MultiTurnRoute: A Platform for Building and Evaluating Multi-Turn LLM Routers

Jiaqi Xue[1], **Mengxin Zheng**[1], **Heng Huang**[2], **Qian Lou**[1]
[1]LoopLeap Lab, University of Central Florida, [2]University of Maryland, College Park
{jiaqi.xue,mengxin.zheng,qian.lou}@ucf.edu heng@umd.edu

Project https://github.com/loopleap-lab/MultiTurnRoute

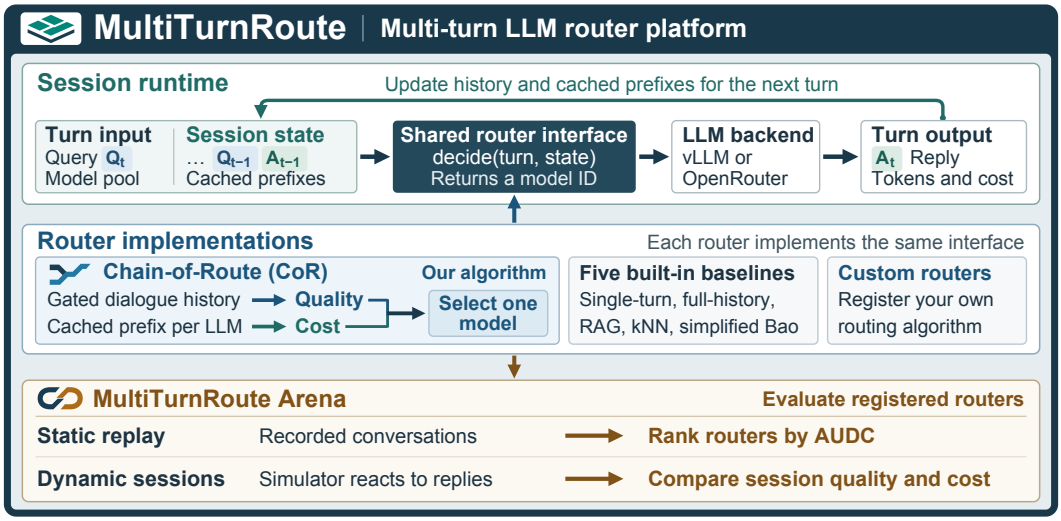


Figure 1: **MultiTurnRoute is a multi-turn LLM router platform.** A registered router reads the query and session state to select a model. The reply and token counts update the state for the next turn. The Arena ranks routers by area under the deferral curve (AUDC) in static replay and compares quality and cost in interactive sessions.

Abstract

In a multi-turn conversation, the choice of language model at one turn can affect which model should answer the next. The selected model's reply shapes the context for the next query, and its cached prompt prefix can reduce the cost of using that model again. We introduce MultiTurnRoute, a platform for building and evaluating multi-turn LLM routers. A shared interface gives built-in and user-defined routers the current query, dialogue history, and an estimate of each model's reusable prefix length. Our routing algorithm, Chain-of-Route (CoR), uses a learned gate over recent and recurrent dialogue history to predict response quality and estimates cost from each model's reusable prefix. MultiTurnRoute Arena ranks routers on recorded conversations and compares session quality and cost in interactive runs where a simulated user responds to each reply. On a benchmark drawn from WildChat and LMSYS-Chat-1M with per-turn quality annotations for five LLMs, CoR reduces the normalized cost required to match the strongest model's quality by up to 20.3% relative to the strongest routing baseline, while matching or improving peak response quality.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

A conversational assistant can draw on several large language models (LLMs), each with different capabilities and inference costs. A router decides which model should answer a given query. Most routers make this decision from the current input alone [1]. In an ongoing conversation [2], however, a short follow-up can depend on several earlier turns. In WildChat [3], a corpus of 1M real user interactions, 41% of conversations span multiple turns. Other conversation datasets show similar patterns [4, 5]. Topics and user goals can also shift within a conversation [6, 7], so the model best suited to one turn may be a poor choice for the next.

We introduce **MultiTurnRoute**, a platform for building and evaluating routers that select a model at each turn (Figure 1). Its session runtime exposes two kinds of state through a common router interface. The *dialogue state* helps a router interpret the query. The *serving state* tracks the prefix each candidate model is expected to reuse. The selected model's reply extends the dialogue, and its token usage updates the cache record for the next decision.

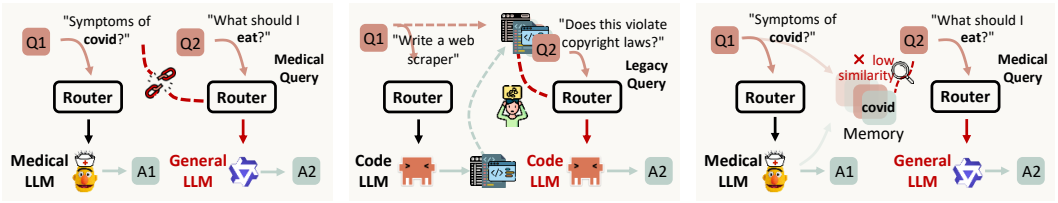


Figure 2: Challenges of adapting single-turn LLM routing to multi-turn conversations regarding dialogue state. (a) Routing using only the current query ignores conversational context. (b) Routing using the full context can let dominant prior content overshadow the current query's intent (*context overshadowing*). (c) Routing using selectively retrieved context misses implicit dependencies when users refer back without repeating keywords.

Which history matters for quality? After a discussion of COVID symptoms, "what should I eat?" asks for dietary advice during illness. A router that sees only the current query can miss this dependency (Figure 2a). Feeding it the full history introduces *context overshadowing* when, for example, a long code response dominates an unrelated follow-up (Figure 2b). Retrieving the most similar turns can also miss an implicit reference because the dietary question shares few words with the medical discussion (Figure 2c). The amount and kind of history needed therefore depend on the current query.

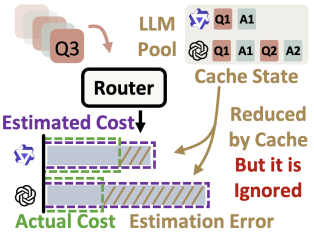


Figure 3: Ignoring serving state produces misleading cost rankings.

Which prefix matters for cost? Serving engines such as vLLM [8] and SGLang [9] can reuse a model's cached key–value (KV) states for a shared prompt prefix. For example, the cited Claude Sonnet prices are $3.00 per 1M input tokens and $0.30 per 1M cached tokens, a 90% reduction [10]. Each model may have answered a different subset of earlier turns and retained a different prefix. In Figure 3, switching to a model with a lower nominal price costs more than continuing with the model that already holds the session prefix.

Routing algorithm. Our router, **Chain-of-Route (CoR)**, estimates quality and cost separately for each candidate. Its Dialogue State Propagator (DSP) combines recent turns with a compact recurrent history and learns how much context to use for each query. Its Serving State Tracker (SST) uses each model's cached prefix to estimate the cost of answering. On WildChat and LMSYS-Chat-1M, CoR reduces Query-Normalized Cost (QNC), the minimum normalized cost required to match the strongest model's quality, by up to 20.3% relative to the strongest routing baseline, while matching or improving peak response quality.

Evaluating routers. CoR, five built-in baselines, and user-defined routers implement the same interface. **MultiTurnRoute Arena** uses that interface in two settings. Static replay compares choices on the same recorded turns and ranks routers by their quality–cost trade-off. In interactive sessions, a simulated user responds to the selected model's reply, so an early routing decision can change the questions a router faces later.

2 Background and Related Work

Multi-Turn LLM Conversations. Both conversational assistants and multi-agent systems make successive LLM calls over shared context [2, 11, 12]. WildChat [3], ShareGPT [4], and LMSYS-Chat-1M [5] document these interactions. Studies of topic shifts [6, 7, 13] and context dependence [14, 15] help explain why different turns may call for models with different capabilities and costs [1]. Existing conversation datasets [3–5, 16, 17] generally record a single model's replies in each dialogue. Routing benchmarks [18] provide quality labels for several models on individual queries. Evaluating per-turn routing requires both dialogue histories and ratings for multiple candidate models on each turn.

KV-Cache Management in LLM Serving. Successive prompts in a conversation share a prefix, allowing an LLM to reuse key–value states from earlier turns [19]. vLLM [8] manages these states with paged allocation, SGLang [9] supports automatic prefix caching, and CachedAttention [20] extends reuse by offloading states to host memory [21]. In the cited Claude Sonnet pricing, cached input tokens cost 10% of the standard input price [10]. Each routing decision affects which model accumulates reusable state. MultiTurnRoute tracks the reusable prefix length for each candidate, and CoR uses it to estimate cost.

LLM Routing. Single-turn routers differ in how they predict model quality [1, 18, 22, 23] and cost [24–28] (Appendix A). GMTRouter [29] learns user preferences across conversations and selects one LLM for a session. Bao et al. [30] extend RouteLLM [1] to choose a model at each turn from the full dialogue history. Their cost model distinguishes switching from staying with the last model. CoR estimates how much of the prompt each candidate retains, including a model used several turns earlier. Concurrent work, MTRouter [31], jointly encodes model embeddings and a token-limited interaction history for agentic tool use. Its cost estimate uses token counts and list prices. The authors also observe that switching models frequently can limit prompt-cache reuse.

3 The Chain-of-Route Algorithm

3.1 Overview: From Single-Turn to Multi-Turn Routing

Let $\mathcal{M} = \{m_1, m_2, \ldots, m_K\}$ denote a set of K candidate LLMs. At each turn t of a conversation, the user submits a query Q_t and the router selects a model m_t to generate the answer A_t. The router estimates each candidate's response quality $\hat{Q}$ and serving cost $\hat{C}$, then selects the model that maximizes the quality-cost tradeoff, where $\lambda \in [0, 1]$ controls cost sensitivity:

$$m_t = \arg\max_{m \in \mathcal{M}} \left[(1 - \lambda) \cdot \hat{Q} - \lambda \cdot \hat{C} \right]. \tag{1}$$

Equation 1 is the Lagrangian relaxation of maximizing response quality subject to a serving-cost budget, with λ acting as the multiplier: sweeping λ over $[0, 1]$ traces the quality-cost Pareto frontier, and our metrics in Section 4 summarize this entire frontier rather than any single operating point. In deployment, λ is a user-specified quality-cost preference, analogous to the effort levels exposed by commercial assistants. Following recent single-turn routers [24, 26], we estimate quality and cost separately. Decoupling the two estimates is natural here: KV-cache reuse avoids recomputing an identical prefix, so it changes serving cost but not the response. This lets SST compute the cache correction analytically while DSP focuses entirely on quality. Single-turn routers estimate $\hat{Q}$ and $\hat{C}$ based solely on the current query Q_t: $\hat{Q} = f_Q(m, Q_t)$ and $\hat{C} = f_C(m, Q_t)$. This does not account for two forms of state that accumulate across turns in multi-turn conversations: the *dialogue state*, i.e., the conversation history that shapes query meaning, and the *serving state*, i.e., the KV-cache retained by each model's serving engine that reduces serving cost.

