RouterFactory: Unified Infrastructure for Developing, Evaluating, and Deploying LLM Routers

Anonymous authors
Paper under double-blind review

Abstract

No single large language model (LLM) is optimal across all queries and budget constraints, making model routing essential for cost-effective LLM deployment. Existing routers span binary quality predictors, cost-aware cascades, graph-based routers, and agentic routers, yet their diverse formalisms and incompatible implementations, coupled with the absence of a standardized evaluation pipeline, hinder fair comparison and further extension. In this paper, we present a unified formulation of LLM routing as a sequential decision process. Under this formulation, a router can be characterized in terms of five types of components: context encoders, model encoders, scoring functions, decision rules, and learning signals. Existing methods can then be organized into three families: single-turn, multi-turn, and personalized routing. Building on this formulation, we develop an automated pipeline that constructs routing supervision by systematically running a pool of candidate models across benchmarks and evaluates routers in terms of both response quality and inference cost under a unified protocol. The resulting benchmark, **xRouteBench**, spans generic LLM tasks, memory-augmented, vision (image and video), time-series, and personalized routing scenarios. Grounded in the formulation and pipeline, we present **`RouterFactory`**, an open-source infrastructure for standardized and modular implementation of LLM routers, where users can add a new router by implementing only a routing method and a loss function and access built-in implementations of 16 representative routers spanning all three families. Using the library and benchmark, we conduct a systematic empirical study of LLM routing and find that learned routers achieve a 17.4% relative improvement over the largest fixed-model baseline, router rankings reverse in favor of lightweight designs under tighter cost constraints, and user-conditioned routing delivers consistent personalization gains.

1 Introduction

The rapid proliferation of large language models (LLMs) has created a heterogeneous ecosystem of models with widely varying costs and task-specific capabilities, ranging from frontier systems to substantially cheaper open-weight alternatives. Since no single model is optimal across all queries and budget constraints, model routing, which determines which model should handle each query, has become essential for cost-effective LLM deployment. Beyond cost efficiency, routing also matches each query to the candidate model best suited to it and adapts model choice to user-specific preferences (Figure 1). Rich research has been devoted to this problem, from binary routers that arbitrate between a weak and a strong model (Ding et al., 2024; Ong et al., 2024) and cost-aware cascades (Chen et al., 2023; Aggarwal et al., 2024), to reward-guided ensembles, contrastive and graph-based routers (Chen et al., 2024; Feng et al., 2024), personalized routers that adapt to individual users (Sun et al., 2025; Dai et al., 2025), and agentic routers trained with reinforcement learning (Zhang et al., 2025; Feng et al., 2026).

Despite this rapid progress, the field still lacks a unified foundation on which these diverse approaches can be developed and compared, mainly due to two obstacles. First, existing routers are developed under distinct formalisms, released as separate codebases with incompatible interfaces, trained with

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

different supervision, and tuned for different candidate pools, making it difficult to isolate the design elements that actually drive performance or determine whether observed differences stem from the routers themselves or from their broader experimental stacks. Second, evaluating a router is fundamentally more demanding than evaluating a single model, as constructing routing supervision and enabling standardized evaluation require running every candidate model on every benchmark query and scoring each response with task-specific metrics. Existing benchmarks precompute candidate responses for fixed model pools (Hu et al., 2024; Huang et al., 2025b), but are limited to single-turn routing and provide no pipeline for generating supervision for new benchmarks or candidate pools. Consequently, multi-turn and personalized routing still lack a standardized, cost-aware evaluation framework, making routing methods difficult to compare, reuse, and transfer from offline studies to real applications.

In this paper, we introduce **`RouterFactory`**, a unified infrastructure for developing, evaluating, and deploying routing policies over heterogeneous LLM backends. Within this infrastructure, a router is characterized by five types of components: context encoders, model encoders, scoring functions, decision rules, and learning signals. This abstraction accommodates existing routers, which we group into three families: single-turn, multi-turn (including agentic), and personalized routing. It also reduces the effort to add a new router to implementing a routing method and a loss function, while data construction, training, inference, and evaluation remain unchanged, so switching the router, candidate pool, or training objective requires only a configuration change rather than reimplementation. `RouterFactory` includes 16 representative routers spanning all three families, and can expose any of them as an OpenAI-compatible server for deployment on messaging platforms via OpenClaw (OpenClaw, 2026) or through a ComfyUI-based visual interface for code-free prototyping.

`RouterFactory` further automates the construction of routing supervision and evaluation, the main obstacle to comparing routers on equal footing. Its pipeline assembles queries from established benchmarks, dispatches each query to a pool of 18 candidate models spanning a broad price range, and scores every response with task-specific metrics while recording token-level cost. Every router is then evaluated on the same queries, candidate pool, and metrics, enabling direct comparison of their quality-cost trade-offs. With this pipeline, we construct **xRouteBench**, a benchmark that spans generic LLM tasks, memory-augmented, vision (image and video), time-series, and personalized scenarios under one protocol.

Leveraging `RouterFactory` and xRouteBench, we conduct a systematic empirical study of LLM routing across the three families under one protocol. Our study surfaces four findings: (i) **no single router dominates**, as the best router varies across tasks and cost budgets. Strong average performance therefore reflects consistency across scenarios rather than dominance in any single setting; (ii) **learned routing still outperforms the largest fixed-model baseline**, because always selecting the largest model incurs the highest cost yet delivers only mediocre performance, whereas learned routers select smaller, cheaper models for many queries that the largest model answers incorrectly; (iii) **multi-turn routing does not consistently outperform single-turn routing**, as additional rounds of decomposition and aggregation often add cost and redundant information, and their benefit hinges on the capability of the base model that performs them; (iv) **personalization pays off, but only when user context is modeled well**, as a user-conditioned router ranks first under both the persona judge and real human preferences, yet the two settings favor different personalized designs. We also release the library and benchmark in the hope of fostering more systematic progress in LLM routing. A detailed discussion of related work is deferred to Appendix A.

2 A Unified Formulation of LLM Routing

2.1 Routing as a Sequential Decision Process

Existing routers take seemingly incompatible forms, from binary routers that arbitrate between a weak and a strong model (Ding et al., 2024; Ong et al., 2024) and cost-aware cascades (Chen et al., 2023; Aggarwal et al., 2024) to graph-based routers (Feng et al., 2024; Yu et al., 2026a) and agentic routers trained with reinforcement learning (Zhang et al., 2025), yet they can all be formulated as a sequential decision process. At step t, the router observes a state $s_t = (q, u, h_t)$, consisting of the input query q, an optional user context u (e.g., a user identifier with past interactions and feedback), and the interaction history h_t accumulated so far, and takes an action $a_t \in \mathcal{M} \cup \{\perp\}$. The dispatch action $a_t = m$ sends the state to candidate m from the pool $\mathcal{M} = \{m_1, \ldots, m_K\}$ and appends its

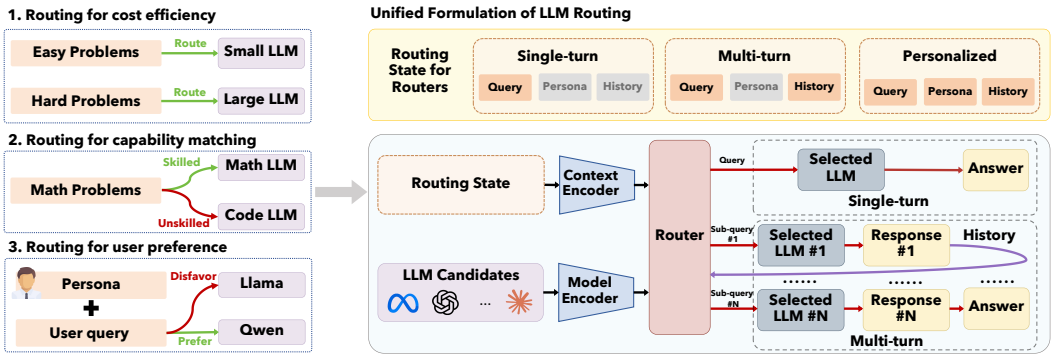


Figure 1: **Overview of LLM routing.** Routing is driven by three needs (left), namely cost efficiency, capability matching, and user preference. Our unified formulation (right) casts all of them as one decision process: a context encoder E_q represents the routing state of query, persona, and interaction history, a model encoder E_m represents each candidate, and the router dispatches the query or its sub-queries to selected models and aggregates their responses into the answer. The single-turn, multi-turn, and personalized families differ only in which part of the state they observe.

response y_t to the history, $h_{t+1} = h_t \oplus y_t$, while the terminating action $a_t = \perp$ ends the episode and aggregates the collected responses into the final answer y; single-turn routing is the special case that terminates after one dispatch. The goal of routing is a policy π whose trajectory $\tau = (a_1, \ldots, a_T)$ produces high-quality answers at low inference cost:

$$\pi^\star \;=\; \arg\max_{\pi}\; \mathbb{E}_{q,\,\tau\sim\pi}\big[\,\alpha\cdot \mathrm{perf}(y \mid q) \;-\; \beta\cdot c(\tau)\,\big], \tag{1}$$

where $\mathrm{perf}(y \mid q)$ aggregates task-specific quality metrics (e.g., accuracy, F1, or an LLM-judged score), $c(\tau)$ sums the monetary or token cost of every call in the routing trajectory τ, and $\alpha, \beta \geqslant 0$ controls the performance–cost trade-off. Under this formulation, a router is characterized by the choice of a **context encoder** E_q that encodes the routing state, a **model encoder** E_m that encodes each LLM candidate, a **scoring function** g and a **decision rule** d that turn the context and model representations into a routing action, and a **learning signal** $\mathcal{L}$ that fits these components toward the optimal policy. This section elaborates on each component and demonstrates how existing routers fall into three families, namely single-turn, multi-turn (including agentic), and personalized, with specific designs of these five components (Figure 1 and Table 1).

