# Chain-of-Route: State-Aware LLM Routing for Multi-Turn Conversations

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

## Abstract

LLM routing dispatches each user query to the most suitable model from a pool of candidates, balancing response quality against serving cost. As LLMs are increasingly deployed in conversational interfaces, routing decisions must now be made repeatedly within an ongoing session. Existing routers treat each query independently, implicitly assuming that the optimal model depends only on the current input. This assumption ignores two forms of session state that accumulate across turns: the dialogue state, which shapes query meaning and difficulty, and the *serving state*, which determines each model's true *serving cost* via KV-cache reuse. Neglecting dialogue state leads to misrouting when queries depend on prior turns, while ignoring serving state causes systematic cost overestimation that worsens as conversations grow longer. A practical multi-turn router must therefore capture dialogue state for accurate quality prediction, and track per-model serving state for accurate cost estimation. To address this gap, we introduce Chain-of-Route (CoR), the first state-aware routing framework for multi-turn conversations. CoR comprises two modules: a Dialogue State Propagator (DSP) that maintains a compact recurrent state to capture session-level context across turns, and a Serving State Tracker (SST) that tracks per-model KV-cache to compute true serving costs. Together with the framework, we construct a unified multi-turn routing benchmark from WildChat and LMSYS-Chat-1M with per-turn quality annotations across five LLMs. Experiments show that CoR achieves state-of-the-art quality-cost tradeoffs, reducing QNC by up to 20.3% over the strongest baseline while improving response quality.

## 1 Introduction

Large language models (LLMs) have achieved remarkable success across a broad range of tasks and are increasingly deployed as general-purpose conversational assistants. This success has catalyzed the emergence of numerous LLMs with diverse architectures, scales, and training objectives, leading to substantial variation in their areas of specialization, response quality, and inference cost. Generally, larger models tend to provide higher quality but come with higher costs, while smaller ones are more affordable but less capable. This diversity creates a new systems-level challenge: deciding, for each query, which model to use to achieve the best balance between quality and cost.

LLM routing has emerged as a promising solution to this challenge, dynamically assigning each query to the most suitable LLM. Existing routers treat each query independently, selecting a model based solely on the current input [1]. In practice, however, queries rarely arrive in isolation: users interact with LLM-based systems through extended conversations involving multiple turns. Analysis of WildChat [2], a corpus of 1M real user interactions, reveals that 41% of conversations span multiple turns, with similar patterns across other conversation datasets [3, 4]. Within such conversations, topics and user goals frequently shift across turns [5, 6], so the optimal model may differ from turn to turn, motivating per-turn routing that adapts model selection dynamically. A straightforward

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

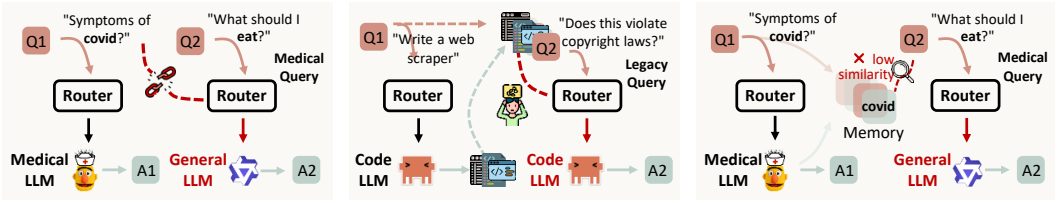


Figure 1: Challenges of adapting single-turn LLM routing to multi-turn conversations regarding dialogue state. (a) Routing using only the current query ignores conversational context. (b) Routing using the full context can let dominant prior content overshadow the current query's intent (*context overshadowing*). (c) Routing using selectively retrieved context misses implicit dependencies when users refer back without repeating keywords.

approach is to apply existing single-turn routers independently at each turn. However, in multi-turn conversations, both query meaning and serving cost depend on state that accumulates across turns. Single-turn routers, which see only the current input, lack access to this state and therefore face two key challenges, as we elaborate below.