We observe that each routing decision both *depends on* and *shapes* the state available to future decisions. On one hand, the router must consult both forms of state to make an informed choice: the dialogue state determines how to assess each model's quality on the current query, while the serving state determines each model's true serving cost. On the other hand, the routing outcome alters both states for subsequent turns: the selected model's answer extends the dialogue history, and its serving engine retains the KV-cache, reducing its serving cost if chosen again later. This means routing decisions are inherently chained, with state propagating from one turn to the next.

Chain-of-Route (CoR), our routing algorithm in MultiTurnRoute, makes this dependence across turns explicit. As illustrated in Figure 4, the router receives the current query Q_t along with two forms

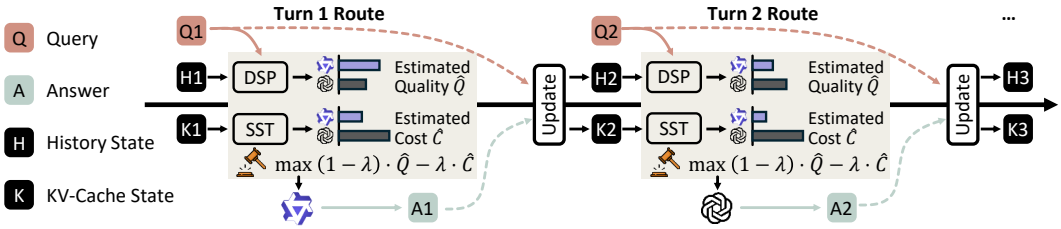


Figure 4: Overview of Chain-of-Route (CoR). At each turn, the router receives the current query Q_t, the dialogue state H_t, and the serving state $\mathcal{K}_t$. DSP conditions on H_t to estimate quality $\hat{Q}$, while SST conditions on $\mathcal{K}_t$ to estimate cost $\hat{C}$. The model maximizing $(1-\lambda)\cdot\hat{Q}-\lambda\cdot\hat{C}$ is selected, and both states are updated before propagating to the next turn.

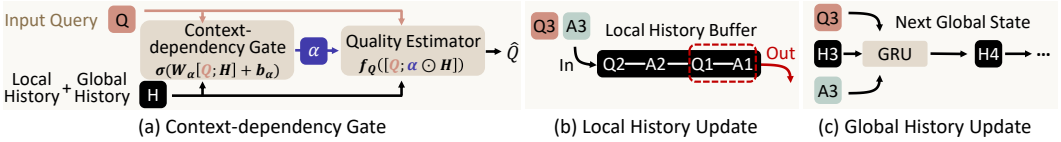


Figure 5: Overview of the Dialogue State Propagator (DSP). (a) Context-dependency Gate: query Q and dialogue state H produce a gate α_t, which scales H before the quality estimator f_Q. (b) Local History Update: a FIFO buffer (e.g., of size W=2) evicts the oldest pair (Q_1, A_1) upon each new arrival (Q_3, A_3). (c) Global History Update: a GRU updates the global state at each turn (e.g., $H_3 \rightarrow H_4$), accumulating conversation-level trends.

of accumulated state: the dialogue state H_t and the serving state $\mathcal{K}_t = \{L_{\text{cache}}^{m,t}\}_{m\in\mathcal{M}}$, where $L_{\text{cache}}^{m,t}$ records the cached token count of candidate model m. Two dedicated modules produce state-aware estimates: the Dialogue State Propagator (DSP) conditions on H_t to produce $\hat{Q} = f_Q(m, Q_t, H_t)$, and the Serving State Tracker (SST) conditions on $L_{\text{cache}}^{m,t}$ to produce $\hat{C} = f_C(m, Q_t, L_{\text{cache}}^{m,t})$. These estimates are plugged into Equation 1 for model selection. After the selected model generates its answer A_t, both states are updated: DSP incorporates the new query-answer pair to produce H_{t+1}, and SST updates each model's cache record to produce $\mathcal{K}_{t+1}$. The updated states carry forward to the next turn, forming a continuous route-then-update chain across the conversation. We detail DSP in Section 3.2 and SST in Section 3.3.

3.2 Dialogue State Propagator (DSP)

As illustrated in Figure 2, routing based solely on the current query is context-blind (a), feeding the full history introduces context overshadowing (b), and selectively retrieving relevant history misses implicit dependencies (c). Rather than conditioning on raw dialogue at each turn, DSP maintains a compact hidden state H_t that incrementally accumulates conversation-level information (e.g., topic evolution and cross-turn dependencies) through a learned recurrent update. This decouples the router from the raw dialogue while preserving the contextual signals necessary for accurate quality estimation.

DSP decomposes H_t into two complementary components: a *local history state* $H_t^\ell \in \mathbb{R}^d$ that captures fine-grained, recent context (e.g., the current topic and turn-level difficulty), and a *global history state* $H_t^g \in \mathbb{R}^d$ that captures coarse-grained, conversation-level trends (e.g., overall user expertise and long-term topic distribution). The concatenated state $H_t = [H_t^\ell; H_t^g]$ is used at each turn to inform the routing decision, then updated with the new query-answer pair before propagating to the next turn.

DSP maintains two states that are read *before* routing each turn and updated *only after* the selected model generates its answer. Concretely, when routing turn t, DSP consults the buffer $\mathcal{B}_{t-1}$ and global state H_t^g produced after turn $t-1$, computes a fresh local state H_t^ℓ from Q_t and $\mathcal{B}_{t-1}$, forms the dialogue state $H_t = [H_t^\ell; H_t^g]$, and feeds H_t together with Q_t to a context-dependency gate that controls how much history reaches the quality estimator. After A_t is generated, a frozen pretrained sentence encoder E (all-MiniLM-L6-v2 [32], d=384) produces d-dimensional embeddings of the new pair, which we denote $Q_t, A_t \in \mathbb{R}^d$ (overloading the textual symbols), and concatenates them as $x_t = [Q_t; A_t] \in \mathbb{R}^{2d}$ to advance both states to $\mathcal{B}_t$ and H_{t+1}^g. We use the convention that subscript t

on H_t denotes the state available at the start of turn t (i.e., before observing A_t). The remainder of this subsection details the three components in the order they execute.

Local History State. The local history state maintains a fixed-size FIFO buffer $\mathcal{B}_{t-1} = \{x_{t-W}, \ldots, x_{t-1}\}$ of the W most recent past turn embeddings, where W is the local-history window size and each $x_i = [Q_i; A_i]$ corresponds to a completed query-answer pair, i.e., Q and A in Figure 5 (b). When routing turn t, the local state is obtained via cross-attention, with the current query Q_t as the query and the buffered past turn embeddings as keys and values:

$$H_t^\ell = \mathrm{CrossAttn}(Q_t,\ \mathcal{B}_{t-1}) = \mathrm{softmax}\left(\frac{(Q_t W_Q^\ell)(\mathcal{B}_{t-1} W_K^\ell)^\top}{\sqrt{d}}\right)\mathcal{B}_{t-1} W_V^\ell, \tag{2}$$

where $W_Q^\ell \in \mathbb{R}^{d\times d}$ and $W_K^\ell, W_V^\ell \in \mathbb{R}^{2d\times d}$ are learned projections. After the selected model produces A_t, the buffer is advanced: $\mathcal{B}_t \leftarrow (\mathcal{B}_{t-1} \cup \{x_t\}) \setminus \{x_{t-W}\}$, with positional order preserved via position embeddings. This allows the router to selectively attend to the most relevant recent turns rather than treating all turns equally.

Global History State. The global history state employs a gated recurrent update to selectively accumulate conversation-level information across all turns, as illustrated in Figure 5 (c). When routing turn t, the router reads the current global state H_t^g (produced after turn $t-1$) directly. After the selected model generates A_t, two gates jointly determine how H_t^g is advanced to H_{t+1}^g using the new turn input $x_t = [Q_t; A_t]$. The *relevance gate* r_t assesses which dimensions of H_t^g are pertinent to the new turn, selectively filtering outdated information:

$$r_t = \sigma(W_r[H_t^g; x_t] + b_r). \tag{3}$$

The *retention gate* z_t controls the balance between preserving the accumulated state and incorporating new information, learning to make small updates for routine turns while allowing larger updates when the turn carries significant conversation-level signals (e.g., a shift in user expertise level):

$$z_t = \sigma(W_z[H_t^g; x_t] + b_z). \tag{4}$$

The candidate state $\tilde{H}_t$ is then computed by combining the relevance-filtered history with the new turn input:

$$\tilde{H}_t = \tanh(W_h[r_t \odot H_t^g; x_t] + b_h), \tag{5}$$

where σ denotes the sigmoid function and $\odot$ denotes element-wise multiplication. Finally, the global state is advanced as a weighted combination of the previous state and the candidate:

$$H_{t+1}^g = (1 - z_t) \odot H_t^g + z_t \odot \tilde{H}_t. \tag{6}$$

The global state is initialized as $H_1^g = \mathbf{0}$, and the local buffer $\mathcal{B}_0$ is initially empty, so at the first turn the router falls back to query-only routing.

Context-dependency Gate. Not all queries in a multi-turn conversation depend equally on prior context. Some are follow-ups that inherit meaning from earlier turns, while others introduce entirely new topics and can be routed independently. This suggests that the router should adaptively decide how much history to incorporate for each query, rather than applying a fixed strategy. With H_t^ℓ and H_t^g in hand, DSP forms the dialogue state $H_t = [H_t^\ell; H_t^g]$ and computes a *context-dependency gate* $\alpha_t \in [0, 1]$ that assesses whether the current query requires conversational context, as shown in Figure 5 (a):

$$\alpha_t = \sigma(W_\alpha[Q_t; H_t] + b_\alpha). \tag{7}$$

When $\alpha_t \approx 1$, the query is context-dependent and the router leverages the full dialogue state; when $\alpha_t \approx 0$, the query is independent and the router effectively reduces to single-turn routing based on Q_t alone. The quality estimator then incorporates context in proportion to this gate:

$$\hat{Q}(m, Q_t, H_t) = f_Q([Q_t; \alpha_t \odot H_t]), \tag{8}$$

where $[\cdot;\cdot]$ denotes concatenation and f_Q is a learned scoring function.

