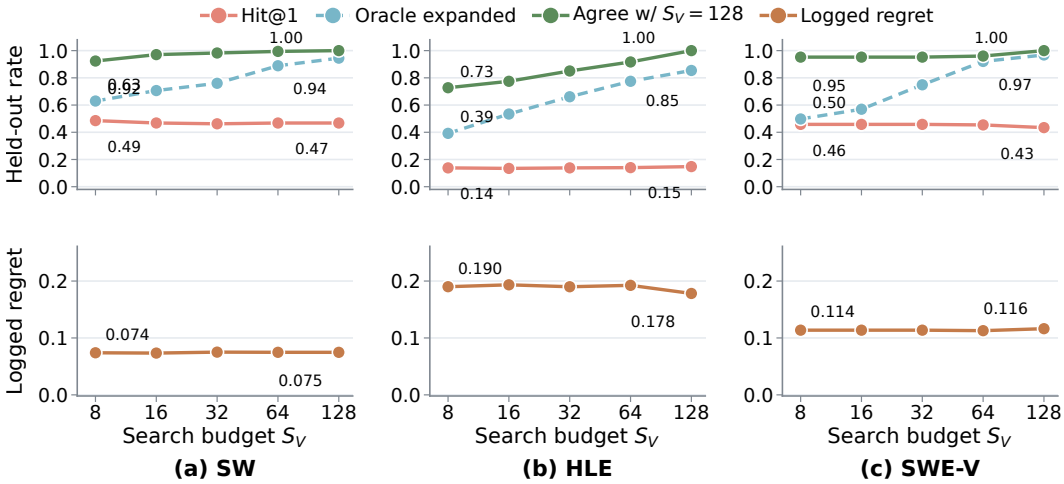

Figure 6: Offline search-budget S_V ablation. Metrics are averaged over three preferences on held-out root states. Hit@1 measures whether the visit-argmax action matches the logged oracle; Oracle expanded indicates whether any simulation visited the oracle; Agreement measures whether the executed root action matches that selected with S_V=128. Logged regret is the utility gap relative to the best logged-first action.

Collection Search Budget. In the main body, we adopt $S_V = 128$ virtual rollouts for OnVir. As described in Appendix B.3, we split the original training set into training and validation subsets in an 8:2 ratio, with the training split used to train ProRe. Here we use the held-out validation split to examine the effect of different numbers of virtual rollouts. Concretely, we group examples by the routing context (s_t, λ, ρ_t) and retain only groups that contain at least two distinct observed future plans. For each group, we run MCTS with virtual rollouts under a total budget of $S_V = 128$ and record the visit counts of the root's child nodes at simulation counts of $S_V \in \{8, 16, 32, 64, 128\}$. We report Hit@1, the fraction of cases where the executed root action matches the logged oracle, Oracle expanded, the fraction where any simulation visited the oracle, Agreement with $S_V = 128$, the fraction where the executed action is unchanged relative to the full budget, and logged regret, the utility gap between the executed action and the best logged first action.

Fig. 6 shows the results. As the budget increases, Oracle expanded rises monotonically, but Hit@1 and logged regret remain nearly flat. Agreement with $S_V = 128$ is already 0.92–0.95 at $S_V = 8$ on SW and SWE-V, indicating that extra simulations rarely change the most probable action. On HLE, Agreement starts lower at 0.73 and rises with the budget, so additional search can still affect the executed action in some scenarios. Overall, since virtual rollouts are much cheaper than real environment rollouts, $S_V = 128$ or a larger budget is a reasonable default.