The first challenge is that query meaning often depends on prior turns. In multi-turn conversations, some queries are self-contained, while others inherit context from the preceding dialogue. For instance, a follow-up such as "what should I eat?" appears to be a general question when viewed in isolation, but given a preceding turn about covid symptoms, it is clearly a medical query. Routing based solely on the current query is *context-blind*: it misses this dependency and misroutes accordingly, as shown in Figure 1 (a). A natural fix is to feed the full dialogue history into the router. However, this introduces a different failure mode that we term *context overshadowing*: when the history contains dominant prior content, such as a lengthy code response, it can overshadow the current query's true intent and again cause misrouting, as illustrated in Figure 1 (b). One might then consider selectively retrieving only the relevant history. Yet context dependency in conversations is often implicit: users refer back to earlier topics without repeating keywords, so similarity-based retrieval misses these *implicit dependencies*. In the example above, the embedding of "what should I eat?" bears little similarity to the preceding medical discussion, yet the query implicitly refers to dietary advice during illness, as shown in Figure 1 (c). In short, the router must be aware of the accumulated *dialogue state*, yet independently routing each turn, prepending the full history, and selectively retrieving relevant turns all fail to reliably capture it.

The second challenge concerns cost estimation. Modern LLM serving engines such as vLLM [7] and SGLang [8] cache intermediate computations (i.e., the KV-cache) from previous turns and automatically reuse them when the same model is called again, significantly reducing inference cost. For example, Anthropic charges \$3.00 per 1M input tokens for Claude Sonnet but only \$0.30 per 1M for cached tokens, a 90% reduction [9]. In multi-turn conversations, each candidate model may have served a different subset of prior turns, so the amount of reusable cache, and thus the true serving cost, varies across models. Existing routers ignore this *serving state* when estimating cost. As illustrated in Figure 2, this causes the router to switch to a seemingly cheaper model that must prefill the full conversation from scratch, while the previously used model, which already holds extensive cached context, would have been cheaper to continue with. This problem worsens as conversations grow longer and more cache accumulates, leading to increasingly suboptimal and costly routing decisions.

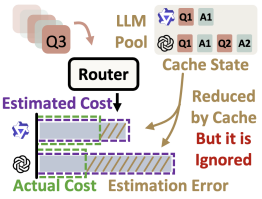


Figure 2: Ignoring serving state produces misleading cost rankings.

To address both challenges, we propose **Chain-of-Route (CoR)**, the first state-aware LLM routing framework for multi-turn conversations, which explicitly tracks both the dialogue state and the serving state to make informed routing decisions. For the dialogue state, we introduce the **Dialogue State Propagator (DSP)**, which maintains a compact hidden state that incrementally captures conversation-level context without re-reading the full dialogue at each turn. DSP decomposes this state into a *local history* that captures turn-level context and a *global history* that captures conversation-level trends, with a context-dependency gate that adaptively controls how much history to incorporate for each query. For the serving state, we introduce the **Serving State Tracker (SST)**, which tracks the KV-cache status of each candidate model in the pool across turns and accounts for cache reuse when

estimating cost, yielding estimates that more closely reflect the actual serving cost. Experiments on WildChat and LMSYS-Chat-1M demonstrate that CoR reduces QNC by up to 20.3% over the strongest baseline while improving response quality.

## 2 Background and Related Work

**Multi-Turn LLM Conversations.** Real-world interactions with LLMs frequently span multiple turns rather than consisting of isolated queries. Large-scale conversation datasets such as WildChat [2], ShareGPT [3], and LMSYS-Chat-1M [4] confirm this prevalence and have been used to study phenomena such as topic shifts [5, 6, 10] and context dependency across turns [11, 12]. Since different LLMs vary considerably in capability and cost [1], and different turns may require different capabilities, the optimal model may change from turn to turn, motivating per-turn routing. However, existing multi-turn datasets [2–4, 13, 14] record each conversation with a single LLM, providing no multi-model annotations. Conversely, existing routing benchmarks [15] provide multi-model quality labels but only for isolated, single-turn queries. To our knowledge, no existing dataset combines multi-turn conversations with per-turn multi-model annotations.

**KV-Cache Management in LLM Serving.** When an LLM processes a conversation, it builds up cached intermediate states, i.e., KV-cache, for the tokens it has seen. In a multi-turn conversation, successive turns share much of the same history, so modern serving engines cache and reuse these states rather than recomputing them from scratch. vLLM [7] manages KV-cache via paged memory allocation, SGLang [8] supports automatic prefix caching, and CachedAttention [16] extends reuse across multi-turn conversations by offloading KV states to host memory. This reuse directly reduces serving cost: for example, cached input tokens cost only 10% of the standard input price for Claude Sonnet [9]. The more cached state a model holds for the current conversation, the less new computation it needs. In a routing setting where different models may serve different turns, each candidate accumulates a different amount of cached state over time. The true serving cost of a model at a given turn therefore depends not only on the input but also on its serving state, that is, how much of the conversation it has already processed. No existing router accounts for this dependency.