3.3 Serving State Tracker (SST)

As discussed earlier, existing routers estimate cost using a static formula: $C(m, Q_t) = c^m_{\text{prefill}} \cdot L^t_{\text{prefill}} + c^m_{\text{gen}} \cdot L^t_{\text{gen}}$, where c^m_{prefill} and c^m_{gen} are model m's per-token prices and L^t_{prefill}, L^t_{gen} are the prefilling and generation token counts. This does not account for KV-cache reuse: a model that has served

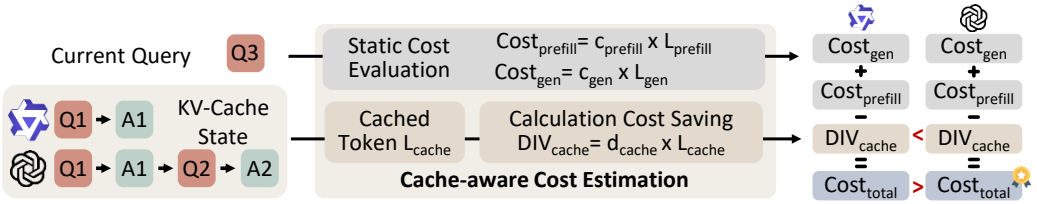


Figure 6: Overview of SST. Taking turn 3 as an example, Qwen caches (Q_1, A_1) while GPT caches (Q_1, A_1, Q_2, A_2). (Left) Each model's serving state and cached token count L_{cache}. (Middle) Static cost estimation yields a lower estimate for Qwen; cache dividend estimation computes each model's $\text{DIV}_{\text{cache}} = d_{\text{cache}} \times L_{\text{cache}}$. (Right) After subtracting $\text{DIV}_{\text{cache}}$, GPT's larger cache advantage reverses the ranking, making it the truly more cost-efficient option.

prior turns benefits from cache hits on previously seen tokens, but the static formula prices every token at the full rate. Since each candidate may have served different prior turns, the true serving cost varies across models. SST addresses this by introducing a *cache dividend* correction to the static cost estimate. In short, SST prices each candidate by how many context tokens it already holds in its KV-cache: cached tokens are charged at the cached-input rate and only the rest at the full prefill rate. We illustrate SST with a concrete example in Figure 6. Qwen answered turn 1. GPT received that exchange as part of its prompt for turn 2 and generated A_2. When Q_3 arrives at t=3, Qwen can therefore reuse (Q_1, A_1) and GPT can reuse the longer prefix (Q_1, A_1, Q_2, A_2), assuming both caches persist.

Serving State Tracking. SST maintains the cache state $L_{\text{cache}}^{m,t}$ for each candidate model m, defined as the number of tokens for which model m currently retains KV-cache at turn t. An instrumented serving engine can expose the retained prefix directly. With provider APIs, MultiTurnRoute estimates it from the prompt and output token counts on the most recent turn served by m, capped at the current prompt length. Eviction between calls can shorten the actual prefix. The backend's reported cache hits determine the realized input cost when available (Appendix H). In the running example of Figure 6 (left), Qwen holds $L_{\text{cache}}^{\text{Qwen},3} = |Q_1, A_1|$ while GPT holds $L_{\text{cache}}^{\text{GPT},3} = |Q_1, A_1, Q_2, A_2|$, reflecting their different serving histories.

Static Cost Estimation. As shown in Figure 6 (middle, top), a state-unaware router prices each candidate by the same static formula based on the full context length L_{prefill}^t: a prefilling cost $\text{Cost}_{\text{prefill}} = c_{\text{prefill}}^m \cdot L_{\text{prefill}}^t$ and a generation cost $\text{Cost}_{\text{gen}} = c_{\text{gen}}^m \cdot L_{\text{gen}}^t$, following the static cost formulation adopted by single-query routing work [26]. Because per-token prices c_{prefill}^m and c_{gen}^m differ across candidates, the static estimate already produces a per-model ranking. In our running example, Qwen has lower per-token prices than GPT, so its static estimate is the smaller of the two. Crucially, this ranking is computed as if every candidate must prefill the entire conversation from scratch, ignoring how much of the context each candidate has already cached.

Cache Dividend Estimation. SST corrects the static estimate by computing the *cache dividend* $\text{DIV}_{\text{cache}}$, the savings from reusing cached KV states, shown in Figure 6 (middle, bottom). With per-token cache discount $d_{\text{cache}}^m = c_{\text{prefill}}^m - c_{\text{cached}}^m$ (where c_{cached}^m is the cached-input price set by the provider for black-box APIs, or ≈ 0 for self-hosted models), the dividend for candidate m is $\text{DIV}_{\text{cache}}(m) = d_{\text{cache}}^m \cdot L_{\text{cache}}^{m,t}$. Subtracting it from the static estimate yields the cache-aware cost:

$$\hat{C}(m, Q_t, L_{\text{cache}}^{m,t}) = \overbrace{c_{\text{prefill}}^m \cdot L_{\text{prefill}}^t}^{\text{Cost}_{\text{prefill}}} - \overbrace{d_{\text{cache}}^m \cdot L_{\text{cache}}^{m,t}}^{\text{DIV}_{\text{cache}}} + \overbrace{c_{\text{gen}}^m \cdot L_{\text{gen}}^t}^{\text{Cost}_{\text{gen}}}. \tag{9}$$

As illustrated in Figure 6 (right), although Qwen has a lower static estimate than GPT, GPT enjoys a much larger $\text{DIV}_{\text{cache}}$ because it retains a longer prefix. Subtracting the dividend reverses the cost ranking in this example. A state-unaware router would prefer Qwen based on the static estimate and miss this opportunity.

By integrating DSP and SST, the final routing decision at turn t reduces to Equation 1, where $\hat{Q}$ is the dialogue-state-conditioned quality from DSP and $\hat{C}$ is the serving-state-aware cost from SST, jointly optimizing quality and cost under realistic multi-turn serving conditions.

4 Evaluation of Chain-of-Route

4.1 Experimental Setup

Model pool. We evaluate CoR and the comparison methods on a shared pool of $K{=}5$ models. The Arena protocols are described separately in Section 5. The experimental pool comprises Qwen3-0.6B, Qwen3-8B, Qwen3-14B, Qwen3-32B, and Qwen3-Next-80B-A3B-Instruct. We take each model's full-prefill price c^m_{prefill} and generation price c^m_{gen} from publicly listed OpenRouter pricing, and set the cached-input price as $c^m_{\text{cached}} = 0.2\, c^m_{\text{prefill}}$ following Alibaba Cloud Model Studio's context-cache convention.[1] The full per-model pricing table is in Appendix C.

Dataset. Existing multi-turn dialogue benchmarks [5, 16, 17] are single-model and do not provide per-turn multi-model quality scores; we therefore construct evaluation data from WildChat [3] and LMSYS-Chat-1M [5]. From each corpus we randomly sample 20,000 English conversations ($\sim$80,000 turns) with ≥ 2 turns per conversation, and split 80/20 by conversation hash.

To train the quality predictor, we need per-(turn, candidate) quality labels. We obtain them by having every candidate model $m \in \mathcal{M}$ generate a response for every training turn t under the *same* dialogue history, using the dataset's original responses for prior turns:

$$\text{context}_{\text{train}}(t) = \{(Q_1, A_1^{\text{orig}}), \ldots, (Q_{t-1}, A_{t-1}^{\text{orig}}), Q_t\}. \tag{10}$$

Each candidate response is independently scored on a 1–10 scale by DeepSeek-V3.1, selected from a panel of candidate judges via meta-evaluation against human ratings [17] (details in Appendix B). We adopt this shared-context construction as a tractable offline protocol: faithfully enumerating all routing trajectories scales as $|\mathcal{M}|^T$ in the conversation length, which is intractable. These quality scores supervise DSP. A separate head predicts output length from observed response lengths. SST combines this prediction with the tracked prefix length and token prices to estimate cost (Appendix D).

At evaluation, each routing strategy follows one trajectory per held-out conversation. For each user query in that conversation, CoR uses DSP and SST to select a model under Equation 1. Each comparison method uses its own decision rule. The selected model generates a reply, which DeepSeek-V3.1 judges and which becomes part of the history for the next query.

Baseline Methods. *Random* selects uniformly per turn. Single-turn routers *RouteLLM* [1], *CAR-ROT* [26], and *CSCR* [25] route each turn from the current query alone. Multi-turn routers *Bao et al.* [30] (full-history input + binary switch penalty) and *GMTRouter* [29] (cross-conversation user graph) are the closest competitors. The *Myopic Oracle* in Figure 7 (a) is a per-turn hindsight oracle: along its own trajectory, it picks the candidate maximizing Equation 1 using judge-assigned quality and realized cost under the current states, without branching over future decisions ($O(KT)$ per λ, not $O(K^T)$). All learned routers share *all-MiniLM-L6-v2* [32] as the text encoder; training hyperparameters, λ grid, and baseline adaptations are in Appendix D.

Evaluation Metrics. We evaluate each routing strategy using a *deferral curve* [25, 33], which plots the average per-turn response quality against the average normalized cost. Sweeping the cost-sensitivity parameter λ over the interval $[0, 1]$ (Equation 1) traces the deferral curve. Following [33], we employ three evaluation metrics: (i) Area Under the Deferral Curve (AUDC), measuring the overall quality-cost trade-off; (ii) Peak Quality, the maximum quality achievable by the router; and (iii) Query-Normalized Cost (QNC), the minimum normalized cost required to match the quality of the most capable LLM in the pool. Unless noted otherwise, ablations are run on WildChat with the default configuration; a sensitivity analysis over CoR's main design choices (local-history window, global state architecture, text encoder, cache discount) is provided in Appendix F.