Context encoder. The context encoder E_q maps the routing state s_t to the representation on which the routing decision is based, in one of two forms. i) **Embedding-based**: the state becomes a vector, obtained from an off-the-shelf sentence embedding (Hu et al., 2024; Shnitzer et al., 2023), a lightweight encoder trained over frozen embeddings (Ding et al., 2024; Stripelis et al., 2024), or user and session nodes of a heterogeneous interaction graph in personalized routers (Sun et al., 2025; Dai et al., 2025); rating-based routers degenerate to a constant that ignores the query. ii) **Text-based**: the state stays in natural language, either as a query augmented with a draft response and a verification confidence in cascades (Chen et al., 2023; Aggarwal et al., 2024), or verbalized in full in the prompt of a fine-tuned LM router (Ong et al., 2024; Zhang et al., 2025). Which portion of the state E_q is read is precisely what separates the three router families, and any router is personalized by swapping in a user-conditioned E_q while inheriting the remaining components.

Model encoder. The model encoder E_m encodes each candidate in the pool by i) **static metadata** such as model size, capability description, and pricing; ii) **historical profiles** of past behavior, representing a model by the embedded queries it has previously solved, a scalar rating, or a latent factor fit by matrix factorization; iii) **learned embeddings** fit jointly with the context encoder (Chen et al., 2024; Feng et al., 2024; Zhuang et al., 2025); or iv) **verbalized descriptions** that name the candidates directly in the prompt of a fine-tuned LM router (Ong et al., 2024; Zhang et al., 2025).

Scoring function and decision rule. The scoring function g measures the compatibility between the encoded state and each candidate, and the decision rule d converts the resulting scores into a routing action. Instantiations of g track the encoders, from embedding similarity and bilinear products to a classification head (Ding et al., 2024; Stripelis et al., 2024), message passing over a query–model graph (Feng et al., 2024), and next-token logits that fold E_q, E_m, and g into one forward pass (Ong

Table 1: **Instantiation of the unified routing formulation for the three router families.** Each family is specified by its routing state s, its encoders E_q and E_m, the routing action given by g and d, and the learning signal $\mathcal{L}$ that optimizes quality and cost.

Family	**State** s	**Encoders** E_q, E_m	**Routing action** (scoring g, decision d)	**Learning signal** $\mathcal{L}$ (surrogate of Eq. 1)
Single-turn	(q)	$E_q(q),\ E_m(m)$	$a = \arg\max_{m \in \mathcal{M}}\ g\big(E_q(q),\ E_m(m)\big)$	fit g to per-candidate reward $\alpha \cdot \mathrm{perf}(y_m \mid q) - \beta \cdot c_m$
Multi-turn	(q, h_t)	$E_q(q, h_t),\ E_m(m)$	$a_t \sim d\big(\{g(E_q(q, h_t),\ E_m(m))\}_m\big)$	maximize episode return $\mathbb{E}_\tau\big[\alpha \cdot \mathrm{perf}(y \mid q) - \beta \cdot c(\tau)\big]$
Personalized	(q, u, h_t)	$E_q(q, u, h_t),\ E_m(m)$	$a = \arg\max_{m \in \mathcal{M}}\ g\big(E_q(q, u, h_t),\ E_m(m)\big)$	fit g to comparisons $m^+ \succ_u m^-$ observing perf_u

et al., 2024). For d, greedy $\arg\max$ is the default is optimal for Eq. 1 only when $\alpha = 1, \beta = 0$; routers whose g predicts quality alone instead rely on cost-aware decision rules instead threshold the predicted quality gap between a cheap and an expensive model (Ding et al., 2024), accept or escalate in cascades (Chen et al., 2023; Aggarwal et al., 2024), or sample for exploration in online settings (Dai et al., 2024), and multi-turn routers further equip d with the terminating action $\perp$ (Zhang et al., 2025).

Learning signal. The learning signal $\mathcal{L}$ specifies how the components above are fit toward Eq. 1, and differs across routers only in the form in which perf is observable. Non-parametric routers require no training and rely purely on stored interactions, supervised routers fit pointwise correctness labels harvested by running the candidate pool over benchmark queries, preference-based routers learn from pairwise comparisons such as human votes (Ong et al., 2024) or contrastive objectives that pull queries toward the models that solve them (Chen et al., 2024), and agentic routers directly optimize trajectory-level rewards with reinforcement learning (Zhang et al., 2025).

2.2 Automatic Evaluation of LLM Routing

Evaluating a router is substantially more demanding than evaluating a single model. Under Eq. 1, a router must be judged on both the quality of its answers and the cost spent to obtain them, and constructing its supervision requires knowing how every candidate performs on every query under task-specific metrics. In current practice, these elements are assembled manually for a single benchmark and fixed candidate pool (Hu et al., 2024; Huang et al., 2025b), requiring fresh engineering for every new task or candidate pool.

`RouterFactory` automates this process end-to-end with a three-stage pipeline: i) **Query Curation**: queries are sampled from source benchmarks, normalized into a unified schema, and split into training and test sets; ii) **Response Collection**: each query is dispatched to every candidate in the pool, which is declared in a single configuration file, and responses are collected together with their token counts; iii) **Metric Scoring and Pricing**: every response is scored with its task metric and priced from its token counts. The product is a dense query–model matrix of performance and cost that serves at once as routing supervision and as the test bed, so a new task or candidate pool enters through a configuration change rather than a re-engineered stack.

Evaluation reuses this path, except that a test query goes only to the candidate the router selects rather than to the whole pool. Every router therefore faces the same queries, candidate pool, and metrics, so measured differences reflect the routing policy rather than the surrounding stack. For the multi-turn and agentic families, every decomposition and aggregation call is priced into the trajectory cost.

Metrics. Task quality is assessed using built-in metrics aligned with standard benchmark conventions, including exact and close matching, multiple-choice accuracy, token-level F1, mathematical answer verification, and execution-based code evaluation. `RouterFactory` also supports optional LLM-based judging and exposes a weighted objective that balances performance and cost, allowing routers and trainers to target performance-first, cost-sensitive, or hybrid operating points.

3 xRouteBench: A Multi-Scenario Benchmark for LLM Routing

Existing routing benchmarks cover only a subset of the settings captured by the formulation in §2.1. RouterBench (Hu et al., 2024) precomputes candidate responses for single-turn text queries over a fixed candidate pool, but does not cover settings in which input-token costs dominate or only a subset of candidate models can process the input. RouterEval (Huang et al., 2025b) aggregates large-scale performance records but likewise focuses on single-turn text tasks and evaluates response quality

independently of inference cost. Recent vision–language routing benchmarks (Huang et al., 2025a; Ma et al., 2026) extend routing evaluation to image inputs, but remain limited to image question answering and do not cover video, long-context, or modality-selection settings. Preference data from Chatbot Arena (Zheng et al., 2023) provides population-level signals but lacks the persistent user context needed to supervise user-conditioned routing. To close these gaps, we construct xRouteBench to evaluate, under a unified cost-aware protocol, regimes in which routing decisions fundamentally differ: long-context inputs for which input-token costs dominate, image and video inputs that only a subset of candidate models can process, time-series inputs with multiple modality encodings, and tasks with user-specific quality preferences.

Design principle. All tasks are constructed by the `RouterFactory` data engine and share a common query schema, supervision format, and evaluation protocol. Each non-text asset is converted by a transformation script into a self-contained textual query, with an optional pointer to the source image, video, or time series. This separates routing from perception by ensuring that text-only and multimodal candidates receive the same textual input. For every query, the protocol jointly evaluates response quality and inference cost, exposing regimes in which input-token costs dominate answer-generation costs. Adding a new application requires only a transformation script and a registered metric.

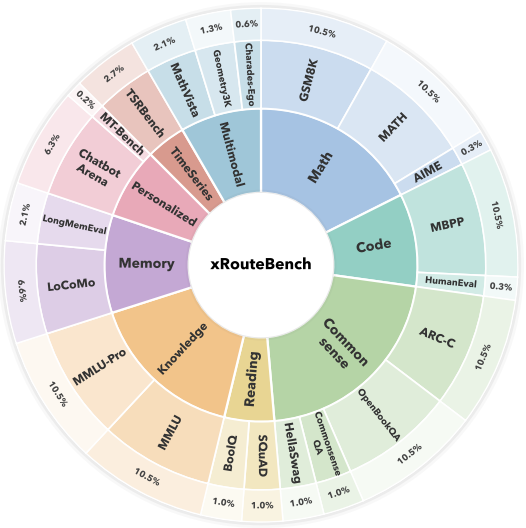


Figure 2: **Task composition of xRouteBench.** The benchmark covers generic LLM tasks, memory, vision, time-series, and personalized routing, with percentages indicating the proportion of test queries contributed by each dataset.