**LLM Routing.** The growing diversity of LLMs in capability and cost has motivated *LLM routing*, where a router selects the most suitable LLM for a given query to balance response quality and serving cost. A rich line of single-turn routers has developed in this space, varying primarily in how they predict per-candidate quality [1, 15, 17] and cost [18–20]; we defer a detailed survey to Appendix A. The unifying limitation is that these routers treat each query independently and do not model how prior turns reshape query meaning or serving cost. Two recent works extend routing beyond single-turn settings. GMTRouter [21] addresses a complementary problem, personalizing model selection to individual user preferences by learning from cross-conversation interaction histories, whereas our work focuses on state-aware routing *within* an ongoing conversation. Bao et al. [22] extend RouteLLM [1] to multi-turn conversations, but two fundamental differences remain. First, they treat each turn as an independent routing decision, whereas we model multi-turn routing as a sequential process in which each decision both depends on and shapes the state available to future decisions. Second, their cost model assumes that serving cost depends only on whether the router switches models at the current turn, whereas we observe that cost depends on the full history of past routing decisions, since each model accumulates a different amount of cached computation over time.

## 3 Method

### 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:

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

Existing 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

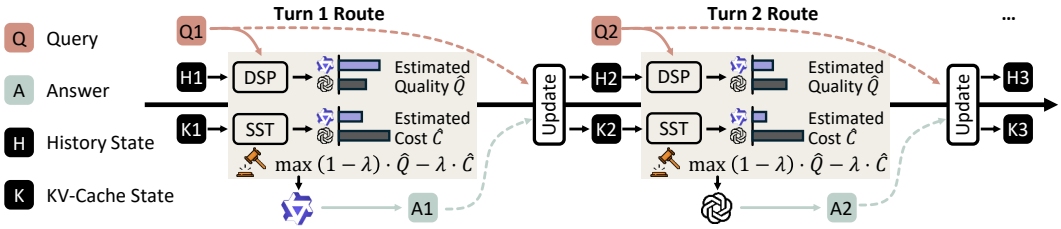


Figure 3: Overview of Chain-of-Route (CoR). At each turn, the router receives the current query $Q_t$, the dialogue state $H_t$, and the serving state $\mathcal{K}_t$. DSP conditions on $H_t$ to estimate quality $\hat{Q}$, while SST conditions on $\mathcal{K}_t$ to estimate cost $\hat{C}$. The model maximizing $(1-\lambda)\cdot\hat{Q}-\lambda\cdot\hat{C}$ is selected, and both states are updated before propagating to the next turn.

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.

Our framework, Chain-of-Route (CoR), makes this chain explicit. As illustrated in Figure 3, the router receives the current query $Q_t$ along with two forms 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 Eq. (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 1, 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 $H_t^\ell$ 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 shared pre-trained text encoder (e.g., BERT) 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

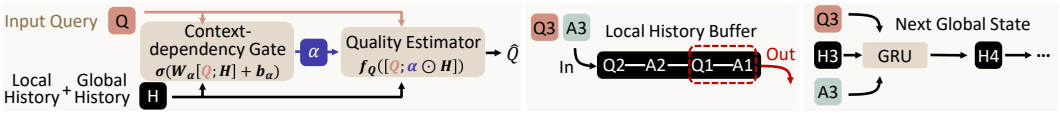


Figure 4: Overview of the Dialogue State Propagator (DSP). (a) Context-dependency Gate: query $Q$ and dialogue state $H$ produce a gate $\alpha_t$, which scales $H$ before the quality estimator $f_Q$. (b) Local History Update: a FIFO buffer (e.g., of size $W{=}2$) evicts the oldest pair $(Q_1, A_1)$ 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.

advance both states to $\mathcal{B}_t$ and $H^g_{t+1}$. 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 4 (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:

$$H^\ell_t = \mathrm{CrossAttn}(Q_t,\ \mathcal{B}_{t-1}) = \mathrm{softmax}\left(\frac{(Q_t W^\ell_Q)(\mathcal{B}_{t-1} W^\ell_K)^\top}{\sqrt{d}}\right)\mathcal{B}_{t-1} W^\ell_V, \tag{2}$$

where $W^\ell_Q \in \mathbb{R}^{d\times d}$ and $W^\ell_K, W^\ell_V \in \mathbb{R}^{2d\times d}$ 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 4 (c). When routing turn $t$, the router reads the current global state $H^g_t$ (produced after turn $t-1$) directly. After the selected model generates $A_t$, two gates jointly determine how $H^g_t$ is advanced to $H^g_{t+1}$ using the new turn input $x_t = [Q_t; A_t]$. The *relevance gate* $r_t$ assesses which dimensions of $H^g_t$ are pertinent to the new turn, selectively filtering outdated information:

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

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):

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

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

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

where $\sigma$ 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:

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

The global state is initialized as $H^g_1 = \mathbf{0}$, 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 $H^\ell_t$ and $H^g_t$ in hand, DSP forms the dialogue state $H_t = [H^\ell_t; H^g_t]$ and computes a *context-dependency gate* $\alpha_t \in [0, 1]$ that assesses whether the current query requires conversational context (Figure 4 (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:

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

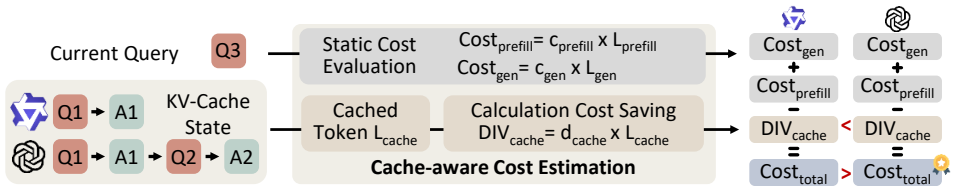


Figure 5: Overview of SST. Taking turn 3 as an example, Qwen caches $(Q_1, A_1)$ while GPT caches $(Q_1, A_1, Q_2, A_2)$. (Left) Each model's serving state and cached token count $L_{\text{cache}}$. (Middle) Static cost estimation yields a lower estimate for Qwen; cache dividend estimation computes each 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.

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 $c^m_{\text{prefill}}$ and $c^m_{\text{gen}}$ are model $m$'s per-token prices and $L^t_{\text{prefill}}$, $L^t_{\text{gen}}$ are the prefilling and generation token counts. This does not account for KV-cache reuse: a model that has served 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. We illustrate SST with a concrete example in Figure 5. Consider turn $t$=3, where the user submits query $Q_3$ and two candidate models have different serving histories: Qwen served only turn 1 and retains the KV-cache for $(Q_1, A_1)$, while GPT served turns 1 and 2 and retains the cache for $(Q_1, A_1, Q_2, A_2)$.

**Serving State Tracking.** SST maintains the cache state $L^{m,t}_{\text{cache}}$ for each candidate model $m$, defined as the number of tokens for which model $m$ currently retains KV-cache at turn $t$. Since the router operates within the multi-LLM serving stack, $L^{m,t}_{\text{cache}}$ is directly read from the serving engine at decision time, agnostic to the engine's specific caching or eviction policy. In our running example (Figure 5, left), Qwen holds $L^{\text{Qwen},3}_{\text{cache}} = |Q_1, A_1|$ while GPT holds $L^{\text{GPT},3}_{\text{cache}} = |Q_1, A_1, Q_2, A_2|$, reflecting their different serving histories.

**Static Cost Estimation.** As shown in Figure 5 (middle, top), a state-unaware router prices each candidate by the same static formula based on the full context length $L^t_{\text{prefill}}$: a prefilling cost $\text{Cost}_{\text{prefill}} = c^m_{\text{prefill}} \cdot L^t_{\text{prefill}}$ and a generation cost $\text{Cost}_{\text{gen}} = c^m_{\text{gen}} \cdot L^t_{\text{gen}}$, following the static cost formulation adopted by single-query routing work [20]. Because per-token prices $c^m_{\text{prefill}}$ and $c^m_{\text{gen}}$ 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 (Figure 5, middle, bottom). With per-token cache discount $d^m_{\text{cache}} = c^m_{\text{prefill}} - c^m_{\text{cached}}$ (where $c^m_{\text{cached}}$ is the cached-input price set by the provider for black-box APIs, or $\approx 0$ for self-hosted models), the dividend for candidate $m$ is $\text{DIV}_{\text{cache}}(m) = d^m_{\text{cache}} \cdot L^{m,t}_{\text{cache}}$. Subtracting it from the static estimate yields the cache-aware cost:

$$\hat{C}(m, Q_t, L^{m,t}_{\text{cache}}) = \overbrace{c^m_{\text{prefill}} \cdot L^t_{\text{prefill}}}^{\text{Cost}_{\text{prefill}}} - \overbrace{d^m_{\text{cache}} \cdot L^{m,t}_{\text{cache}}}^{\text{DIV}_{\text{cache}}} + \overbrace{c^m_{\text{gen}} \cdot L^t_{\text{gen}}}^{\text{Cost}_{\text{gen}}}. \tag{9}$$