4.2 Results

Overall Quality-Cost Tradeoff. Table 1 shows that CoR delivers the best quality-cost trade-off on both datasets. Single-turn routers miss both the dialogue state and the serving state. GMTRouter targets cross-conversation personalization (*who* asks rather than *what context* accumulates) and lands near the single-turn cluster. Bao et al., the strongest baseline, feeds full history into a RouteLLM-style router and reduces serving cost to a binary switching penalty; CoR pulls ahead by maintaining a

[1] `https://www.alibabacloud.com/help/en/model-studio/context-cache`

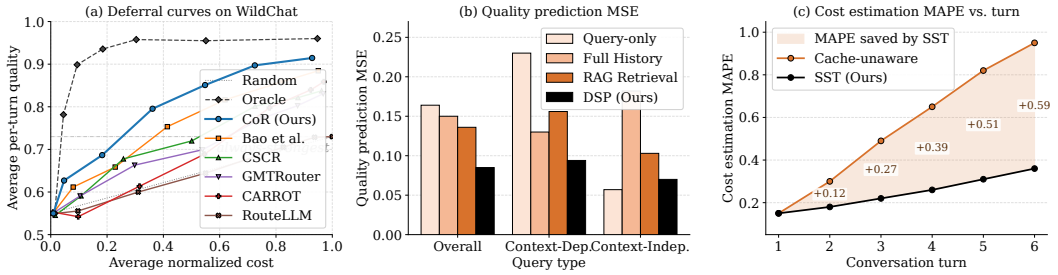


Figure 7: (a) Deferral curves on WildChat as the cost-sensitivity parameter λ is swept; "Oracle" is the per-turn Myopic Oracle defined in Section 4. (b) Quality prediction MSE for four history strategies, broken down by query type. (c) Cost estimation MAPE as a function of conversation turn for cache-unaware estimation vs. SST.

compact dialogue state and tracking per-model cache. The QNC gap is the largest, as SST exploits cache reuse that baselines do not plan for.

Table 1: Quality-cost tradeoff on WildChat and LMSYS-Chat-1M.

Method	WildChat			LMSYS-Chat-1M		
	AUDC ↑	QNC ↓	Peak ↑	AUDC ↑	QNC ↓	Peak ↑
Random	0.49	0.97	0.87	0.55	0.93	0.78
RouteLLM [1]	0.56	0.98	0.94	0.62	0.93	0.83
CARROT [26]	0.59	0.95	0.94	0.65	0.91	0.83
CSCR [25]	0.68	0.85	0.94	0.65	0.87	0.83
GMTRouter [29]	0.65	0.87	0.94	0.62	0.89	0.84
Bao et al. [30]	0.70	0.74	0.95	0.67	0.78	0.85
CoR (Ours)	0.74	0.59	0.95	0.70	0.65	0.87

Figure 7 (a) plots the full deferral curve on WildChat as λ is swept over $[0, 1]$. CoR Pareto-dominates all baselines across most of the cost range, with the gap widest at moderate cost budgets where DSP's context-aware selection and SST's cache-aware costing combine to extract the largest savings.

Effect of Dialogue State on Quality Prediction. To verify that DSP's recurrent dialogue state improves quality prediction, we compare it against three history-consumption strategies (i.e., Query-only, Full History, and RAG Retrieval) on two kinds of queries: *context-dependent* (where removing all prior turns drops the gold quality score by at least 3 points on the 1–10 scale, a $\geq 30\%$ relative drop) and *context-independent* (the rest). Under this partition, 58.7% of held-out WildChat turns and 54.2% of LMSYS-Chat-1M turns are context-dependent, so dialogue state materially affects quality on the majority of turns. As shown in Figure 7 (b), DSP outperforms all baselines on context-dependent queries. Query-only is the weakest on this slice, matching the context-blind failure in Figure 2 (a), while Full History and RAG Retrieval also underperform, consistent with context overshadowing (b) and implicit dependencies (c). On context-independent queries, DSP and Query-only perform comparably, so DSP does not inject harmful signals when history is unnecessary.

Although never supervised with this partition, the gate fires strongly on context-dependent queries (mean $\alpha_t \approx 0.78$) and stays low on context-independent ones (≈ 0.28; Table 2), so DSP learns to modulate history use rather than uniformly mixing it in.

Table 2: Context-dependency gate α_t behavior.

Query type	mean α_t
Context-dependent	0.78
Context-independent	0.28

Effect of Serving State on Cost Estimation. Figure 7 (c) shows cost estimation mean absolute percentage error (MAPE) versus conversation turn. At turn 1, both methods start with the same MAPE (~ 0.15); this floor reflects unavoidable output-length prediction uncertainty (both methods predict generation tokens before observing them), and any cost estimator inherits it. As turns increase, cache-unaware estimation grows steeply and approaches ~ 0.95 by turn 6 because it prices every token at the full prefill rate and ignores cached reuse. SST grows only mildly with turn depth (the residual error reflects imperfect cache-state tracking under realistic serving variability) and stays within a narrow band well below cache-unaware. The widening gap shows that cache-unaware estimation

increasingly overestimates cost in longer conversations, biasing routing toward unnecessary switches, while SST keeps the bias bounded and enables more efficient routing.

Table 3: Quality-cost performance stratified by conversation length on WildChat. Buckets group test conversations by total turn count. CoR's gain over the strongest single-turn baseline (CSCR) widens as conversations grow longer, consistent with SST exploiting more accumulated cache.

Turn bucket	#conv.	AUDC ↑		QNC ↓		Peak ↑	
		CSCR	CoR	CSCR	CoR	CSCR	CoR
2 turns	∼1,680	0.69	0.71	0.86	0.65	0.96	0.94
3–4 turns	∼1,240	0.67	0.74	0.85	0.58	0.94	0.95
5–7 turns	∼ 700	0.65	0.75	0.88	0.53	0.92	0.96
8+ turns	∼ 380	0.63	0.78	0.92	0.45	0.90	0.97
Δ (8+ vs 2)	—	−0.06	+0.07	+0.06	−0.20	−0.06	+0.03

The downstream effect appears in Table 3. As conversation length increases, CoR's AUDC rises from 0.71 (2 turns) to 0.78 (8+ turns), while CSCR's falls from 0.69 to 0.63. The QNC gap widens from 0.21 to 0.47. This divergence indicates that SST benefits from cache accumulation: longer conversations provide more reuse, whereas cache-unaware methods repeatedly incur full prefill cost.

Cross-dataset transfer. Training on WildChat and evaluating on LMSYS yields AUDC 0.68, and the reverse direction (LMSYS $\rightarrow$ WildChat) gives 0.71, only 0.02–0.03 below in-domain results. This suggests DSP learns transferable conversation-level signals rather than dataset-specific features.

Candidate-pool composition. Varying pool size $K \in \{2, 3, 4, 5, 6\}$ and replacing Qwen3 members with Llama-3.1 and Mistral models in a cross-family ablation, CoR remains the best method in all settings, and cross-family performance matches the same-family pool, indicating no reliance on family-specific shortcuts. See Appendix G.

Additional analyses. CoR's gains hold under cross-family judges, dialogue-length and topic shift, and real commercial billing (Appendices I.1, I.3, and I.4).

5 The MultiTurnRoute Platform

5.1 Session Runtime and Router Interface

MultiTurnRoute connects a conversational application to a pool of LLMs (Figure 1). At each turn a registered router reads the current query, dialogue history, and the estimated reusable prefix length for each candidate. It selects one model, and the runtime dispatches the request to its backend. The backend (vLLM [8] or a provider API) returns the reply and reports billed and cached tokens when available. The runtime records the exchange and updates the session state. Applications can use an OpenAI-compatible endpoint, including deployments behind gateways such as vLLM Semantic Router [34].

CoR and five built-in baselines (single-turn, full-history, RAG, kNN, and simplified Bao et al.) implement `decide(turn, state)`. A researcher can register another router with that interface and use the same implementation in the runtime and Arena. The router can use either part of the state. For example, a cache-aware policy can compare the cost of returning to a previously used model with that of continuing on the current model. A history-based policy can use the same input to estimate which model will answer a follow-up well. The runtime executes either choice and accounts for the realized cost using the backend's input, cached, and output token counts. This lets researchers change the decision rule while using the same backend and accounting code. The implementation is available at `https://github.com/loopleap-lab/MultiTurnRoute`.

5.2 MultiTurnRoute Arena

Single-turn arenas such as RouterArena [35] can look up a pre-scored answer for each query. In a conversation, a model's reply can also change what the user asks next. MultiTurnRoute Arena measures these effects in two tracks.

Static replay. The static track uses 3,996 conversations drawn from the benchmark in Section 4. A separate pool of nine models answered every recorded turn in its original context, and a judge rated

each answer. On the held-out split, the Arena looks up the rating of the model selected by the router. All routers see the same user queries and recorded replies. Their costs can still diverge because each router's sequence of model choices determines which prefixes its models can reuse. The Arena holds the model pool, prices, and cost weights fixed and ranks routers by area under the deferral curve (AUDC). The leaderboard therefore compares policies within this pool. The main experiments in Section 4 use their own model pool and quality annotations.

Interactive sessions. The dynamic track starts routers from the same task. A simulated user reads the selected model's reply and writes the next question or ends the conversation. An answer that requests clarification, for example, can lead to a different sequence of later turns than an answer that completes the task immediately. A judge rates each answer in its own dialogue context, and the Arena computes cost from the token usage reported by the backend. Routers run on matched tasks with the same model pool and user simulator. For each cost weight, the Arena averages answer quality and normalized cost over rated turns and sums the billed cost across all turns.

6 Conclusion

MultiTurnRoute gives researchers a common interface for building multi-turn LLM routers and comparing their decisions on recorded and interactive conversations. Our routing algorithm, Chain-of-Route, uses DSP to estimate response quality from dialogue history and SST to estimate cost from each model's cached prefix. On WildChat and LMSYS-Chat-1M, CoR reduces QNC by up to 20.3% relative to the strongest baseline while matching or improving peak response quality. Ablations show that both modules contribute. MultiTurnRoute Arena lets researchers compare other routing policies under the same conditions and examine how a model choice affects later turns.

Acknowledgements

This work was supported in part by the National Science Foundation (NSF) under Award No. 2523407. H.H. was partially supported by NSF IIS 2347592, 2348169, DBI 2405416, CCF 2348306, CNS 2347617, RISE 2536663. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *The Thirteenth International Conference on Learning Representations*.

[2] Xi Shi, Mengxin Zheng, and Qian Lou. Learning latency-aware orchestration for parallel multi-agent systems. *arXiv preprint arXiv:2601.10560*, 2026.

[3] Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. *arXiv preprint arXiv:2405.01470*, 2024.

[4] ShareGPT Contributors. ShareGPT dataset. `https://sharegpt.com`, 2023. Accessed: 2024-01-01.

[5] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric P Xing, et al. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. *arXiv preprint arXiv:2309.11998*, 2023.

[6] Yi Xu, Hai Zhao, and Zhuosheng Zhang. Topic-aware multi-turn dialogue modeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14176–14184, 2021.