Tracks. xRouteBench spans five tracks comprising 4,767 instances, whose composition is shown in Figure 2 and detailed in Appendix C. Specifically, we include: (i) *Generic LLM Tasks* mix established knowledge and commonsense QA (MMLU (Hendrycks et al., 2020), MMLU-Pro (Wang et al., 2024), ARC-Challenge (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), CommonsenseQA (Talmor et al., 2019), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), SQuAD (Rajpurkar et al., 2016)), mathematical reasoning (GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), AIME), and code generation (MBPP (Austin et al., 2021), HumanEval (Chen, 2021)), the conventional single-turn text setting. (ii) *Memory* routes long-horizon conversational QA over hundreds of accumulated turns (LoCoMo (Maharana et al., 2024), LongMemEval (Wu et al., 2024)), where token cost is governed by the history rather than the answer. (iii) *Vision* covers image-grounded mathematical reasoning (Geometry3K (Lu et al., 2021), MathVista (Lu et al., 2024b)) and egocentric video understanding (Charades-Ego (Sigurdsson et al., 2018)). (iv) *TimeSeries* covers time-series reasoning (TSRBench (Yu et al., 2026b)), with each series rendered as numerical sequences. (v) *Personalized* draws open-ended prompts from Chatbot Arena and MT-Bench (Zheng et al., 2023), each tied to a user persona and scored by a persona-conditioned LLM judge, so its supervision is preference feedback rather than pointwise correctness.

4 THE `RouterFactory` LIBRARY

`RouterFactory` ties the formulation of §2.1, the evaluation protocol of §2.2, and xRouteBench into one executable system organized as six modules around a single query–model matrix (Figure 3). Its organizing principle is that the five components of a router are the only thing a user writes, while data construction, training, inference, evaluation, and deployment are shared infrastructure that operates on any router unchanged. Swapping a router, a candidate pool, or a training objective is therefore a configuration change rather than a reimplementation.

Data Engine. The data engine implements the three-stage construction pipeline of §2.2, turning a declared task list and candidate pool into the query–model matrix that supervises and tests every

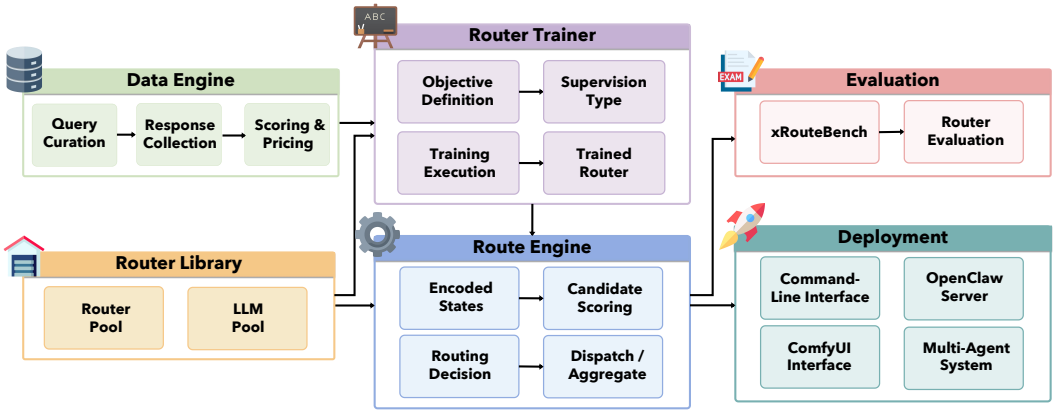


Figure 3: **Architecture of `RouterFactory`.** The system consists of six modules that support routing data construction, router implementation and training, inference, evaluation, and deployment.

router. Adding a task requires only a prompt template and a registered metric, and adding a candidate requires only its endpoint and per-token price.

Router Library. The library implements 16 routers spanning all three families under a unified `MetaRouter` interface, which hides mechanisms as different as nearest-neighbor retrieval in a kNN router and autoregressive decoding in a fine-tuned LM router behind one call. Adding a new router requires subclassing `MetaRouter` and implementing either `route_single` or its batched counterpart, `route_batch`. Within this method, the context encoder E_q, model encoder E_m, scoring function g, and decision rule d map a routing state to a routing action. A short YAML file specifies the router's candidate pool and objective weights, after which the router can be invoked by name using the same commands as any built-in method. Personalization and component ablations can then be performed with a one-line change, without forking the codebase. Figure 6 in Appendix B shows the complete code needed to design a new router.

Trainer. Training is decoupled from routing through a `BaseTrainer`. Its `loss_func` defines the learning signal $\mathcal{L}$ as a pointwise loss, pairwise loss, or trajectory-level reward, while its `train` loop uses this signal to optimize the router for the weighted objective in Eq. 1. A router and its trainer are paired but can be swapped independently, allowing the same scorer to be trained with different forms of supervision without modifying its routing code. Non-parametric routers bypass this module.

Route Engine. At inference time, the route engine drives any router through the same call, dispatching the query to the selected candidate. For multi-turn policies, it repeats the decision step until a termination action is produced and aggregates the collected responses into the final answer.

Evaluation. The evaluation module implements the evaluation protocol of §2.2, scoring each router on the same test queries, candidate pool, and metrics, and sweeping the trade-off weight β to trace its performance–cost frontier.

Deployment. In addition to training and inference commands, `RouterFactory` can expose any router as an OpenAI-compatible server that integrates with OpenClaw (OpenClaw, 2026) for deployment on messaging platforms such as Slack and Discord. A routing memory persists the interaction history h across turns, while a ComfyUI-based canvas supports code-free prototyping. The same router evaluated offline can therefore serve live single-agent and multi-agent traffic without modification.

5 Experiments

5.1 Experimental Setups

Benchmarks. We evaluate routers across the five xRouteBench tracks defined in §3: Generic LLM Tasks, memory, vision, time-series, and personalized. Together, they comprise eight test sets, with full statistics reported in Table 6. For each query in the memory track, we retrieve up to five memory items as context and score responses using token-level F1.

Table 2: **Results on xRouteBench under the performance-first setting** $(\alpha, \beta) = (1.0, 0.0)$**.** Scores are reported across the Generic LLM Tasks, memory, vision, and time-series tracks, together with their average. Following the original implementations where applicable, all multi-turn routers use Qwen2.5-3B-Instruct as the base model. The top two results are highlighted in **bold** and underline.

Router	Generic LLM Tasks	Memory		Vision			TimeSeries	Avg
		LoCoMo	LongMemEval	Geometry3K	MathVista	Video		
Rule-based baselines								
Smallest-LLM	57.55	25.44	36.77	27.87	35.00	**33.33**	49.61	37.94
Largest-LLM	70.29	26.59	35.57	37.70	33.00	22.22	45.67	38.72
Single-turn routers								
*k*NNRouter	71.37	25.24	**38.74**	31.15	41.00	29.63	51.97	41.30
SVMRouter	74.21	**27.64**	38.68	42.62	47.00	29.63	55.91	45.10
MLPRouter	68.12	26.78	32.27	27.87	34.00	29.63	56.69	39.34
MFRouter	67.23	24.49	34.91	40.98	29.00	22.22	51.97	38.69
EloRouter	64.15	25.70	37.27	**45.90**	**50.00**	25.93	**63.78**	44.68
Hybrid LLM	64.68	25.89	36.56	32.79	37.00	**33.33**	51.18	40.20
RouterDC	**80.56**	24.93	36.77	16.39	24.00	25.93	45.67	36.32
GraphRouter	80.54	25.94	33.93	42.62	**50.00**	22.22	62.99	**45.46**
CausalLM	66.90	25.40	37.60	24.60	34.00	**33.33**	45.70	38.22
Multi-turn routers								
Router-R1	35.64	24.60	17.28	14.75	18.00	22.22	23.62	22.30
*k*NN-MultiRound	13.99	24.70	18.32	16.39	30.00	25.93	33.07	23.20
LLM-MultiRound	12.98	24.60	17.44	14.29	31.03	25.93	30.33	22.37

LLM Candidates. The candidate pool contains 18 open-weight models served through two providers (i.e., Together API[1] and NVIDIA NIM API[2]), spanning 7B to 671B parameters and covering the Gemma (Team et al., 2024), Mistral and Mixtral (Jiang et al., 2023; 2024), Qwen (Yang et al., 2024; 2025), Llama (Grattafiori et al., 2024; Adcock et al., 2026), GPT-OSS (Agarwal et al., 2025), DeepSeek (Liu et al., 2024), rnj-1 (Callahan et al., 2026), and Cogito (Deep Cogito, 2025) families. The full pool with per-token pricing is listed in Table 9.

Implemented Routers. We evaluate the 16 routers implemented in `RouterFactory`, covering (i) **single-turn routers** (Smallest-LLM, Largest-LLM, kNNRouter, SVMRouter, MLPRouter, EloRouter, MFRouter, RouterDC, Hybrid LLM, GraphRouter, and CausalLM) together with two rule-based baselines that always select the smallest or largest model; (ii) **multi-turn routers** (Router-R1, kNN-based and LLM-based multi-round routers); and (iii) **personalized routers** (GMTRouter and PersonalizedRouter). Table 8 details each router and its citation.

Evaluation Protocol. We score each router by a weighted reward $\alpha \cdot \text{perf} - \beta \cdot \text{cost}$. We sweep five weight settings from the quality-only $(\alpha, \beta) = (1.0, 0.0)$ to the heavily cost-weighted $(0.2, 0.8)$. Multi-round and RL-based routers cannot optimize this weighted objective and are therefore run once under a single configuration. On the personalized track, answers are scored by a persona-conditioned LLM judge (DeepSeek-V3.1) as win, tie, or loss $(1, 0.5, 0)$, and the judge's own cost is excluded from the reported cost.

5.2 Main Results

Table 2 and Table 3 report the results under the performance-only setting. Based on these results, we have the following key observations:

No single router dominates across all tasks: The winning router varies across tasks. For example, RouterDC performs best on Generic LLM Tasks and SVMRouter on LoCoMo. Though GraphRouter attains the best average on xRouteBench, it does not consistently outperform other routers in all tasks.