As illustrated in Figure 5 (right), although Qwen has a lower static estimate than GPT, GPT enjoys a much larger $\text{DIV}_{\text{cache}}$ because it has served more prior turns; after subtracting the dividend, GPT's true serving cost is lower, reversing the static ranking. 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 Results

## 4.1 Experimental Setup

**Model pool.** We use $K{=}5$ models: Qwen3-0.6B, Qwen3-8B, Qwen3-14B, Qwen3-32B, and Qwen3-Next-80B-A3B-Instruct. We take each model's full-prefill price $c^m_{\text{prefill}}$ and generation price $c^m_{\text{gen}}$ 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 [4, 13, 14] are single-model and do not provide per-turn multi-model quality scores; we therefore construct evaluation data from WildChat [2] and LMSYS-Chat-1M [4]. From each corpus we randomly sample 20,000 English conversations ($\sim$80,000 turns) with $\geq 2$ 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 [14] (details in Appendix B). This shared-context construction is computationally necessary: faithfully enumerating all routing trajectories scales as $|\mathcal{M}|^T$ in the conversation length, which is intractable. Within a fixed context, however, candidate quality scores are directly comparable across models, yielding the per-(turn, candidate) labels DSP and SST are trained against.

At evaluation, we run auto-regressive routing on each held-out conversation. At every turn the router uses DSP and SST to predict per-candidate quality and cost, and selects a model under the routing objective in Equation (1); that model then generates the actual response, which is judged by the same DeepSeek-V3.1 evaluator and fed back as context for the next turn. Because each routing strategy is run end-to-end as a single trajectory per conversation, the evaluation does not require shared-context responses and faithfully reflects deployment conditions, including any response-distribution shift introduced by routing-induced model switches.

**Baseline Methods.** *Random* selects uniformly per turn. Single-turn routers *RouteLLM* [1], *CARROT* [20], and *CSCR* [19] route each turn from the current query alone. Multi-turn routers *Bao et al.* [22] (full-history input + binary switch penalty) and *GMTRouter* [21] (cross-conversation user graph) are the closest competitors. All learned routers share *all-MiniLM-L6-v2* [23] as the text encoder; training hyperparameters, $\lambda$ grid, and baseline adaptations are in Appendix D.

**Evaluation Metrics.** We evaluate each routing strategy using a *deferral curve* [19, 24], which plots the average per-turn response quality against the average normalized cost. Sweeping the cost-sensitivity parameter $\lambda$ over the interval $[0, 1]$ (Equation (1)) traces the deferral curve. Following [24], 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, i.e., RouteLLM, CARROT, CSCR, evaluate each query in isolation and therefore miss both the dialogue state that disambiguates context-dependent queries and the serving state that determines true cache-aware cost. GMTRouter [21] targets cross-conversation user personalization rather than within-conversation state, capturing *who* asks rather than *what context* accumulates, and lands near the single-turn cluster. Bao et al. [22], the strongest multi-turn baseline, feeds full history into a RouteLLM-style router and reduces serving cost to a binary switching penalty;

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

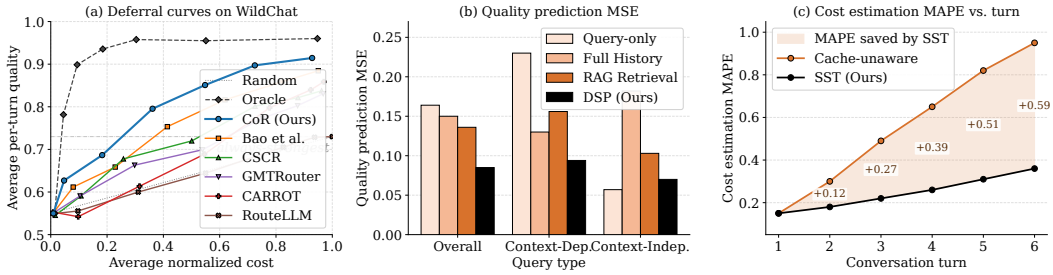


Figure 6: (a) Deferral curves on WildChat as the cost-sensitivity parameter $\lambda$ is swept. (b) Quality prediction MSE for four history strategies, broken down by query type. (c) Cost estimation MAPE as a function of conversation turn for cache-unaware estimation vs. SST.

CoR pulls ahead by avoiding context overshadowing with a compact recurrent dialogue state and by tracking per-model cache. The QNC gap is the largest: CoR matches the most capable LLM's quality at substantially lower cost, an effect driven by SST exploiting cache reuse that all baselines pay for as if from scratch. The two improvements are complementary: DSP makes the router's quality estimate context-aware, and SST makes its cost estimate cache-aware. Combining them together yields better model selection.