Collection Mixing Coefficient. When a node has both real and virtual statistics, OnVir mixes them as $Q_\eta = \eta Q^{\text{real}} + (1 - \eta) Q^{\text{vir}}$, as described in Sec. 3.3. Collection uses $\eta = 0.5$ in the

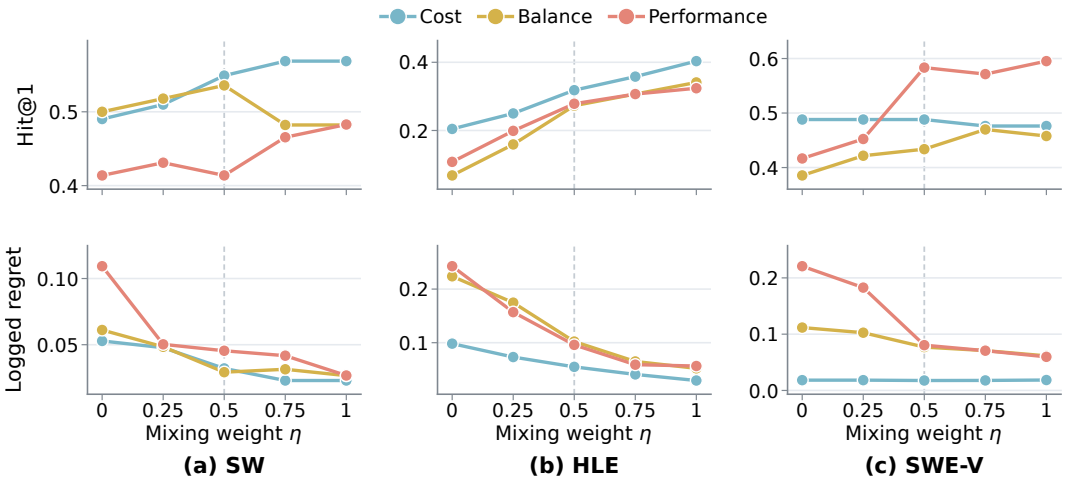

Figure 7: Held-out mixing-coefficient ablation at the last collection round. The dashed vertical line is the collection setting η=0.5. Hit@1 is whether visit-argmax matches the logged oracle first action. Logged regret is the utility gap between the executed action and the best logged first action.

Table 13: Our performance estimator versus two pretrained reward models (1500 examples per benchmark). Pairwise is ranking accuracy on pairs with distinct targets. RMSE applies only to Ours. The best Spearman, Pearson, and pairwise values in each benchmark are in **bold**.

Benchmark	Method	Spearman	Pearson	Pairwise	RMSE
SW	Ours	**0.664**	**0.704**	**77.5%**	0.266
	Skywork	0.203	0.112	57.9%	—
	ArmoRM	−0.065	0.013	47.1%	—
SWE-V	Ours	**0.473**	**0.485**	**78.4%**	0.447
	Skywork	0.083	0.059	55.0%	—
	ArmoRM	−0.045	−0.022	46.8%	—
HLE	Ours	**0.134**	**0.216**	**64.9%**	0.270
	Skywork	0.012	0.017	51.2%	—
	ArmoRM	−0.052	−0.071	43.8%	—

main experiment. To study this choice, we evaluate the last collection round on a held-out validation split, following the protocol in the collection search budget ablation. We freeze the checkpoint from the previous round, warm-start real backups from earlier trajectories, and resimulate with $\eta \in \{0, 0.25, 0.5, 0.75, 1\}$ at $S_V = 128$ using visit-argmax. Fig. 7 reports Hit@1 and logged regret. The results show that increasing η generally improves Hit@1, with clear gains on HLE and SWE-V and a non-monotone effect on SW. Logged regret decreases consistently across benchmarks and preferences, indicating that trusting real statistics more shrinks the utility gap even when Hit@1 is flat. Given that Hit@1 declines for some preferences, we set the mixing coefficient to the intermediate value $\eta = 0.5$.

Effect of Online Learning on Performance of ProRe. OnVir updates ProRe after each collection round. To test whether these updates help, we evaluate checkpoints from rounds 1, 2, and 3 on the same held-out validation split, following the protocol in the collection search budget ablation. Fig. 8 reports Hit@1 and logged regret for the three preferences and their mean. Averaged over preferences, Hit@1 improves from round 1 to round 3 on all three benchmarks, and logged regret decreases in most benchmark–preference cells. The gains are steady on SW, whereas HLE and SWE-V show minor fluctuations but remain above round 1. Iterative online training, therefore, improves ProRe in most settings while a three-round schedule is sufficient, and extending it is unlikely to change the final data quality.