Multi-turn routing does not consistently outperform single-turn routing: Across all benchmarks, multiple rounds of routing and aggregation provide no consistent gain over a single routing decision. For many queries, one well-chosen route is sufficient, whereas additional rounds introduce redundant information and computational overhead. Multi-turn routers also rely on a base model (Qwen2.5-3B-Instruct) to decompose queries and aggregate responses, making their performance sensitive to the capabilities of this model. These results highlight the need for better sufficiency estimation, early stopping, and more effective decomposition and aggregation.

[1] https://www.together.ai/
[2] https://build.nvidia.com/

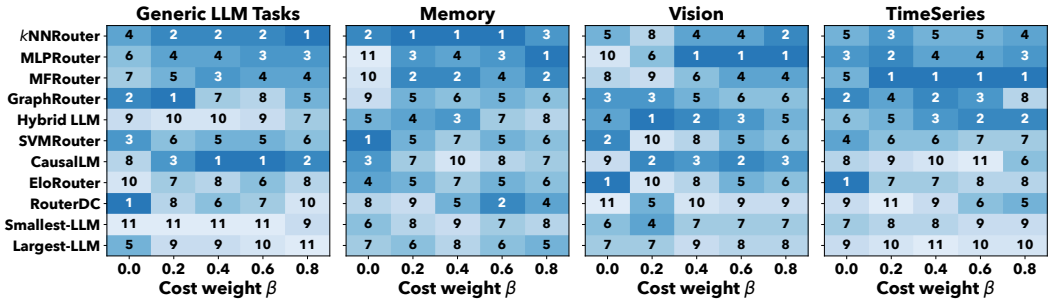


Figure 4: **Router rankings across the Generic LLM Tasks, memory, vision, and time-series tracks as the cost weight β increases.** Each cell gives a router's rank under the weighted performance–cost objective, with smaller rank values indicating better performance.

Conditioning on user context helps, but how it is modeled matters: Table 3 reports persona-judge accuracy on the personalized track. GMTRouter achieves the highest accuracy of 68.78, outperforming PersonalizedRouter (67.86) and the best user-agnostic router, EloRouter (66.40). The strong performance of both personalized methods confirms the benefit of conditioning routing decisions on user context, while GMTRouter's additional 0.92-point gain over PersonalizedRouter shows that the way user context is encoded and integrated remains important.

Table 3: **Performance comparison on the personalized track.** Top two results are highlighted in **bold** and underlined.

Router	Acc.	Router	Acc.
GMTRouter	**68.78**	RouterDC	56.44
PersonalizedRouter	<u>67.86</u>	MFRouter	54.39
EloRouter	66.40	MLPRouter	52.93
GraphRouter	65.23	*k*NNRouter	51.76
SVMRouter	65.08	CausalLM	46.78
Largest-LLM	58.05	Router-R1	45.46
Hybrid LLM	57.91	Smallest-LLM	42.53

5.3 Performance–Cost Trade-offs

No single router is best under every performance–cost trade-off. Figure 4 ranks the routers by reward within each category as the cost weight β grows, and the rankings shift dramatically along the sweep. RouterDC tops Generic LLM Tasks when only quality matters, yet falls to tenth of eleven under the most cost-sensitive setting; EloRouter leads Vision and TimeSeries at $\beta = 0$ but drops out of the lead once cost enters the objective. However, weakness at one operating point does not imply weakness at another, as MLPRouter sits near the bottom of Vision under the quality-first setting yet becomes the best choice there for every $\beta \geqslant 0.4$. Therefore, it is advisable to choose the router that matches the performance–cost requirements of the deployment at hand.

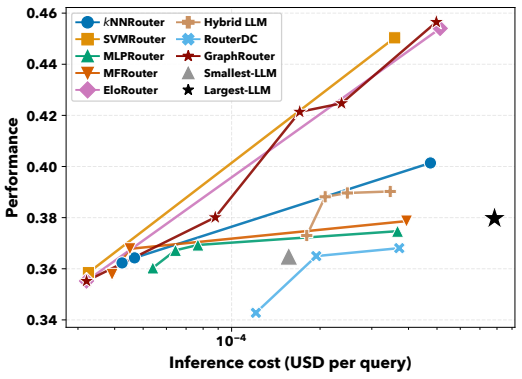


Figure 5: **Performance–cost trade-offs of routers averaged across the xRouteBench tracks.** Each point represents an operating setting with a different cost weight β, where higher performance and lower per-query inference cost are preferred.

Increasing inference cost leads to improved performance. Figure 5 presents the trade-off between performance and inference cost. For most routers, performance and cost exhibit a clear positive correlation, with the operating points rising from the low-cost to the high-cost end. This is because increasing the inference budget unlocks more powerful and expensive models, which perform better. Meanwhile, always calling the largest model incurs the highest cost yet delivers only mediocre performance and is dominated by the learned routes, since many queries that the largest model fails are solved by smaller and cheaper ones. This confirms that no single model covers all queries, which is exactly the headroom that routing exploits.

Table 5: **Router performance on the Generic LLM Tasks test split when each node in a multi-agent system is routed independently.** Results are reported across five coordination topologies, with the final column showing the average performance.

Router	Star	Tree	Graph	Chain	Plan-Exec-Sum	Avg
Largest-LLM	69.00	67.00	77.20	69.00	75.20	71.48
kNNRouter	74.80	78.60	78.60	76.60	71.80	76.08
SVMRouter	76.20	75.60	80.00	74.40	75.20	76.28
MLPRouter	75.40	76.60	76.80	78.00	71.40	75.64
MFRouter	75.40	74.20	**81.00**	**78.60**	73.20	**76.48**
EloRouter	73.80	72.40	78.60	76.60	75.20	75.32
GraphRouter	68.20	70.80	66.20	72.00	69.00	69.24
RouterDC	**77.60**	**79.60**	74.20	72.00	**76.20**	75.92

5.4 Routing in Deployment: Real Users and Multi-Agent Systems

The deployment layer of `RouterFactory` carries routers beyond static benchmarks. We study two settings it enables: routing for real users served through OpenClaw and routing inside multi-agent systems.

Table 4: **Router performance on held-out real-user sessions collected through the Slack deployment.** Accuracy measures how often each router's model selection agrees with the users' pairwise preferences.

Router	Acc.	Router	Acc.
PersonalizedRouter	**83.05**	RouterDC	65.25
EloRouter	82.20	kNNRouter	60.17
MLPRouter	78.81	kNN-MultiRound	60.17
SVMRouter	77.12	Smallest-LLM	55.08
Hybrid LLM	73.73	MFRouter	51.69
GMTRouter	70.70	Largest-LLM	41.53
GraphRouter	67.17	CausalLM	27.97

Routing for real users. Using the OpenClaw server of `RouterFactory`, we deploy the routing stack behind Slack and collect live preference feedback. Fifteen users contribute 40 sessions of 1 to 12 turns, totaling 234 pairwise records. For each query, two models are sampled from a pool of ten candidates, their answers are shown in randomized positions, and the user marks one as better or declares a tie. We split by session into 32 training sessions and 8 test sessions, train every router on the human training split, and score how often its selection matches the human preference on held-out sessions. Table 4 shows that PersonalizedRouter leads at 83.05, while the fine-tuned CausalLM router ranks last. The simulated ranking does not fully transfer, as GMTRouter, the winner under the persona judge, drops to sixth on real users, showing that it matters to validate personalized routers against real feedback.

Routing inside multi-agent systems. A multi-agent system (MAS) is conventionally instantiated with a single base model shared by every agent. We instead treat model choice as a per-agent decision, where a router receives the prompt of each functional node and selects the most suitable LLM for that call. Since node prompts differ substantially, covering planning, execution, and verification instructions, this setting stress-tests how well a router trained on ordinary queries generalizes. Following MultiAgentBench (Zhu et al., 2025) and GraphPlanner (Feng et al., 2026), we instantiate five coordination topologies, namely Star, Tree, Graph, Chain, and Plan-Exec-Sum, illustrated with their node roles in Appendix G. The learning-based routers are trained on the Generic LLM Tasks training split, each test query is then fed through the full MAS, and the final MAS answer is scored by the task metric. Table 5 shows that routing every node pays off, as six of the seven learned routers beat always selecting the largest model on average, with MFRouter attaining the best average of 76.48 against 71.48.

6 Conclusion

We introduced `RouterFactory`, a unified framework for LLM routing that casts single-turn, multi-turn, and personalized routing as instances of one sequential decision process, together with an automatic pipeline that constructs routing supervision and evaluation for new tasks and candidate pools, the multi-scenario xRouteBench benchmark, and an open-source library of 16 routers that deploys to real users and multi-agent systems. We hope `RouterFactory` will serve as a common foundation for developing, evaluating, and deploying LLM routers.

AI USE DISCLOSURE

In this work, we used generative AI tools only for polishing writing. We have reviewed all AI-assisted work.

REFERENCES

Aaron Adcock, Aayushi Srivastava, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pande, Abhinav Pandey, Abhinav Sharma, Abhishek Kadian, Abhishek Kumawat, Adam Kelsey, et al. The llama 4 herd: Architecture, training, evaluation, and deployment notes. *arXiv preprint arXiv:2601.11659*, 2026.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *Advances in Neural Information Processing Systems*, 37: 131000–131034, 2024.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Essential AI: Mike Callahan, Adarsh Chaluvaraju, Aleksa Gordić, Devaansh Gupta, Yash Jain, Philip Monk, Michael Pust, Tim Romanski, Peter Rushton, Ali Shehper, Divya Shivaprasad, Saurabh Srivastava, Anil Thomas, Alok Tripathy, Ameya Velingker, and Ashish Vaswani. Rnj-1-5-Instruct, 2026. URL https://huggingface.co/EssentialAI/rnj-1.5-instruct. Long-context Instruction-tuned model release.