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 | 0.61 | 0.55 | 0.93 | 0.68 |
| RouteLLM [1] | 0.56 | 0.98 | 0.73 | 0.62 | 0.93 | 0.78 |
| CARROT [20] | 0.59 | 0.95 | 0.86 | 0.65 | 0.91 | 0.88 |
| CSCR [19] | 0.68 | 0.85 | 0.84 | 0.65 | 0.87 | 0.82 |
| GMTRouter [21] | 0.65 | 0.87 | 0.83 | 0.62 | 0.89 | 0.81 |
| Bao et al. [22] | 0.70 | 0.74 | 0.88 | 0.67 | 0.78 | 0.83 |
| **CoR (Ours)** | 0.74 | 0.59 | 0.92 | 0.70 | 0.65 | 0.89 |

Figure 6 (a) plots the full deferral curve on WildChat as $\lambda$ 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). As shown in Figure 6 (b), DSP outperforms all baselines on context-dependent queries, confirming that its recurrent state effectively captures dialogue state and leverages it for accurate quality prediction. Query-only is the weakest on this slice, validating the context-blind failure mode (Figure 1 (a)) and the importance of tracking dialogue state. On context-independent queries, DSP and Query-only perform comparably, indicating that DSP does not inject harmful signals when history is unnecessary. Full History underperforms on context-dependent queries, validating the context-overshadowing failure mode (Figure 1 (b)); RAG Retrieval likewise underperforms there, validating the implicit-dependency failure mode (Figure 1 (c)). DSP avoids both by compressing history into a compact recurrent state that retains salient context while discarding noise, yielding more accurate quality prediction.

To verify that the gate is not vacuous, Table 2 reports the empirical distribution of $\alpha_t$ on the test split, using the same context-dependent / context-independent partition. The gate fires strongly on context-dependent queries (mean $\alpha_t \approx 0.78$) and stays low on context-independent ones (mean $\alpha_t \approx 0.28$), a separation of about $+0.50$. This confirms that DSP learns to modulate history use rather than uniformly mixing it in, and matches the qualitative motivation in Section 3.2.

Table 2: Context-dependency gate $\alpha_t$ behavior.

| Query type | mean $\alpha_t$ |
|---|---|
| Context-dependent | 0.78 |
| Context-independent | 0.28 |

**Effect of Serving State on Cost Estimation.** Figure 6 (c) shows cost estimation mean absolute percentage error (MAPE) versus conversation turn. At turn 1, both methods start with the same MAPE ($\sim 0.15$); 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 $\sim 0.95$ 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 | 0.85 | 0.91 |
| 3–4 turns | ∼1,240 | 0.67 | 0.74 | 0.85 | 0.58 | 0.84 | 0.93 |
| 5–7 turns | ∼ 700 | 0.65 | 0.75 | 0.88 | 0.53 | 0.82 | 0.93 |
| 8+ turns | ∼ 380 | 0.63 | 0.78 | 0.92 | 0.45 | 0.81 | 0.94 |
| Δ (8+ vs 2) | — | −0.06 | +0.07 | +0.06 | −0.20 | −0.04 | +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 → 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.

## 5 Conclusion

We identify a key limitation of existing LLM routers: they treat each query independently, ignoring two forms of session state that accumulate in multi-turn conversations. Dialogue state shapes query meaning and difficulty across turns, while serving state determines each model's true serving cost via KV-cache reuse. Neglecting either leads to suboptimal model selection and systematic cost overestimation. To address this, we propose Chain-of-Route (CoR), the first state-aware routing framework for multi-turn conversations. CoR introduces two complementary modules: DSP, which maintains a compact recurrent state for context-aware quality prediction, and SST, which tracks per-model KV-cache to estimate true serving costs. We also construct a unified multi-turn routing benchmark from WildChat and LMSYS-Chat-1M with per-turn quality annotations across five LLMs. Experiments show that CoR achieves state-of-the-art quality-cost tradeoffs, reducing QNC by up to 20.3% over the strongest baseline while improving response quality, with ablations confirming the complementary contributions of both modules.

## 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] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

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

[20] 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.

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

[22] 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.

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

[24] 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.

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

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

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

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

[29] 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.

[30] 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.

[31] 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.

[32] 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*.

[33] 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.