[7] Hainan Zhang, Yanyan Lan, Liang Pang, Hongshen Chen, Zhuoye Ding, and Dawei Yin. Modeling topical relevance for multi-turn dialogue generation. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 3737–3743. International Joint Conferences on Artificial Intelligence Organization, 2020.

[8] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

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

[10] Anthropic. Model pricing. `https://platform.claude.com/docs/en/about-claude/pricing`, 2025. Accessed: 2025-07-01.

[11] Jiaqi Xue, Yanjun Wang, Xiangci Li, Lingbo Mo, Aritra Sengupta, Shweta Garg, Murali Krishna Ramanathan, and Myeongsoo Kim. Aecp: Artifact-exclusive communication protocol for multi-agent code generation. *arXiv preprint arXiv:2610.06481*, 2026.

[12] Rakibul Hasan Rajib, Mengxin Zheng, and Qian Lou. Learning what to retain: Gated-memory routing for efficient collaboration in multi-agent LLM systems. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026.

[13] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[14] Hui Su, Xiaoyu Shen, Rongzhi Zhang, Fei Sun, Pengwei Hu, Cheng Niu, and Jie Zhou. Improving multi-turn dialogue modelling with utterance ReWriter. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 22–31, Florence, Italy, July 2019. Association for Computational Linguistics.

[15] Yu Wu, Wei Wu, Chen Xing, Can Xu, Zhoujun Li, and Ming Zhou. A sequential matching framework for multi-turn response selection in retrieval-based chatbots. *Computational Linguistics*, 45(1):163–197, March 2019.

[16] Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, et al. Mt-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7421–7454, 2024.

[17] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

[18] 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.

[19] Xi Shi, Mengxin Zheng, and Qian Lou. KVShareArena: KV-cache reuse across contexts and model checkpoints. *arXiv preprint arXiv:2609.10266*, 2026.

[20] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. {Cost-Efficient} large language model serving for multi-turn conversations with {CachedAttention}. In *2024 USENIX annual technical conference (USENIX ATC 24)*, pages 111–126, 2024.

[21] Rakibul Hasan Rajib, Mengxin Zheng, and Qian Lou. AgentServeSim: Serving-system simulation and policy search for LLM agent programs. *arXiv preprint arXiv:2606.09613*, 2026.

[22] Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. *arXiv preprint arXiv:2410.03834*, 2024.

[23] Jiaqi Xue, Qian Lou, Jiarong Xing, and Heng Huang. R2-router: A new paradigm for llm routing with reasoning. *ArXiv*, abs/2602.02823, 2026.

[24] 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)*, pages 15629–15644, 2025.

[25] Reza Shirkavand, Shangqian Gao, Peiran Yu, and Heng Huang. Cost-aware contrastive routing for llms. *arXiv preprint arXiv:2508.12491*, 2025.

[26] Seamus Somerstep, Felipe Maia Polo, Allysson Flavio Melo de Oliveira, Prattyush Mangal, Mírian Silva, Onkar Bhardwaj, Mikhail Yurochkin, and Subha Maity. Carrot: A cost aware rate optimal router. *arXiv preprint arXiv:2502.03261*, 2025.

[27] Ahasan Kabir, Jiaqi Xue, Mengxin Zheng, and Qian Lou. Hw-router: Hardware-aware routing for scalable multi-llm serving. In *Proceedings of the 63rd Design Automation Conference (DAC)*, 2026.

[28] Ahasan Kabir, Jiaqi Xue, Mengxin Zheng, and Qian Lou. INFRAMIND: Infrastructure-aware multi-agent orchestration. *arXiv preprint arXiv:2606.11440*, 2026.

[29] Encheng Xie, Yihang Sun, Tao Feng, and Jiaxuan You. Gmtrouter: Personalized llm router over multi-turn user interactions. *ArXiv*, abs/2511.08590, 2025.

[30] Rui Bao, Nan Xue, Yaping Sun, and Zhiyong Chen. Dynamic quality-latency aware routing for llm inference in wireless edge-device networks. In *2025 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, pages 1–6. IEEE, 2025.

[31] 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 (ACL)*, 2026. arXiv:2604.23530.

[32] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *ArXiv*, abs/1908.10084, 2019.

[33] Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, et al. Universal model routing for efficient llm inference. *arXiv preprint arXiv:2502.08773*, 2025.

[34] vLLM Semantic Router Team. vllm semantic router. `https://github.com/vllm-project/semantic-router`, 2025.

[35] Yifan Lu, Rixin Liu, Jiayi Yuan, Xingqi Cui, Shenrun Zhang, Hongyi Liu, and Jiarong Xing. Routerarena: An open platform for comprehensive comparison of LLM routers. In *The Fourteenth International Conference on Learning Representations*, 2026.

[36] Hugging Face Router. Llm router – chat ui configuration. `https://huggingface.co/docs/chat-ui/configuration/llm-router`, 2026. Online; accessed 26 Jan 2026.

[37] OpenAI. Gpt-5 system card. `https://cdn.openai.com/gpt-5-system-card.pdf`, August 2025. Accessed: Sep 2025.

[38] Microsoft. Model router for microsoft foundry. `https://learn.microsoft.com/en-us/azure/ai-foundry/openai/concepts/model-router`, 2025. Accessed: Sep 2025.

[39] 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, 2024.

[40] Marija Šakota, Maxime Peyrard, and Robert West. Fly-swat or cannon? cost-effective language model choice via meta-modeling. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 606–615, 2024.

[41] Dimitris Stripelis, Zhaozhuo Xu, Zijian Hu, Alay Dilipbhai Shah, Han Jin, Yuhang Yao, Jipeng Zhang, Tong Zhang, Salman Avestimehr, and Chaoyang He. Tensoropera router: A multi-model router for efficient llm inference. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 452–462, 2024.

[42] Fangzhou Wu and Sandeep Silwal. Efficient training-free online routing for high-volume multi-llm serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

[43] Quang H Nguyen, Thinh Dao, Duy C Hoang, Juliette Decugis, Saurav Manchanda, Nitesh V Chawla, and Khoa D Doan. Metallm: A high-performant and cost-efficient dynamic framework for wrapping llms. *arXiv preprint arXiv:2407.10834*, 2024.

[44] Co Tran, Salman Paracha, Amina Hafeez, and Shuguang Chen. Arch-router: Aligning llm routing with human preferences. *ArXiv*, abs/2506.16655, 2025.

[45] Clovis Varangot-Reille, Christophe Bouvard, and Antoine Gourru. Generalising llm routing using past performance retrieval: A few-shot router is sufficient. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 304–319, 2026.

[46] Jiaqi Xue, Mengxin Zheng, Ting Hua, Yilin Shen, Yepeng Liu, Ladislau Bölöni, and Qian Lou. Trojllm: A black-box trojan prompt attack on large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[47] Mengxin Zheng, Jiaqi Xue, Xun Chen, Yanshan Wang, Qian Lou, and Lei Jiang. Trojfsp: Trojan insertion in few-shot prompt tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024.

[48] Mengxin Zheng, Jiaqi Xue, Zihao Wang, Xun Chen, Qian Lou, Lei Jiang, and Xiaofeng Wang. Ssl-cleanse: Trojan detection and mitigation in self-supervised learning. In *European Conference on Computer Vision (ECCV)*, 2024.

[49] Jiaqi Xue, Qian Lou, and Mengxin Zheng. Badfair: Backdoored fairness attacks with group-conditioned triggers. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8257–8270, 2024.

[50] Jiaqi Xue, Mengxin Zheng, Yi Sheng, Lei Yang, Qian Lou, and Lei Jiang. Trojfair: Trojan fairness attacks. In *Proceedings of the 1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis (LAMPS)*, 2024.

[51] Jiaqi Xue, Mengxin Zheng, Yebowen Hu, Fei Liu, Xun Chen, and Qian Lou. Badrag: Identifying vulnerabilities in retrieval augmented generation of large language models. *arXiv preprint arXiv:2406.00083*, 2024.

[52] Qian Lou, Xin Liang, Jiaqi Xue, Yancheng Zhang, Rui Xie, and Mengxin Zheng. Cr-utp: Certified robustness against universal text perturbations on large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

[53] Jiaqi Xue, Yifei Zhao, Mansour Al Ghanim, Shangqian Gao, Ruimin Sun, Qian Lou, and Mengxin Zheng. Pro: Enabling precise and robust text watermark for open-source llms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2026.

[54] Mansour Al Ghanim, Jiaqi Xue, Rochana Prih Hastuti, Mengxin Zheng, Yan Solihin, and Qian Lou. Evaluating the robustness and accuracy of text watermarking under real-world cross-lingual manipulations. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 7396–7416, Suzhou, China, November 2025. Association for Computational Linguistics.

[55] Yancheng Zhang, Jiaqi Xue, Mengxin Zheng, Mimi Xie, Mingzhe Zhang, Lei Jiang, and Qian Lou. Cipherprune: Efficient and scalable private transformer inference. In *International Conference on Learning Representations (ICLR)*, 2025.

[56] Jiaqi Xue, Mengxin Zheng, and Qian Lou. Robpi: Robust private inference against malicious client. In *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2026.

[57] Jiaqi Xue, Mayank Kumar, Yuzhang Shang, Shangqian Gao, Rui Ning, Mengxin Zheng, Xiaoqian Jiang, and Qian Lou. Dictpfl: Efficient and private federated learning on encrypted gradients. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[58] Muhammad Husni Santriaji, Jiaqi Xue, Yancheng Zhang, Qian Lou, and Yan Solihin. Dataseal: Ensuring the verifiability of private computation on encrypted data. In *IEEE Symposium on Security and Privacy (S&P)*, 2025.

[59] Jiaqi Xue, Yancheng Zhang, Yanshan Wang, Xueqiang Wang, Hao Zheng, and Qian Lou. Cryptotrain: Fast secure training on encrypted dataset. In *Proceedings of the 1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis (LAMPS)*, 2024.

[60] Jiaqi Xue, Xin Xin, Wei Zhang, Mengxin Zheng, Qianqian Song, Minxuan Zhou, Yushun Dong, Dongjie Wang, Xun Chen, Jiafeng Xie, Liqiang Wang, David Mohaisen, Hongyi Wu, and Qian Lou. Sok: Can fully homomorphic encryption support general ai computation? a functional and cost analysis. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2026.

[61] Ardhi Wiratama Baskara Yudha, Jiaqi Xue, Qian Lou, Huiyang Zhou, and Yan Solihin. Boostcom: Towards efficient universal fully homomorphic encryption by boosting the word-wise comparisons. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2024.