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

Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, pp. 2924–2936, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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

Zhongjie Dai, Tao Feng, and Jiaxuan You. Personalizedrouter: Personalized llm routing via graph-based user preference modeling. *arXiv preprint arXiv:2511.16883*, 2025.

Deep Cogito. Cogito v2 preview: Deepseek 671b moe. https://huggingface.co/deepcogito/cogito-v2-preview-deepseek-671B-MoE, 2025. Hugging Face model card. Accessed: 2026-07-28.

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

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

Tao Feng, Haozhen Zhang, Zijie Lei, Peixuan Han, and Jiaxuan You. Graphplanner: Graph memory-augmented agentic routing for multi-agent llms. *arXiv preprint arXiv:2604.23626*, 2026.

Evan Frick, Connor Chen, Joseph Tennyson, Tianle Li, Wei-Lin Chiang, Anastasios N Angelopoulos, and Ion Stoica. Prompt-to-leaderboard. *arXiv preprint arXiv:2502.14855*, 2025.

Tao Ge, Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. Scaling synthetic data creation with 1,000,000,000 personas. *arXiv preprint arXiv:2406.20094*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

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

Zhehao Huang, Baijiong Lin, Jingyuan Zhang, Jingying Wang, Yuhang Liu, Ning Lu, Tao Li, and Xiaolin Huang. Vl-routerbench: A benchmark for vision-language model routing. *arXiv preprint arXiv:2512.23562*, 2025a.

Zhongzhan Huang, Guoming Ling, Yupei Lin, Yandong Chen, Shanshan Zhong, Hefeng Wu, and Liang Lin. Routereval: A comprehensive benchmark for routing llms to explore model-level scaling up in llms. *arXiv preprint arXiv:2503.10657*, 2025b.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*, 2022.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Yang Li. Rethinking predictive modeling for llm routing: When simple knn beats complex learned routers. *arXiv preprint arXiv:2505.12601*, 2025.

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

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. Routing to the expert: Efficient reward-guided ensemble of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1964–1974, 2024a.

Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-gps: Interpretable geometry problem solving with formal language and symbolic reasoning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 6774–6786, 2021.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations*, volume 2024, pp. 23439–23554, 2024b.

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

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2381–2391, 2018.

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 with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

OpenClaw. OpenClaw: Personal AI assistant. https://github.com/openclaw/openclaw, 2026. GitHub repository, accessed August 3, 2026.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 2383–2392, 2016.

Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. Large language model routing with benchmark datasets. *arXiv preprint arXiv:2309.15789*, 2023.

Gunnar A Sigurdsson, Abhinav Gupta, Cordelia Schmid, Ali Farhadi, and Karteek Alahari. Charades-ego: A large-scale dataset of paired third and first person videos. *arXiv preprint arXiv:1804.09626*, 2018.

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*, pp. 452–462, 2024.

Yihang Sun, Encheng Xie, Tao Feng, and Jiaxuan You. Gmtrouter: Personalized llm router over multi-turn user interactions. *arXiv preprint arXiv:2511.08590*, 2025.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, 2019.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Xinyuan Wang, Yanchi Liu, Wei Cheng, Xujiang Zhao, Zhengzhang Chen, Wenchao Yu, Yanjie Fu, and Haifeng Chen. Mixllm: Dynamic routing in mixed large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 10912–10922, 2025.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

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

Fangxu Yu, Tao Feng, Dehai Min, Lu Cheng, Ge Liu, and Tianyi Zhou. Tsrouter: Dynamic modality-model selection for time series reasoning. *arXiv preprint arXiv:2607.08940*, 2026a.

Fangxu Yu, Xingang Guo, Lingzhi Yuan, Haoqiang Kang, Hongyu Zhao, Lianhui Qin, Furong Huang, Bin Hu, and Tianyi Zhou. Tsrbench: A comprehensive multi-task multi-modal time series reasoning benchmark for generalist models. *arXiv preprint arXiv:2601.18744*, 2026b.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

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.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, et al. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8580–8622, 2025.

Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. Embedllm: Learning compact representations of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 76913–76926, 2025.

```python
from routerfactory.models import MetaRouter, BaseTrainer

class MyRouter(MetaRouter):                    # (E_q, E_m, g, d): state -> action
    def route_single(self, query):
        s = self.encode_state(query)           # context encoder  E_q
        scores = self.score(s, self.models)    # model encoder E_m + scoring g
        query["model_name"] = self.decide(scores)   # decision rule d
        return query

class MyRouterTrainer(BaseTrainer):            # learning signal  L
    def loss_func(self, outputs, batch):
        return my_objective(outputs, batch)    # pointwise / pairwise / RL reward

# Train and run through the same interface as every built-in router.
router  = MyRouter(yaml_path="my_router.yaml")
trainer = MyRouterTrainer(router)
trainer.train()
answer  = router.route_single({"query": "..."})
```

Figure 6: **The five components of the routing formulation map onto two classes in `RouterFactory`.** A router subclasses `MetaRouter` and implements `route_single` (or `route_batch`), where the context encoder E_q, model encoder E_m, scoring function g, and decision rule d turn a state into a selected model; the learning signal $\mathcal{L}$ lives in a `BaseTrainer` subclass.

A Related Work

LLM Routing. Prior routers can be read along the axes of our formulation. Single-turn routers differ mainly in how they encode a query and score candidates, from quality predictors that arbitrate between a weak and a strong model (Ding et al., 2024; Ong et al., 2024) and classifiers over frozen embeddings (Shnitzer et al., 2023; Stripelis et al., 2024; Li, 2025), to reward-guided rankings (Lu et al., 2024a), prompt-conditioned preference models (Frick et al., 2025), contrastive query–model matching (Chen et al., 2024), learned model embeddings (Zhuang et al., 2025), graph-based scorers (Feng et al., 2024), and online methods under bandit feedback (Dai et al., 2024; Wang et al., 2025). Multi-turn routers instead enrich the state across rounds, whether as cascades that escalate upon failed verification (Chen et al., 2023; Aggarwal et al., 2024) or as agentic routers that decompose a query and route sub-queries with reinforcement learning (Zhang et al., 2025; Feng et al., 2026). Personalized routers add the user to the state and learn from preference feedback (Sun et al., 2025; Dai et al., 2025). Each is developed in its own formalism and evaluated on its own stack; `RouterFactory` instead expresses them as instantiations of a single sequential decision process behind one interface, so single-turn, multi-turn, and personalized routers meet on the same performance–cost frontier.

Routing Benchmarks and Evaluation. RouterBench (Hu et al., 2024) precomputes candidate responses over a fixed pool, RouterEval (Huang et al., 2025b) aggregates large-scale performance records for routing study, preference data from Chatbot Arena has served as routing supervision (Ong et al., 2024), and recent benchmarks extend routing to vision–language pools (Huang et al., 2025a; Ma et al., 2026). However, existing benchmarks each target a single scenario, one-shot text or image QA, with fixed pools, and provide no pipeline for constructing supervision on new tasks or candidate sets. `RouterFactory` closes this gap with automatic supervision construction and cost-aware evaluation over configurable pools, and xRouteBench spans generic LLM tasks, memory-augmented, vision (image and video), time-series, and personalized scenarios under one protocol.

B Library Details

Figure 6 shows the complete code needed to register a new router in `RouterFactory`.

C BENCHMARK DETAILS

Table 6 lists the eight test sets of xRouteBench, their categories, sizes, and metrics, totaling 4,767 test queries. The Generic LLM Tasks track further decomposes into 13 subtasks, and the memory datasets are retrieved with top-5 RAG over turn pairs.

Table 6: **The eight test sets of xRouteBench.** Sizes are the number of test queries; metrics are exact match (EM), multiple-choice accuracy (MC), token-level F1, execution-based code pass rate, math answer matching, and a persona-conditioned LLM judge.

Category	Test set	Content	#Test	Metric
Generic LLM Tasks	Generic mix	13 subtasks	3,729	EM/MC/F1
Memory	LoCoMo	long-conversation QA	314	F1
	LongMemEval	long-term memory QA	101	F1
TimeSeries	TimeSeries	7 reasoning skills	127	MC
Vision	Geometry3K	geometry math (image)	61	EM
	MathVista	visual math reasoning	100	EM/MC
	Charades-Ego	egocentric video	27	EM
Personalized	Chatbot Arena / MT-Bench	preference prompts	308	LLM judge
Total			**4,767**	

Table 7 details the 13 subtasks that make up the Generic LLM Tasks track.

Table 7: **Composition of the Generic LLM Tasks track.** The table lists the 13 subtasks, their target skills, and the number of test queries, totaling 3,729 examples.

Subtask	Skill	#Test
MBPP	Code generation	500
MATH	Mathematical reasoning	500
GSM8K	Mathematical reasoning	500
MMLU-Pro	Knowledge QA	500
ARC-Challenge	Knowledge QA	500
MMLU	knowledge QA	500
OpenBookQA	Commonsense QA	500
CommonsenseQA	Commonsense QA	50
BoolQ	Reading Comprehension QA	50
SQuAD	Reading comprehension	50
HellaSwag	Commonsense QA	50
HumanEval	Code Generation	16
AIME (2020–2024)	Competition Math	13

C.1 GENERIC LLM TASKS

The Generic LLM Tasks track is deliberately a mixture rather than a single task family. It places knowledge questions, commonsense inference, reading comprehension, mathematical reasoning, and code generation behind the same routing interface. All examples use text input, but an ARC-Challenge question, an AIME problem, and a Python synthesis task reward very different capabilities and impose different output constraints. The track therefore asks whether a router can recognize these distinctions within an apparently uniform text setting, rather than defaulting to one model for every natural language request. Table 7 lists the 13 source benchmarks and their test sizes.