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

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

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

# A Survey of Single-Turn LLM Routers

LLM routing has been widely adopted in open-source platforms [25, 26] and commercial deployments such as OpenAI's GPT-5 [27] and Azure AI Foundry [28]. 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, 29, 30], $k$-nearest neighbors [15, 31, 32], and graph neural networks [17].

**Cost predictors.** Early routers rely on static attributes such as per-token pricing [18, 19], while more recent work incorporates output-length prediction [20, 32, 33] 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 [34] aligns routing with human preferences over latency-quality trade-offs, R2-Router [35] introduces reasoning into the routing decision, and past-performance retrieval [36] retrieves past performance records to generalize routing to unseen models.

None of the above account for dialogue or serving state across turns; CoR is the first router to integrate both.

# B Dataset Construction Details

**Why existing multi-turn benchmarks are inadequate.** MT-Bench [14] contains only 80 two-turn dialogues, each completed by a single LLM. MT-Bench-101 [13] scales to 1,388 multi-turn dialogues but remains single-model. LMSYS Chatbot Arena [4] 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 $\geq 2$ turns and $\geq 10$ 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) scoring tractability—candidate-level quality can be aggregated directly without enumerating $|\mathcal{M}|^T$ routing trajectories at training time; 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 [14]: 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 $c^m_{\text{prefill}}$ (paid for tokens that miss the cache), cached-input $c^m_{\text{cached}}$ (paid for tokens served from KV-cache), and generation $c^m_{\text{gen}}$ (paid per output token). The serving-state-aware cost in Section 3.3 multiplies each by the corresponding token count $L^t_{\text{prefill}}$, $L^{m,t}_{\text{cache}}$, $L^t_{\text{gen}}$. We take $c^m_{\text{prefill}}$ and $c^m_{\text{gen}}$ 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 ($c_{\text{prefill}}$) and generation ($c_{\text{gen}}$) 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 | $c_{\text{prefill}}$ | $c_{\text{cached}}$ | $c_{\text{gen}}$ |
|---|---|---|---|---|
| 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 |

## D Implementation Details

**Encoder and training.** All learned routers (CoR and baselines) use *all-MiniLM-L6-v2* [23] as the shared text encoder ($d$=384). Routing models are trained on the 80% training split with AdamW (lr $1\times10^{-4}$, 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).

**Cost normalization and $\lambda$ 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 [20]: 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.* [22] 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* [21] 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.

**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.

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

## 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 $p_{\text{evict}}$) 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. This is a one-time cost paid during dataset construction and is not incurred during routing-model training or evaluation; replication does not require re-generating responses if the released benchmark is used.

**LLM-judge scoring.** DeepSeek-V3.1 is queried through its API to score every generated response on a 1–10 scale; total cost scales linearly with $|\text{turns}| \times K$ and is also one-time. 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 ($\Delta$AUDC $+0.03$) 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 $\leq 0.01$, 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 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 state is observable, not predicted.** A potential concern is whether a model's KV-cache from turn $t-1$ persists when turn $t$ of the same conversation arrives. We emphasize that the router in our setting is owned by the operator of the multi-LLM serving stack: the cache state $L^{m,t}_{\text{cache}}$ of every candidate is therefore directly observable, in the same way that per-token pricing or model-pool composition is observable to the operator. Modern serving engines (vLLM [7], SGLang [8]) typically retain prefix caches at the conversation level under normal traffic. Under sustained high-traffic

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

| Variant | AUDC ↑ | QNC ↓ | Peak ↑ |
|---|---|---|---|
| *(a) Local-history window size $W$ (DSP)* | | | |
| $W = 1$ | 0.72 | 0.65 | 0.91 |
| $W = 2$ *(default)* | 0.74 | 0.59 | 0.92 |
| $W = 3$ | 0.74 | 0.61 | 0.92 |
| $W = 5$ | 0.73 | 0.64 | 0.91 |
| *(b) Global state architecture (DSP)* | | | |
| *GRU (default)* | 0.74 | 0.59 | 0.92 |
| RNN | 0.71 | 0.68 | 0.90 |
| Linear Attention | 0.73 | 0.62 | 0.91 |
| Transformer (1-layer) | 0.74 | 0.60 | 0.92 |
| *(c) Text encoder* | | | |
| *all-MiniLM-L6 (default)* | 0.74 | 0.59 | 0.92 |
| BERT-base | 0.74 | 0.61 | 0.92 |
| E5-small-v2 | 0.75 | 0.58 | 0.93 |
| *(d) Cache discount $d^m_{cache}/c^m_{prefill}$ (SST)* | | | |
| 0.5 (small dividend) | 0.71 | 0.78 | 0.91 |
| 0.8 *(default)* | 0.74 | 0.59 | 0.92 |
| 0.95 (vLLM near-zero) | 0.75 | 0.55 | 0.92 |