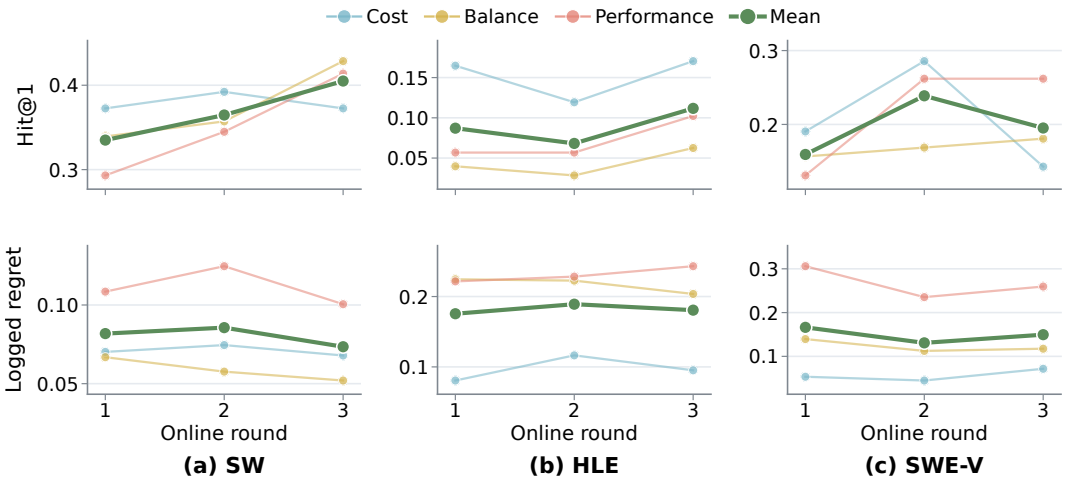

Figure 8: Held-out online-training ablation. Rounds 1–3 are ProRe checkpoints trained on $\mathcal{D}_0$, $\mathcal{D}_0 \cup D_1$, and $\mathcal{D}_0 \cup D_1 \cup D_2$. Hit@1 is whether sample-and-re-rank matches the logged oracle first action. Logged regret is the utility gap between the executed action and the best logged first action. The mean is the unweighted average over the three preferences.

Performance Head versus Pretrained Reward Models. Inspired by SeqRouter (Xu et al., 2026), which uses reward models to estimate immediate turn rewards, we consider pretrained reward models as an alternative to our learned performance estimator. Following the previous ablation setup, we use the held-out validation split and sample 500 predictor examples per preference for each benchmark, each containing the task description, interaction history, and terminal performance. We compare with Skywork-Reward-V2-Llama-3.1-8B (Liu et al., 2025) and ArmoRM-Llama3-8B-v0.1 (Wang et al., 2024), the latter used in SeqRouter. Since reward-model scores and task returns are not on the same scale, we report Spearman and Pearson correlations and pairwise ranking accuracy. RMSE is reported only for our estimator in Table 13. Our estimator ranks pairs substantially better than both chat reward models on all three benchmarks. Skywork is only slightly above chance on SW and SWE-V, while ArmoRM is at or below chance. We attribute this to the reward model aligning with human preferences rather than terminal task performance, and to the difficulty of predicting terminal performance in long-horizon tasks without task-specific fine-tuning.

D Limitation and Future Work

First, we train ProRe with fixed data and model scale, using the task splits in Appendix B.1, following previous work (Zhang et al., 2026b). While this suffices for the studied benchmarks, scaling to larger datasets and models towards general application remains future work, but is promising. Second, following prior work (Feng et al., 2025; Liu et al., 2026), we train separate checkpoints for each cost–performance preference. Integrating them into a single preference-conditioned model is left for future work. Although the joint-λ ablation in Appendix C shows a mild utility drop at the current scale, it will diminish with lots of training data and a bigger model parameter scale.