We serialize each example as one fixed text query before collecting candidate responses. A system instruction specific to the task states the required response format, and the user problem follows it. Choice items include their options. Mathematics items request a final answer in the expected form. Code generation items provide the programming task and tests, while SQuAD supplies its passage and question. We score each response with the source benchmark's native objective. We use choice

accuracy for knowledge and commonsense tasks, token-level F1 for SQuAD, answer matching for GSM8K, MATH, and AIME, and execution-based pass rates for MBPP and HumanEval.

C.2 Long Context Conversational Memory

The Memory track fixes retrieval and asks a narrower routing question. Given the same retrieved conversational evidence, which model is most likely to use it correctly? LoCoMo and LongMemEval contain questions whose supporting facts may be separated from the query by many turns or sessions. Some can be answered by recovering one explicit fact, whereas others require combining facts across sessions or resolving a later update against an earlier statement. This distinction is useful for routing because it separates the cost of carrying a long history from the ability to interpret the evidence selected from it.

We construct every memory query through the same retrieval pipeline. Each conversation history is divided into adjacent turn pairs that retain speaker and date information. A fixed Contriever (Izacard et al., 2022) encoder embeds the question and the turn pairs, and the five most similar pairs are inserted into a brief answer prompt. LongMemEval additionally includes the date of the question, which provides the temporal reference needed for its time-sensitive items. Every candidate therefore receives the same retrieved evidence for a query. Differences in score reflect its use of that evidence rather than a different retrieval result. We evaluate both datasets with token-level F1.

C.3 Time Series Pattern Reasoning

The TimeSeries track is built from TSRBench and covers anomaly detection, similarity analysis, noise understanding, pattern recognition, inductive reasoning, causality analysis, and event prediction. These problems are numerical, but they often require recognizing structure that is easier to see as a shape than as a list of values. A local spike may signal an anomaly, while periodicity, changes in trend, or agreement between two series emerge over a longer range. The track asks whether a router can distinguish the models that handle these different forms of temporal reasoning well.

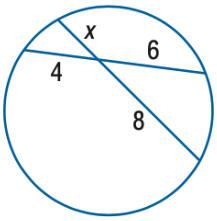
We convert every series into a common text query before collecting candidate responses. First, we render each series as a line chart and pass the chart to a fixed Gemma-3-27B-IT (Team et al., 2025) captioner, which produces a paragraph description of its visible temporal pattern. We append these descriptions and the raw numerical values to the original question and its answer options. The raw values preserve exact numerical evidence and are truncated after 200 values when necessary, while the descriptions make higher-level structure explicit. The charts are used only in this offline captioning step. Every candidate model receives the same text query rather than an image. Models return an option letter, and we score the responses with multiple-choice accuracy.

C.4 Visual Mathematical Reasoning

The Visual Mathematical Reasoning track routes image-grounded mathematics drawn from Geometry3K (Lu et al., 2021) and MathVista (Lu et al., 2024b), where a geometry diagram or a scientific figure carries information the question depends on. The strongest candidate on these problems can differ from the strongest on text mathematics, so the track probes whether a router follows per-model strength as the query type changes. We render each problem to text before it reaches the pool, describing its image with a fixed vision-language model and appending the description to the question, so every candidate reasons over one identical query. Rendering the image once holds perception constant across the pool and turns visual mathematics into a query the whole pool can answer, so the track scores the reasoning quality of each candidate on the same input. These rendered queries read as machine-written descriptions of a figure, a distribution that departs from the natural questions the routers are trained on, so the track also tests how well a router generalizes when the query type shifts.

A single frozen vision-language model, Gemma-3-27B-IT by default, produces every description, so all candidates see the same rendering of a given image. Its prompt asks it to report every visible number, symbol, angle, length, and relationship and to withhold the solution, which keeps the answer out of the query and leaves the reasoning to the routed model. The returned text is appended to the original problem, from which Geometry3K's image placeholder is removed, to form the routed query. Geometry3K is graded by exact answer matching against its numeric answer, and MathVista

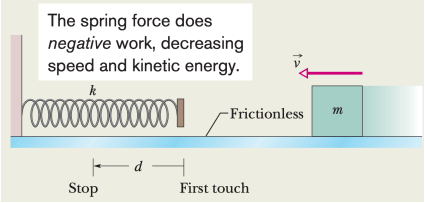
is graded by exact match on its open-ended items and by multiple-choice accuracy on its choice items, following the question-type field of each query. Every query keeps its ground-truth answer and, where present, its answer choices, so the scored responses populate the same query–model matrix of quality and cost that supervises every other track (§2.2). Figure 7 shows one example from each dataset with the description Gemma-3-27B-IT produces for it.



(a) Geometry3K, which asks for x.

Gemma-3-27B-IT description

The diagram shows a circle with two intersecting chords. One chord is divided into segments of length 4 and 6, and the other into segments of length x and 8. The label x marks one of these segments, and the two chords cross at a point inside the circle.



(b) MathVista, which asks for the spring compression d.

Gemma-3-27B-IT description

A block of mass m slides to the left along a horizontal frictionless surface with velocity v and approaches a coiled spring of constant k fixed to a wall. Labels mark where the block first touches the spring and where it stops, and d denotes the distance between these two points. A caption states that the spring force does negative work, decreasing speed and kinetic energy.

Figure 7: **One example from each dataset in the Vision track, shown with the description that Gemma-3-27B-IT produces for its image.** Geometry3K (a) provides a geometry diagram, and MathVista (b) provides a scientific figure. Each description is appended to the problem text to form the query the router sees.

C.5 Multi-View Video Recognition

The video tasks in vision track build on Charades-Ego (Sigurdsson et al., 2018), which records each activity simultaneously from a first-person egocentric camera worn by the actor and a third-person exocentric camera facing the scene (Figure 8). The two viewpoints carry different evidence, and a deployment often captures only one of them, so the routing decision here depends on which views a query provides. We describe every available view in text and route the merged description, so the whole candidate pool answers the same query whether one view or both are present. This makes the video track the regime in xRouteBench where the available modality varies from query to query, and it tests whether a router still selects the right model when the visual evidence is partial.

We build three classification tasks from the annotations, predicting the activity, the verb, or the object of the depicted action, each answered by a compact identifier drawn from the task label inventory, with the verb and object labels recovered from the Charades action mapping. For each paired clip we match the two views by their shared identifier and select the action segment whose egocentric and exocentric occurrences overlap most closely in time, which keeps both views on the same moment, and we then keep the first-person view, the third-person view, or both at random to populate the single-view and dual-view regimes. From each retained view we sample frames inside the aligned window and pass them to the vision-language model, which returns a structured description of the actor's motion, the handled objects, and the scene. These descriptions are merged into one query that states the task, lists the candidate identifiers with their names, and requests an identifier as the answer, holding every candidate to the same text input.



Figure 8: **A multi-view sample from Charades-Ego (Sigurdsson et al., 2018), recorded from a first-person egocentric camera (top) and a third-person exocentric camera (bottom).** Each row is a viewpoint and each column is a sampled time step. Our transformation samples these frames, describes the available views with a vision-language model, and routes the resulting text query, randomly withholding one view to form single-view and dual-view queries.

C.6 Personalized Dialogue Preference

The Personalized track combines open-ended dialogue from MT-Bench and Chatbot Arena. Each query retains its original message sequence, including preceding system, user, and assistant messages. MT-Bench contributes instruction-following conversations in which a later turn depends on an earlier exchange, while Chatbot Arena contributes natural user requests across diverse conversational settings. Rather than assuming that one answer is universally better, the track constructs supervision from the preference that a particular user profile would express after seeing two answers.

The user profiles used for preference elicitation are drawn from PersonaHub (Ge et al., 2024). We sample 200 personas from this resource for data collection. Figure 9 presents ten examples from the full persona set used during data collection.

Ten example personas used to condition the judge

1. A 71-year-old retired nurse from Italy, volunteering in hospice care and advocating for compassionate end-of-life support.
2. A 54-year-old divorced mother from Spain, running a successful winery and promoting sustainable viticulture practices.
3. A 63-year-old retired teacher from China, teaching calligraphy and preserving the art form for future generations.
4. A 68-year-old retired engineer from Japan, practicing ikebana and teaching the art to younger generations.
5. A 21-year-old photographer from Paris who spends weekends volunteering.
6. A 34-year-old scientist from London who is a social media influencer.
7. A 41-year-old scientist from London who loves hiking.
8. An 87-year-old World War II veteran from Poland, sharing stories of his experiences and advocating for peace.
9. A 31-year-old social worker from Colombia, supporting victims of domestic violence and fighting for gender equality.
10. A 23-year-old aspiring musician from Brazil, fusing traditional and modern sounds and promoting cultural exchange through music.

Figure 9: **Ten examples from the 200 PersonaHub personas sampled for preference collection.** For each comparison, DeepSeek-V3.1 receives one selected persona as its role specification when judging the two candidate answers. The candidate models do not receive the persona.

For each dialogue, we sample two models at random from the candidate pool of 18 models and generate one response from each under the same conversation history. DeepSeek-V3.1 is conditioned on the selected persona to act as the judge, comparing the two answers and returning a preference for either response or a tie. The persona is supplied to the judge only, not to either candidate model. Each comparison is converted into a pairwise supervision record whose label is the final judged user preference. The router learns from these preference outcomes rather than from the persona

Table 8: **Built-in routers in `RouterFactory`, grouped by routing family.** For each method, the table summarizes the information available to the routing decision (State) and the rule used to select or aggregate candidate models (Selection).