[62] Mayank Kumar, Jiaqi Xue, Mengxin Zheng, and Qian Lou. Fhe-coder: Secure agentic code generation for fully homomorphic encryption. In *International Conference on Learning Representations (ICLR)*, 2026.

[63] Jiaqi Xue, Yifei Zhao, Mengxin Zheng, Fan Yao, Yan Solihin, and Qian Lou. Securing transformer-based ai execution via unified tees and crypto-protected accelerators. *arXiv preprint arXiv:2507.03278*, 2025.

[64] Yukuan Zhang, Mengxin Zheng, and Qian Lou. SecureRouter: Encrypted routing for efficient secure inference. In *Proceedings of the 63rd Design Automation Conference (DAC)*, 2026.

[65] Seanie Lee, Dong Bok Lee, Dominik Wagner, Minki Kang, Haebin Seong, Tobias Bocklet, Juho Lee, and Sung Ju Hwang. Saferoute: Adaptive model selection for efficient and accurate safety guardrails in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025.

A Survey of Single-Turn LLM Routers

LLM routing has been widely adopted in open-source platforms [34, 36] and commercial deployments such as OpenAI's GPT-5 [37] and Azure AI Foundry [38]. Existing routers vary primarily in how they predict per-candidate quality and cost, but uniformly treat each query in isolation.

Quality predictors. Lightweight models are trained to estimate each candidate's response quality via neural networks [1, 39, 40], k-nearest neighbors [18, 41, 42], and graph neural networks [22].

Cost predictors. Early routers rely on static attributes such as per-token pricing [24, 25], while more recent work incorporates output-length prediction [26, 42, 43] for more accurate per-query cost estimates.

Complementary routing signals. A separate line of work introduces complementary signals into the routing decision while still routing each query independently: Arch-Router [44] aligns routing with human preferences over latency-quality trade-offs, R2-Router [23] introduces reasoning into the routing decision, HW-Router [27] and INFRAMIND [28] incorporate real-time hardware and serving-system signals such as queue lengths, KV-cache usage, and recent latency to meet latency SLOs, and past-performance retrieval [45] retrieves past performance records to generalize routing to unseen models.

Safety-aware routing. Beyond quality and cost, foundation models raise safety and security concerns. They are vulnerable to trojan and backdoor attacks that hijack their outputs through hidden triggers [46–48] or bias their predictions against specific groups [49, 50], to poisoned retrieval corpora in retrieval-augmented generation [51], and to adversarial text perturbations [52]. Tracing the provenance of model-generated text calls for robust watermarking [53, 54], and protecting user data during training and inference motivates cryptographic privacy-preserving computation [55–62] and trusted execution environments [63]. Routing offers a natural point at which to enforce such safeguards [64]: SafeRoute [65] routes each input between a small and a large safety guard model, invoking the large model only on hard inputs to keep guardrail accuracy high at lower cost. Like the routers above, it decides for each input independently.

None of the above account for dialogue or serving state across turns; CoR explicitly and jointly models both. We compare against Bao et al. [30], which routes each turn of a conversation, as a baseline.

B Dataset Construction Details

Why existing multi-turn benchmarks are inadequate. MT-Bench [17] contains only 80 two-turn dialogues, each completed by a single LLM. MT-Bench-101 [16] scales to 1,388 multi-turn dialogues but remains single-model. LMSYS Chatbot Arena [5] provides large-scale human preferences at the conversation level, not per-turn quality scores. None of these provides what per-turn routing requires: for each turn t, quality scores $Q(m, Q_t)$ from *multiple* candidate models under the *same* dialogue context.

Preprocessing. For both WildChat and LMSYS-Chat-1M we apply identical preprocessing: filter to English conversations with ≥ 2 turns and ≥ 10 characters per user query, then randomly sample 20,000 conversations ($\sim$80,000 turns) per corpus.

Shared-context protocol for training-label construction. The shared-context design used to build the per-(turn, candidate) training labels ensures: (1) fair comparison—all candidate models see identical context at each turn, so quality differences reflect model capability alone; (2) tractability—it yields K comparable labels per turn in $O(KT)$ generation and judge calls, without depending on a particular routing policy; and (3) extensibility—adding a new model requires only generating and scoring its responses without regenerating existing data. This shared-context construction is restricted to label generation: at evaluation time we run auto-regressive routing, where each routing strategy traces a single trajectory per held-out conversation and the routed model's actual response is fed back as context for the next turn, faithfully reflecting deployment conditions including any response-distribution shift induced by routing-driven model switches.

LLM judge selection. To select a reliable LLM judge, we followed the LLM-as-a-judge validation protocol [17]: we randomly sampled 500 candidate responses, and recruited a pool of 30 expert

annotators with each response receiving independent ratings from 3 annotators (yielding $1{,}500$ human ratings whose mean is the gold label). We then evaluated 4 candidate judges (DeepSeek-V3.1, Qwen3-235B-A22B, GPT-5-mini, and Llama-3.3-70B-Instruct) by computing their Pearson correlation with the gold labels. Among them, DeepSeek-V3.1 achieved the highest correlation ($\rho = 0.84$) and was selected as the final judge. This LLM-as-a-judge approach is crucial for evaluating open-ended dialogue queries, where traditional automatic evaluation methods such as exact match fail to capture response quality.

Scoring details. Each model's response at each turn is independently scored on a 1–10 scale. We use individual (non-pairwise) scoring for scalability (K evaluations per turn vs. $\binom{K}{2}$ for pairwise) and compatibility with our quality predictor $\hat{Q}$, which estimates absolute quality rather than relative preference. Dialogue context is truncated to 1,000 characters per prior message to manage judge token cost; this caps prompt length while preserving topical and stylistic cues that drive multi-turn quality. Scores are returned in structured JSON format with reasoning.

C Per-Model Pricing Table

Each candidate model m has three distinct per-token unit prices in our cost model: full-prefill c^m_{prefill} (paid for tokens that miss the cache), cached-input c^m_{cached} (paid for tokens served from KV-cache), and generation c^m_{gen} (paid per output token). The serving-state-aware cost in Section 3.3 multiplies each by the corresponding token count L^t_{prefill}, $L^{m,t}_{\text{cache}}$, L^t_{gen}. We take c^m_{prefill} and c^m_{gen} directly from publicly listed OpenRouter pricing for each model. The cached-input price is not separately listed for these open-source Qwen3 models, so we set $c^m_{\text{cached}} = 0.2\, c^m_{\text{prefill}}$ (i.e., an 80% discount), following the cache-discount convention documented by Alibaba Cloud Model Studio.[2] Table 4 reports the resulting per-token prices.

Table 4: Per-token prices for the five Qwen3 models in our pool, in USD per 1M tokens. Full-prefill (c_{prefill}) and generation (c_{gen}) prices are taken from publicly listed OpenRouter pricing (snapshot retrieved during paper preparation); the cached-input price is set to $c_{\text{cached}} = 0.2\, c_{\text{prefill}}$ following Alibaba Cloud Model Studio's context-cache convention. Note that posted prices are non-monotone in model size (e.g., Qwen3-32B at \$0.16/M is cheaper than Qwen3-14B at \$0.35/M) because providers set prices by tier rather than by size; we use posted prices as-is rather than imputing a monotone schedule.

Model	Total params	c_{prefill}	c_{cached}	c_{gen}
Qwen3-0.6B	0.6B	\$0.11	\$0.022	\$0.42
Qwen3-8B	8B	\$0.18	\$0.036	\$0.70
Qwen3-14B	14B	\$0.35	\$0.070	\$1.40
Qwen3-32B	32B	\$0.16	\$0.032	\$0.64
Qwen3-Next-80B-A3B-Instruct	80B	\$0.15	\$0.030	\$1.20

Cost accounting in evaluation. Cost accounting is identical for all methods. Equation 9 is used only by CoR at routing time. For evaluation, every method's cost is recomputed from its *realized* cache-hit tokens, uncached tokens, and output tokens with the unit prices in Table 4, so baselines receive full credit for any cache reuse.

D Implementation Details

Encoder and training. All learned routers (CoR and baselines) use *all-MiniLM-L6-v2* [32] as the shared text encoder (d=384). Routing models are trained on the 80% training split with AdamW (lr 1×10^{-4}, batch size 64, 20 epochs, early stopping on validation MSE). The training objective is mean-squared error between predicted quality $\hat{Q}(m, Q_t, H_t)$ and the judge-assigned ground-truth quality, summed across all K=5 candidates.

DSP configuration. The local-history window is W=2, and the global state dimension is d=384 (matching the encoder).

[2] `https://www.alibabacloud.com/help/en/model-studio/context-cache`

Cost normalization and λ sweep. We divide each turn's raw cost (token-weighted by per-model unit cost) by the maximum-model cost so that costs lie in $[0, 1]$. Output length per candidate is predicted by a lightweight regression head following CARROT [26]: a per-candidate linear head on the same MiniLM-L6 query embedding, trained on the 80% training split with mean-squared error against the candidate's actual response length and reused at routing time. We sweep $\lambda \in \{0, 0.05, 0.10, \ldots, 1.0\}$ when computing the deferral curve.

Baseline adaptations. *Bao et al.* [30] originally route between two models with a binary switching penalty; we adapt their formulation to our five-model pool by treating each pairwise switch under the same penalty schedule. *GMTRouter* [29] originally learns per-user preferences from cross-conversation histories; we instantiate it by treating each conversation hash as a distinct user, using the authors' reference implementation. *History-strategy variants* (Query-only, Full History, RAG Retrieval) share CoR's encoder and training pipeline but differ in how they construct the input to the quality predictor. *Full-history + SST baselines* feed the concatenated dialogue history to CARROT, CSCR, and IRT-Router, keep each router's own quality predictor, and replace its cost estimator with CoR's SST. The *BERT full-history* variant fine-tunes a BERT-base encoder (109.5M parameters) end-to-end on the full dialogue history.

Component ablation training. To ablate DSP or SST individually, we re-train CoR with the corresponding module replaced by its single-turn or cache-unaware counterpart while keeping the other module unchanged.

E Compute Resources

Routing module training. CoR's routing modules consist of a frozen *all-MiniLM-L6-v2* encoder ($\sim$22M parameters, kept fixed) plus lightweight per-candidate task heads. Each training configuration trains for at most 20 epochs over the 80% training split with batch size 64 and completes in under 2 hours on a single NVIDIA A100 (40GB) GPU. The full sensitivity sweep (Appendix F: window W, global-state architecture, encoder, cache discount; Appendix G: pool size K; Appendix H: eviction rate p_{evict}) totals roughly 30 training runs and approximately 60 GPU-hours.