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 ↓ | Peak ↑ | AUDC ↑ | QNC ↓ | Peak ↑ |
| 2 | Qwen3-0.6B + Qwen3-Next-80B-A3B-Instruct | 0.78 | 0.49 | 0.90 | 0.74 | 0.74 | 0.85 |
| 3 | + Qwen3-32B | 0.77 | 0.53 | 0.91 | 0.71 | 0.79 | 0.86 |
| 4 | + Qwen3-14B | 0.75 | 0.56 | 0.92 | 0.70 | 0.81 | 0.85 |
| 5 | + Qwen3-8B *(default)* | 0.74 | 0.59 | 0.92 | 0.68 | 0.85 | 0.84 |
| 6 | + Qwen3-4B | 0.73 | 0.61 | 0.93 | 0.66 | 0.87 | 0.84 |
| *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 | 0.90 | 0.66 | 0.86 | 0.83 |

deployment, an entry may be evicted under memory pressure before the next turn arrives, but the eviction policy is set by the deployer and is therefore known to the router; SST consumes the resulting cache state as input rather than estimating it.

**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 $p_{\text{evict}}$ a given prior-turn cache entry is evicted before the next turn arrives, in which case the corresponding tokens are charged at the full prefill rate. Because the eviction policy is set by the operator, SST observes the post-eviction state $L^{m,t}_{\text{cache}}$ at decision time; the sweep therefore characterizes how CoR's cache-aware advantage scales with the rate at which cache is actually retained. 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 its lead on every metric at every eviction rate (e.g., at $p_{\text{evict}}$=$30\%$: AUDC $0.69$ vs. $0.67$; QNC $0.71$ vs. $0.80$). Peak quality is essentially invariant in $p_{\text{evict}}$ for both methods, consistent with eviction being a cost-side phenomenon that does not change which model answers a given turn.

# I 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

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. Under high-traffic deployment, an entry may be evicted before the next turn arrives. Since the router operator owns the serving stack, the eviction policy is observable; we therefore expose an eviction probability $p_{\text{evict}}$ to SST so it can correctly downweight the expected cache hit. Below, we sweep $p_{\text{evict}} \in \{0\%, 10\%, 30\%\}$ on WildChat. CoR retains its lead over the strongest baseline (Bao et al.) at every eviction rate.

| $p_{\text{evict}}$ | CoR (Ours) | | | Bao et al. | | |
|---|---|---|---|---|---|---|
| | AUDC ↑ | QNC ↓ | Peak ↑ | AUDC ↑ | QNC ↓ | Peak ↑ |
| 0% *(default, main text)* | 0.74 | 0.59 | 0.92 | 0.70 | 0.74 | 0.88 |
| 10% | 0.72 | 0.63 | 0.92 | 0.69 | 0.76 | 0.88 |
| 30% | 0.69 | 0.71 | 0.92 | 0.67 | 0.80 | 0.88 |

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.

# J 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: The abstract and Section 1 state our two contributions—DSP for dialogue-state-aware quality prediction and SST for serving-state-aware cost estimation—which are formalized in Section 3 and substantiated by the experiments in Section 4.

   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 I discusses two limitations: the candidate pool excludes proprietary frontier models (Claude family).

   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, $\lambda$ 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: [Yes]

   Justification: The full reference implementation (DSP, SST, all six baselines, training loop, auto-regressive evaluation, and per-experiment shell scripts) is included as anonymized supplementary material under `nips_code/`, with a README, YAML configurations, and unit tests. The underlying corpora (WildChat, LMSYS-Chat-1M) are publicly available and the dataset-construction pipeline is provided in `cor/data/`; the constructed routing benchmark itself will be released alongside the camera-ready to preserve double-blind anonymity during review.

   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 $1 \times 10^{-4}$, batch size 64, 20 epochs, early stopping on validation MSE), DSP configuration ($W{=}2$, $d{=}384$), $\lambda$ 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 J 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 [2], LMSYS-Chat-1M [4], the Qwen3 model family, MiniLM-L6 [23], 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: The constructed multi-turn routing benchmark—per-(turn, candidate) quality scores over five Qwen3 models on WildChat and LMSYS-Chat-1M—is documented in Appendix B (preprocessing, judge selection, scoring details) and will be released alongside the code under the source corpora's original licenses.

    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.