Router	State	Selection
	Rule-based baselines	
Smallest-LLM	candidate parameter counts	always selects the smallest candidate
Largest-LLM	candidate parameter counts	always selects the largest candidate
	Single-turn routers	
kNNRouter	query embedding and nearby logged queries	votes over the models preferred by nearest neighbors
SVMRouter	query embedding	kernel classifier predicts a candidate
MLPRouter	query embedding	MLP classifier predicts a candidate
MFRouter	query and model latent factors	ranks candidates by their interaction score
EloRouter	logged pairwise model outcomes	always selects the highest-rated candidate
RouterDC	query and candidate representations	contrastive query–model matching score
Hybrid LLM	query embedding and a small-/large model pair	predicts whether the small model is sufficient
GraphRouter	query–model interaction graph	predicts performance on query–model edges
CausalLM Router	textual query and candidate list	generates the selected model name
	Multi-turn routers	
Router-R1	query and accumulated search results	iteratively searches specialists or terminates and aggregates
kNN-MultiRound	sub-queries and their embeddings	routes each sub-query with kNN and aggregates the answers
LLM-MultiRound	textual query, decomposition, and candidate list	an LLM chooses routes for sub-queries and aggregates
	Personalized routers	
GMTRouter	user, session, query, model, and response interactions	predicts user-conditioned model preference
PersonalizedRouter	user features, task description, query, and model	predicts preference for a user–query pair

description itself. This construction keeps the candidate responses comparable while making the routing target sensitive to the user utility represented by the judge.

D Router Details

The routers in `RouterFactory` share the candidate pool and task interface described above, but they expose different information to the routing decision. Rule-based baselines select from a fixed property of the pool, while single-turn methods make one model selection from the query and logged routing outcomes. Multi-turn methods additionally condition on intermediate calls and may decide whether further routing is useful. Personalized methods add a user and interaction history, so their target is the model a particular user is most likely to prefer rather than one model that is best on average. Table 8 summarizes the 16 built-in implementations along these differences.

D.1 Rule-Based Baselines

Smallest-LLM and Largest-LLM provide fixed reference points for the learned routers. They ignore the query and always select the candidate with the smallest or largest declared parameter count, respectively. These rules make no attempt to identify per-query model strengths, but they expose the two simple deployment policies against which query-aware routing should be compared: consistently favoring the smallest available model or consistently favoring the largest one.

D.2 Single-Turn Routers

Single-turn routers terminate after one model selection, so their differences lie in how they represent a query and score candidates. EloRouter is query-independent, but it replaces a fixed parameter-count rule with a global ranking estimated from pairwise outcomes in the routing data. It therefore captures which model is strongest on average while deliberately discarding the variation between individual queries.

kNNRouter, SVMRouter, and MLPRouter instead make the selection query-specific from its embedding. kNNRouter retrieves similar training queries and transfers their observed best-model choices through a vote or distance-weighted vote, requiring no fitted decision function beyond the stored examples (Li, 2025; Shnitzer et al., 2023). SVMRouter and MLPRouter fit discriminative boundaries over the same embedding space, using a kernel classifier and a multilayer perceptron, respectively. MFRouter takes a different view of the logged query–model matrix: it learns latent representations for both sides and selects the model with the strongest query–model interaction. These methods all learn from per-query outcomes, but differ in whether the candidate is represented by neighboring solved examples, a classifier label, or a learned latent factor.

The remaining single-turn methods enrich this compatibility score in different ways. RouterDC learns query–model matching with dual contrastive objectives over query–model, query–query, and cluster-level relations (Chen et al., 2024). GraphRouter instead passes information over a query–model interaction graph and predicts the quality of an unobserved query–model edge (Feng et al., 2024). CausalLM Router verbalizes the query and the available candidates, then fine-tunes a causal language model to generate the selected model name (Ong et al., 2024). In this case, the query representation, candidate representation, and selection score are combined within the language model rather than implemented as separate embedding modules.

Hybrid LLM and AutoMix are cost-aware two-model cascades. Hybrid LLM learns from the quality gap between the smallest and largest candidates, then uses a thresholded prediction to decide whether the smaller model is sufficient (Ding et al., 2024). AutoMix first obtains a draft from the smaller model and a verification signal for that draft; it retains the draft when the signal is reliable and otherwise escalates the query to the larger model (Aggarwal et al., 2024). We list both methods with the single-turn family because they return one final model answer through a fixed cascade, rather than repeatedly choosing among arbitrary candidates and aggregating an open-ended history. Their extra calls are nevertheless part of the inference cost.

D.3 Multi-Turn Routers

Multi-turn routers treat intermediate responses as part of the routing state. Router-R1 is a pretrained routing agent that reasons over the query and the results gathered so far. At each step it can issue a `<search>` call to a suitable specialist, incorporate the returned evidence, or terminate and produce an aggregate answer; its policy is trained with trajectory-level reinforcement learning rather than pointwise best-model labels (Zhang et al., 2025). The cost of its reasoning, search, and final aggregation calls is included in the routed trajectory.

The two multi-round baselines use decomposition more explicitly. kNN-MultiRound first breaks a complex query into a small set of sub-queries, applies the kNN routing rule to each one, and combines the resulting answers. LLM-MultiRound uses an LLM to express both the decomposition and the model choice in text, then executes the selected sub-queries and aggregates their outputs. Both methods can assign different candidates to different parts of a problem, but they also introduce decomposition and aggregation calls that a one-shot router does not pay for.

D.4 Personalized Routers

Personalized routers replace the single global notion of quality with a user-conditioned preference. GMTRouter represents users, sessions, queries, candidate models, and responses as nodes in a heterogeneous interaction graph, allowing a decision for the current query to draw on earlier interactions from the same user (Sun et al., 2025). PersonalizedRouter similarly uses a graph-based scorer, while explicitly incorporating user features and task descriptions alongside the query and candidate representations (Dai et al., 2025). Both methods are trained from comparisons between candidate answers, so a high score means that a model is predicted to be preferred for this user and query, not simply that it has the highest population-average task score.

E Candidate Pool and Pricing

Table 9 lists the 18 candidate models and their per-token prices. Prices are in USD per 1M tokens, with input and output priced separately; the blended average is their mean. The input price spans a roughly 25× range, from $0.05 to $1.25 per 1M tokens. The 16-model no-Cogito pool drops the most expensive model (cogito-v2-1-671b).

Table 9: **The 18 candidate LLMs, sorted by blended average price.** Prices in USD per 1M tokens.

#	Model	Params	Input	Output	Service
1	gemma-2-9b-it	9B	0.10	0.10	NVIDIA
2	llama-3-8b-instruct-lite	8B	0.10	0.10	Together
3	gpt-oss-20b	20B	0.05	0.20	Together
4	rnj-1-instruct	8B	0.15	0.15	Together
5	mistral-7b-instruct-v0.3	7B	0.20	0.20	NVIDIA
6	mistral-small-3-24b-instruct	24B	0.10	0.30	Together
7	qwen2.5-7b-instruct	7B	0.20	0.20	NVIDIA
8	qwen2.5-7b-instruct-turbo	7B	0.30	0.30	Together
9	gpt-oss-120b	120B	0.15	0.60	Together
10	llama-4-maverick	402B	0.27	0.85	Together
11	mixtral-8x7b-instruct-v0.1	46.7B	0.60	0.60	NVIDIA
12	qwen3-next-80b-a3b-instruct	80B	0.15	1.50	Together
13	qwen3-coder-next	80B	0.50	1.20	Together
14	llama-3.3-70b-instruct-turbo	70B	0.88	0.88	Together
15	llama3-70b-instruct	70B	0.90	0.90	NVIDIA
16	deepseek-v3.1	671B	0.60	1.70	Together
17	mixtral-8x22b-instruct-v0.1	140.6B	1.20	1.20	NVIDIA
18	cogito-v2-1-671b	671B	1.25	1.25	Together

F Human Preference Collection on Slack

The human preference dataset of §5.4 is collected through a Slack application built on the OpenClaw server of `RouterFactory`, as illustrated in Figure 10. A user asks a question directly in Slack, and the system samples two LLMs at random from the ten-candidate pool and generates one answer with each. The conversation view then presents the two responses as anonymized Answer A and Answer B, with their positions randomly shuffled, and the user clicks a button to mark A better, B better, or a tie. In multi-turn sessions the user follows up freely, and every turn is labeled in the same way. Fifteen users contributed 40 sessions and 234 pairwise preference records in total.