Candidate-response generation. Generating per-(turn, candidate) responses across the five Qwen3 models on $\sim$160K turns (WildChat + LMSYS-Chat-1M) is performed via the OpenRouter API. The resulting shared-context responses are generated once and can be reused for training. Held-out evaluation generates the replies selected along each routing trajectory.

LLM-judge scoring. DeepSeek-V3.1 scores each shared-context candidate response on a 1–10 scale. This one-time labeling cost scales with $|\text{turns}| \times K$. The same judge scores replies generated during held-out evaluation. The judge meta-evaluation (500 samples $\times$ 4 candidate judges, Appendix B) is a small fraction of the total judge spend.

Evaluation. Auto-regressive evaluation runs each routing strategy as a single end-to-end trajectory per held-out conversation. Per-strategy evaluation across the held-out test split with $|\lambda|$=21 values completes in under 1 hour on the same A100 GPU; total wall-clock is dominated by candidate inference latency rather than the routing decision itself, which is sub-millisecond per turn.

F Sensitivity to Design Choices

This appendix provides the full sensitivity analysis referenced in the main text. Table 5 sweeps four key design axes of CoR on WildChat. **(a)** The local-history window W has a clear optimum at W=2: W=1 underfits short-range coreference, while $W \geq 3$ admits diluting attention from less relevant turns. AUDC varies by at most 0.02 across the sweep, indicating mild sensitivity. **(b)** The global state architecture matters: the gated GRU update outperforms a plain RNN (ΔAUDC $+0.03$) and matches a single-layer Transformer at a fraction of the compute, supporting the choice of a recurrent rather than purely attentional aggregator. **(c)** CoR is largely insensitive to the text encoder: substituting BERT-base or E5-small for MiniLM-L6 changes AUDC by ≤ 0.01, with E5-small giving a small additional gain. **(d)** The cache-discount sweep is the most consequential. As $d^m_{\text{cache}}/c^m_{\text{prefill}}$ shrinks from 0.95 (near-zero cache cost, vLLM-style) to 0.5 (small dividend), AUDC drops by 0.04 and QNC rises from 0.55 to 0.78. CoR remains the strongest method across all settings, but absolute gains

Table 5: Sensitivity of CoR to design choices on WildChat. Default configuration in *italics*.

Variant	AUDC ↑	QNC ↓
(a) Local-history window size W (DSP)		
$W = 1$	0.72	0.65
$W = 2$ *(default)*	0.74	0.59
$W = 3$	0.74	0.61
$W = 5$	0.73	0.64
(b) Global state architecture (DSP)		
GRU (default)	0.74	0.59
RNN	0.71	0.68
Linear Attention	0.73	0.62
Transformer (1-layer)	0.74	0.60
(c) Text encoder		
all-MiniLM-L6 (default)	0.74	0.59
BERT-base	0.74	0.61
E5-small-v2	0.75	0.58
(d) Cache discount $d^m_{cache}/c^m_{prefill}$ (SST)		
0.5 (small dividend)	0.71	0.78
0.8 *(default)*	0.74	0.59
0.95 (vLLM near-zero)	0.75	0.55

Table 6: Robustness to candidate-pool composition on WildChat. *Top:* pool size K varied from 2 to 6 over Qwen3 models. *Bottom:* cross-family pool at $K{=}5$ that replaces three Qwen3 models with Llama-3.1 and Mistral to test family generalization.

K	Pool composition	CoR (Ours)		CSCR	
		AUDC ↑	QNC ↓	AUDC ↑	QNC ↓
2	Qwen3-0.6B + Qwen3-Next-80B-A3B-Instruct	0.78	0.49	0.74	0.74
3	+ Qwen3-32B	0.77	0.53	0.71	0.79
4	+ Qwen3-14B	0.75	0.56	0.70	0.81
5	+ Qwen3-8B *(default)*	0.74	0.59	0.68	0.85
6	+ Qwen3-4B	0.73	0.61	0.66	0.87
Cross-family pool at $K{=}5$ (replaces 3 Qwen3 members with Llama-3.1 and Mistral)					
5	Qwen3-0.6B + Llama-3.1-8B + Mistral-7B-Instruct + Qwen3-32B + Qwen3-Next-80B-A3B-Instruct	0.73	0.62	0.66	0.86

are correlated with how much the deployment platform actually rewards cache reuse, an empirically observable property of the serving stack.

G Candidate-Pool Composition

This appendix expands the candidate-pool composition study summarized in the main text. Table 6 reports the full quality-cost metrics for pool sizes $K \in \{2, 3, 4, 5, 6\}$ over Qwen3 members, and a cross-family pool at $K{=}5$ that replaces three Qwen3 members with Llama-3.1 and Mistral models. CoR's AUDC ranges from 0.78 at $K{=}2$ to 0.73 at $K{=}6$, declining gracefully as harder routing decisions are introduced while QNC stays below 0.62 across all pool sizes. The cross-family pool at $K{=}5$ matches the same-family $K{=}6$ result on AUDC (0.73) within rounding, indicating that DSP's quality estimator and SST's cache tracker do not exploit Qwen-family-specific shortcuts.

H Sensitivity to Cache Eviction

Cache visibility and retention. An instrumented deployment can expose the surviving prefix of each candidate at turn t. Serving engines such as vLLM [8] and SGLang [9] can reuse cached prefixes across turns, but memory pressure can evict a prefix between calls. In the provider-API runtime, the router tracks the prefix from prior calls and learns the realized cache hits from the selected backend's usage report when available.

Sensitivity to eviction rate. We sweep an eviction rate $p_{\text{evict}} \in \{0\%, 10\%, 30\%\}$ that the deployment applies between consecutive turns: with probability p_{evict} a given prior-turn cache entry is evicted

Table 7: Sensitivity of CoR to per-turn cache eviction. The main experiments assume that cache from the current conversation is retained across consecutive turns of the same conversation, which is the typical behavior under prefix-caching engines (vLLM, SGLang) when serving traffic permits cache retention. In this experiment, SST observed the surviving prefix length of each candidate before CoR selected a model. We vary the eviction probability $p_{\text{evict}} \in \{0\%, 10\%, 30\%\}$ on WildChat. CoR retains its lead over the strongest baseline (Bao et al.) at every eviction rate.

p_{evict}	CoR (Ours)			Bao et al.		
	AUDC ↑	QNC ↓	Peak ↑	AUDC ↑	QNC ↓	Peak ↑
0% *(default, main text)*	0.74	0.59	0.95	0.70	0.74	0.95
10%	0.72	0.63	0.95	0.69	0.76	0.95
30%	0.69	0.71	0.95	0.67	0.80	0.95

before the next turn arrives, in which case the corresponding tokens are charged at the full prefill rate. After eviction and before model selection, SST observed each candidate's surviving prefix length and used it to estimate that candidate's cost. Table 7 reports the result. As expected, CoR's lead over the cache-unaware Bao et al. baseline narrows monotonically as fewer cache hits are available to exploit: the AUDC gap shrinks from 0.04 at $p_{\text{evict}}{=}0\%$ to 0.02 at $p_{\text{evict}}{=}30\%$, and the QNC gap shrinks from 0.15 to 0.09. CoR nonetheless retains higher AUDC and lower QNC at every eviction rate (e.g., at $p_{\text{evict}}{=}30\%$: AUDC 0.69 vs. 0.67; QNC 0.71 vs. 0.80). Both methods attain the same reported peak quality at every eviction rate.

Memory overhead in local deployments. CoR does not require the serving stack to retain KV-cache for all K candidates. The serving engine manages retention and eviction. SST uses prefix lengths to estimate routing costs. The memory-cost trade-off of retaining per-model conversation caches under high load therefore manifests as a higher effective eviction rate, which is exactly the axis swept in Table 7: CoR maintains its AUDC and QNC advantages even at a 30% eviction rate. End-to-end results under real, finite commercial cache lifetimes are reported in Appendix I.4.

I Additional Analyses

This appendix reports the full results of the additional analyses summarized in Section 4. Unless noted otherwise, all experiments use WildChat, the default five-model pool, and the evaluation protocol of Section 4.

I.1 Robustness to the LLM Judge

Our main experiments use DeepSeek-V3.1 both to construct training labels and to score evaluation trajectories, which raises the concern that a learned router could exploit judge-specific preferences. We therefore evaluate two cross-family configurations in which the *training judge* (used to re-label the training set and re-train *all* methods) and the *test judge* come from different model families; no model responses are regenerated. The test judge never sees the training labels. As shown in Table 8, method rankings and the relative gaps between methods are preserved. Together with the human-anchored judge selection in Appendix B ($\rho{=}0.84$ against human gold labels), this indicates that the observed gains are not explained by sharing a judge between label construction and evaluation.

I.2 Stability of the Recurrent State across Dialogue Depth

A recurrent state could in principle drift over long conversations. Table 9 reports quality-prediction MSE by turn depth. DSP remains stable from turns 1–2 to turns 8+ ($0.094 \rightarrow 0.088$), as does Global-only ($0.128 \rightarrow 0.127$), whereas Query-only degrades substantially ($0.154 \rightarrow 0.193$). We find no evidence of state drift.

I.3 Out-of-Distribution Generalization

Dialogue-length shift. We train both the full-history BERT and DSP quality predictors only on 2–4-turn WildChat conversations and test on longer conversations without retraining; SST and all other routing settings are unchanged. Within the training range, the two predictors are comparable.

Table 8: Cross-judge evaluation on WildChat. All methods are re-trained on labels from the training judge and evaluated with the test judge.

Training judge	Test judge	Method	AUDC ↑	QNC ↓	Peak ↑
DeepSeek-V3.1	DeepSeek-V3.1 (Table 1)	CSCR	0.68	0.85	0.94
		Bao et al.	0.70	0.74	**0.95**
		CoR	**0.74**	**0.59**	**0.95**
DeepSeek-V4-Pro	Claude Opus 4.8	CSCR	0.664	0.873	0.836
		Bao et al.	0.693	0.761	0.875
		CoR	**0.719**	**0.642**	**0.902**
Claude Opus 4.8	DeepSeek-V4-Pro	CSCR	0.676	0.847	0.851
		Bao et al.	0.704	0.746	0.886
		CoR	**0.737**	**0.608**	**0.916**

Table 9: Quality-prediction MSE by conversation depth on WildChat.

Variant	Turns 1–2 ↓	Turns 3–4 ↓	Turns 5–7 ↓	Turns 8+ ↓	Δ (8+ vs. 1–2) ↓
Query-only	0.154	0.169	0.181	0.193	+0.039
Local-only	0.115	0.106	0.112	0.123	+0.008
Global-only	0.128	0.119	0.118	0.127	−0.001
DSP (Ours)	**0.094**	**0.081**	**0.079**	**0.088**	**−0.006**

Under length shift, BERT's error rises sharply, whereas DSP remains stable through its explicit local/global state (Table 10). End-to-end, DSP + SST gains 0.089 AUDC and reduces QNC by 0.187 at 8+ turns. Since AUDC and QNC include realized cost, DSP's improvement with length does not imply easier quality prediction: longer sessions offer more prefix-cache reuse, and stable routing avoids unnecessary switches, letting SST exploit the accumulated cache.

Table 10: Dialogue-length shift: both quality predictors are trained on 2–4-turn WildChat conversations only.

Test length	Quality MSE ↓		BERT + SST		DSP + SST	
	BERT	DSP	AUDC ↑	QNC ↓	AUDC ↑	QNC ↓
2–4 turns (in-distribution)	0.086	0.092	0.735	0.605	0.730	0.613
5–7 turns	0.121	0.096	0.701	0.668	0.737	0.576
8+ turns	0.183	0.098	0.653	0.739	0.742	0.552

Topic-distribution shift. We transfer bidirectionally between WildChat and LMSYS-Chat-1M without target-corpus training or recalibration. WildChat is dominated by creative and assistive writing (61.9%, vs. 6.7% coding [3]), whereas LMSYS-Chat-1M's largest topic clusters are coding and software [5]. Table 11 shows that DSP + SST loses only 0.020–0.030 AUDC under transfer, compared with 0.073–0.075 for BERT + SST.

I.4 End-to-End Commercial Billing

Commercial serving platforms already implement the per-model prefix reuse assumed by Equation 9, and SST only tracks each candidate's reusable-prefix length, not the KV-cache itself. The 20% cached-input price used in the main experiments follows Alibaba Cloud Model Studio's explicit-cache policy; Anthropic publishes a similar explicit-cache structure with a 5-minute lifetime, and Gemini Flash uses an implicit cache with a lifetime of approximately 3–5 minutes. To validate the cost model end-to-end under such finite cache lifetimes, we run WildChat conversations through OpenRouter with a heterogeneous pool of three model families (Claude, Gemini, and Qwen), with a lightweight and a mid-tier model from each, under each provider's own caching policy. Cache hits and charges for all methods, including baselines, are read directly from OpenRouter's usage-accounting response rather than computed from our cost model. Table 12 shows that CoR's billed cost is \$1.142 per 1K turns, 19.7% lower than Bao et al.'s \$1.422, with the cache-hit rate rising from 19.4% to 31.8%. Together with Appendix H, this shows that CoR's advantage holds under real, finite commercial cache policies and in a cross-family pool.

Table 11: Topic-distribution transfer without target-corpus training. Drops are relative to target-trained models (BERT + SST / DSP + SST: 0.684 / 0.700 on LMSYS-Chat-1M; 0.723 / 0.740 on WildChat).

Train → Test	BERT + SST		DSP + SST	
	AUDC ↑	Transfer drop	AUDC ↑	Transfer drop
WildChat (writing) → LMSYS (coding)	0.611	−0.073	**0.680**	**−0.020**
LMSYS (coding) → WildChat (writing)	0.648	−0.075	**0.710**	**−0.030**

Table 12: End-to-end billing on WildChat under finite commercial cache policies (heterogeneous Claude / Gemini / Qwen pool via OpenRouter).

Method	AUDC ↑	QNC ↓	Peak ↑	Billed cost ($/1K turns) ↓	Cache-hit rate ↑
CSCR	0.664	0.869	0.887	1.646	11.0%
Bao et al.	0.699	0.758	0.912	1.422	19.4%
CoR	**0.734**	**0.616**	**0.941**	**1.142**	**31.8%**

I.5 Routing Distribution across the Candidate Pool

To test whether CoR makes fine-grained selections or relies on a small subset of the pool, we add Qwen3-4B to the default pool, producing a six-model pool with four intermediate candidates (AUDC/QNC $0.73/0.61$ for CoR vs. $0.66/0.87$ for CSCR; Table 6), and report routing shares at representative λ (Table 13). As λ increases, CoR shifts toward cheaper candidates while making substantial use of the intermediate models; at λ=0.5, the four intermediate models receive 70.7% of CoR's turns, compared with 52.9% for Bao et al.

Table 13: Routing shares (%) in the six-model Qwen3 pool on WildChat.

λ / regime	Method	0.6B	4B	8B	14B	32B	80B-A3B
0.1 / quality-focused	CoR	3.4	5.8	8.1	11.5	23.9	47.3
	Bao et al.	2.9	4.8	7.4	10.6	21.7	52.6
0.5 / balanced	CoR	12.6	17.4	13.8	9.1	30.4	16.7
	Bao et al.	21.8	11.3	10.1	6.8	24.7	25.3
0.9 / cost-focused	CoR	56.8	26.1	4.7	1.3	9.4	1.7
	Bao et al.	75.9	12.7	3.1	1.0	5.2	2.1

J Limitations

Pool excludes proprietary frontier models. Our main pool consists of open-weight Qwen3 models, with cross-family checks using Llama-3.1 and Mistral. State-of-the-art proprietary models such as Claude Opus, Claude Sonnet, and Claude Haiku are not included, even though production routers in practice often mix open-weight and proprietary endpoints. Incorporating such models would broaden the quality-cost frontier the router operates over and is a natural next step. Our commercial-billing experiment (Appendix I.4) already includes lightweight and mid-tier Claude and Gemini models.

Domain-specialized candidates. Evaluating explicitly domain-specialized candidates (e.g., medical or legal LLMs) requires only per-turn labels for them and no change to CoR.

Cache retention under load. Measuring cache retention in a local multi-model deployment under sustained load is left to future work.

K Broader Impacts

CoR is a routing layer over publicly available LLMs and does not generate content itself, so most societal impact accrues to the underlying models rather than the router. We highlight three considerations specific to CoR's operating regime.

Efficiency and access. By exploiting KV-cache reuse and cost-aware model selection, CoR can reduce per-conversation serving cost and energy, which may broaden access to high-quality conversational LLMs at lower price points and lower carbon footprints per query.

Routing-induced exposure. If the router systematically prefers particular candidates for particular query types, any biases, factual errors, or failure modes of those candidates are amplified for users issuing those query types. Deployers should monitor per-model behavior across the realized routing distribution, not only models in isolation, and consider auditing routing decisions for fairness across sensitive query categories.

Privacy and cache isolation. SST consumes KV-cache state from the serving engine across turns of the same conversation. Operators should ensure cache isolation across distinct user sessions (a default property of vLLM and SGLang under standard configurations) so that cache reuse cannot leak information across users; CoR itself adds no new persistence beyond what the serving engine already maintains.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Figure 1 and Section 1 introduce the MultiTurnRoute platform, its router interface, and its Arena. Section 3 describes CoR and its DSP and SST modules. The quality and cost results in Section 4 measure CoR on the five-model benchmark.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix J discusses the coverage of the main candidate pool and cache retention under load.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper makes empirical contributions and does not include formal theorems or proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 3 fully specifies the DSP and SST architectures with all governing equations; Appendix B documents dataset construction, preprocessing, and the LLM-judge selection protocol; Appendix C reports per-model pricing; Appendix D lists encoder, optimizer, learning rate, batch size, epochs, λ grid, and baseline adaptations.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The submission supplement includes DSP, SST, all six experimental baselines, and the evaluation scripts. WildChat and LMSYS-Chat-1M are public. The project repository is currently private, and the derived routing data and pretrained router checkpoints do not have public downloads.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 (Experimental Setup) specifies the model pool, dataset, $80/20$ conversation-level split, and evaluation metrics; Appendix D provides the encoder, AdamW optimizer settings (lr 1×10^{-4}, batch size 64, 20 epochs, early stopping on validation MSE), DSP configuration (W=2, d=384), λ grid, and baseline adaptations.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report point estimates rather than error bars to keep tables compact; statistical robustness is instead evidenced by consistency across the multiple ablation and sensitivity studies (Appendix F, G, H), which sweep design axes orthogonally and converge on the same qualitative ranking.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix E reports compute resources for each experiment phase: routing-module training (single A100, $<$2 hours per configuration, $\sim$60 GPU-hours total for the full sensitivity sweep), candidate-response generation and LLM-judge scoring (one-time API costs during dataset construction), and auto-regressive evaluation ($<$1 hour per routing strategy on a single A100).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses publicly released conversation datasets under their original licenses, runs lightweight routing on top of openly available LLMs, and introduces no privacy- or safety-sensitive component.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix K discusses three considerations specific to CoR's operating regime: (i) potential positive impact via reduced per-conversation serving cost and energy; (ii) routing-induced exposure, where systematic candidate preference for particular query types can amplify the biases or failure modes of those candidates and warrants per-distribution auditing; (iii) privacy and cache isolation requirements that operators must enforce when SST consumes KV-cache across turns.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We do not release new high-risk pretrained models or scraped data; CoR is a thin routing layer over publicly available LLMs and conversation corpora, which retain their original licenses and access controls.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All third-party assets (WildChat [3], LMSYS-Chat-1M [5], the Qwen3 model family, MiniLM-L6 [32], DeepSeek-V3.1, OpenRouter pricing) are cited at first use, accessed under their publicly stated terms, and used within their license boundaries.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: Appendix B documents preprocessing, judge selection, and per-turn quality scoring for the five-model Qwen3 benchmark. Section 5 documents the router interface, the separate nine-model Arena replay pool, and the interactive evaluation protocol.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [Yes]

Justification: Appendix B (LLM judge selection, Scoring details) describes the meta-evaluation protocol used to select the LLM judge: 500 sampled responses, 3 independent expert ratings per response on a 1–10 scale, total 1,500 ratings, with rubric and JSON-formatted output described. Annotators were internal expert raters; no crowdsourcing platform was used.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The annotation activity consisted of internal expert raters scoring LLM outputs for judge meta-evaluation; it is exempt from human-subjects research and did not require IRB review.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The candidate LLMs being routed (Qwen3 family) and the LLM judge (DeepSeek-V3.1) are central to the experimental pipeline and are described in Section 4 and Appendix B. LLM usage in our setting is standard (judge meta-evaluation, candidate inference) rather than a novel methodological component.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.