G Multi-Agent Topologies

Table 10 summarizes the five coordination topologies used in §5.4. Star, Tree, Graph, and Chain follow MultiAgentBench (Zhu et al., 2025), and Plan-Exec-Sum follows the static router design of GraphPlanner (Feng et al., 2026) with width 3 and depth 1 (see illustration in Figure 11). Star and Plan-Exec-Sum differ in two respects: in Star, the planner produces free-form subtasks and consolidates the actors' outputs itself, whereas in Plan-Exec-Sum the planner decomposes the query

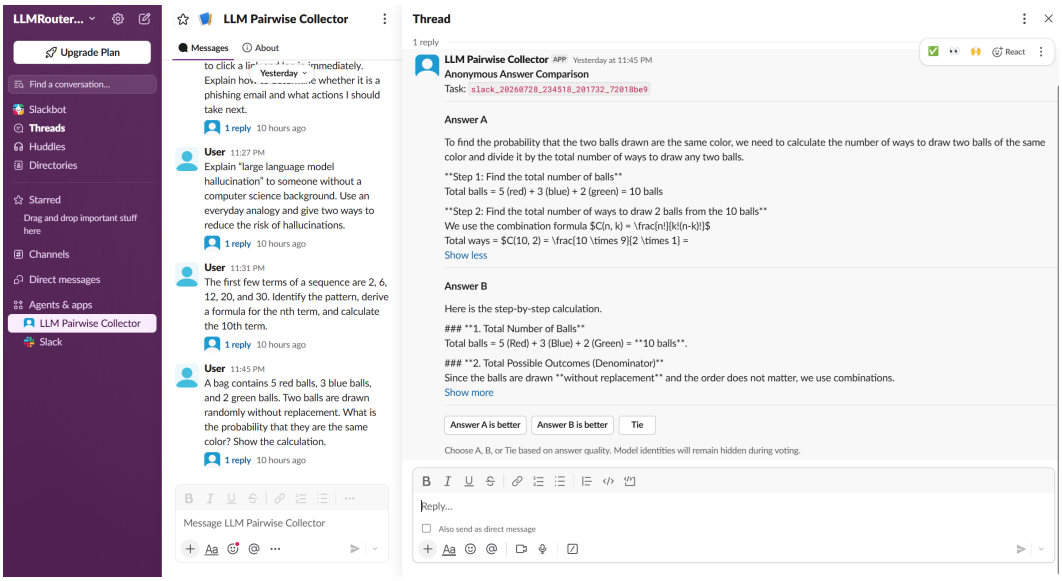


Figure 10: **Slack interface for collecting pairwise user preferences.** Two anonymized model responses are shown as Answer A and Answer B in randomized order, and users select A, B, or a tie for each interaction turn.

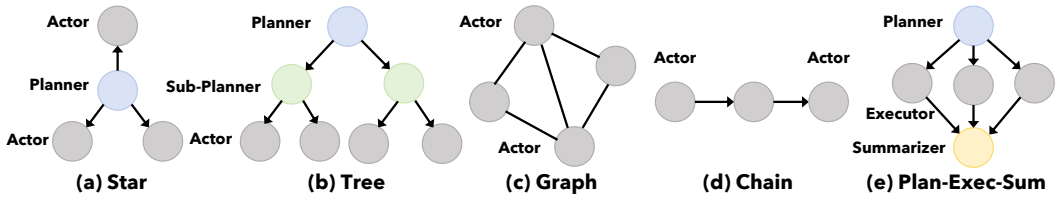


Figure 11: **Representative multi-agent system architectures and coordination topologies:** (a) star-based centralized coordination, (b) hierarchical tree-based delegation, (c) graph-based peer interaction, (d) sequential chain collaboration, and (e) planner–executor–summarizer workflow.

into three atomic sub-queries and a dedicated summarizer merges the executors' answers. Every node is replaced by a router that receives the node's prompt and selects a model from the candidate pool for that call. We adopt the topology of GraphPlanner without training its planner, since the training requires live answers from candidate models that have since been retired.

Table 10: **The five multi-agent topologies, their structures, and the number of LLM calls per query.**

Topology	Structure	LLM calls
Star	planner decomposes → 3 actors in parallel → planner consolidates	6
Tree	root planner → 2 sub-planners refine → 2 actors → root consolidates	7
Graph	3 actors answer independently → one full-communication revision round	7
Chain	3 agents relay sequentially, each verifying and improving the previous answer	4
Plan-Exec-Sum	planner emits 3 atomic sub-queries → 3 executors → summarizer merges	6

H PROMPT USAGE

This section reports the fixed prompt templates used to construct and evaluate xRouteBench, as well as the templates used by routers that make language-model calls internally. Curly brackets denote instance-dependent fields. Each prompt is presented in a titled box, combining the fixed instruction and the instance fields in the order in which the model reads them, rather than splitting them into

separate system and user blocks. In the implementation, the fixed instruction is sent as a system message when the API supports that role; otherwise, the two parts are concatenated with `[System Instruction]` and `[User Query]` delimiters. Thus, all candidate models receive the same logical prompt for a given query.

H.1 Benchmark-Query Templates

Generic LLM Tasks. The 13 Generic LLM Tasks use seven task-specific templates. The multiple-choice instruction is shared by MMLU, MMLU-Pro, BoolQ, and HellaSwag; CommonsenseQA, OpenBookQA, and ARC-Challenge use the same instruction but retain their source choice formatting.

Standard multiple choice

```
Answer the following multiple-choice question by selecting the correct option (A, B, C,
or D). You MUST put your final answer letter in a parenthesis.

Question: {question}
Options: A. {choice_1} B. {choice_2} C. {choice_3} D. {choice_4}
```

Source-formatted multiple choice

```
Answer the following multiple-choice question by selecting the correct option (A, B, C,
or D). You MUST put your final answer letter in a parenthesis.

Question: {question} (A) {choice_1} (B) {choice_2} (C) {choice_3} ...
```

GSM8K

```
Answer the following math question step by step.

Question: {question}
```

MATH and AIME

```
Answer the following math question.  Make sure to put the answer (and only answer)
inside \boxed {}.

Question: {question}
```

MBPP

```
You are an expert Python programmer. Implement the function with no irrelevant words
or comments. Put your code in this format: [BEGIN] <Your Code> [Done]

Task: {task_description}
Your code should pass these tests: {tests}
```

HumanEval

```
You are an expert Python programmer. Implement the function body only. Do not repeat
the function signature or docstring. Put the code in this format: [BEGIN] <Your Code -
function body only> [Done]

Complete the following function: {function_signature_and_docstring}
```

SQuAD

```
Answer the question using the provided context. If the answer is not in the context,
output noanswer.
{source query containing the context and question}
```

Conversational memory. For both memory datasets, Contriever retrieves the top five turn-pair chunks before the following prompt is formed. The retrieval result is fixed across candidate models for each query. LoCoMo uses:

LoCoMo

```
You answer questions about conversation history. Reply with a short phrase only.
Below is relevant information from the conversation history:
{retrieved turn-pair chunks}

Based on the above context, write an answer in the form of a short phrase for the
following question. Answer with exact words from the context whenever possible.
Question: {question} Short answer:
```

If no chunk is retrieved, the context field is replaced by `No relevant information available.` LongMemEval uses the analogous template, with its question date made explicit:

LongMemEval

```
You answer questions based on chat history. Reply with a short phrase only.
I will give you several history chats between you and a user. Please answer the question
based on the relevant chat history.

History Chats:
###### Session {index}: Session Content: {retrieved chunk}
...
Current Date: {current_date}
Question: {question}
Short Answer:
```

Here, an empty history is represented by `No relevant chat history available.`

Visual mathematical reasoning. Geometry3K and MathVista first use one frozen vision-language model to turn the image into text. The captioning call is:

Image captioning

```
Describe the image for solving the problem. Include all visible text, numbers, symbols,
angles, lengths, and geometric relationships. Be concise and factual. Do not include
introductory phrases.
{image}
```

The resulting caption is then given to every candidate model with the following answer prompt:

Visual mathematical reasoning

```
You are an expert in solving math and geometry problems. Think step-by-step, and when
you are ready to provide the final answer, you MUST enclose it in \boxed {}.

For example, if the answer is 42, write \boxed {42}. If it is a multiple choice question
and the option is C, write \boxed {C}.

Image Description: {caption}
Answer the following question based on the provided image description: {question}
```

Time-series reasoning. We first render every series as an image and request a one-paragraph caption. The captioner receives the following system instruction and one instruction sampled uniformly from 20 equivalent phrasings (the first phrasing is shown):

Time-series captioning

```
You are a time series captioner.
Write a paragraph that analyzes the time series.
{time-series plot}
```

The other phrasings differ only in wording (e.g., “Create a detailed description of the time series in one paragraph”) and impose the same one-paragraph description requirement. The answer prompt combines the caption with the raw values:

Time-series reasoning

```
You are a time series analysis expert. Answer the multiple-choice question by providing
the correct option letter (e.g., A, B, C, or D). Be concise.

1. Time Series Description: (1) {series_name}: {caption}, ...
2. Raw time series: (1) {series_name}: [{values}], ...

Based on one of the following pieces of information to reason the final answer:
{original_question}
```

For perception and causality-analysis items, this two-source block replaces the <ts><ts/> marker in the source question; it is appended for the other two task types.

Multi-view video recognition. Five frames from each available egocentric or exocentric view are summarized independently before classification. The video captioner receives:

Video captioning

```
You are given multiple frames sampled from a short time window of a video.

Return ONLY a JSON object with the following keys (no extra text):
 "motion": string,
 "objects": [string],
 "summary": string

Definitions:

motion: describe what the person is doing (hands/arms/body) in one short phrase.
objects: list the main target objects the person is acting on / interacting with (not
background clutter).
summary: a short description of the scene in one sentence.
{video frames}
```

Each candidate model then receives the two structured captions, the relevant label inventory, and this classification prompt:

Video classification

```
You are a helpful AI assistant tasked with answering questions about videos based on
descriptions.
Read the provided text carefully and return ONLY the ID (e.g.  c001, v003, o012)
representing the correct answer.

Task: Identify the {activity/verb/object} from video descriptions.
Determine the correct id using the structured descriptions below.

First-person view description: {ego_caption}
Third-person view description: {exo_caption}
Available {activity/verb/object}s: {id}: {label} ...

Provide your answer as the ID (e.g., c001, v003, o012).
```

Personalized dialogue preference. For MT-Bench, the original multi-turn user/assistant message sequence is preserved; for Chatbot Arena, the original user message is preserved. No additional fixed instruction is inserted before a candidate answers. Two candidate responses are then compared by a judge conditioned on a sampled persona:

Persona preference judge

```
You are simulating the following user persona: {persona}.
As this persona, evaluate which assistant response you would personally prefer.

[User Question] {question}
[Assistant 1] {answer_1}
[Assistant 2] {answer_2}

Output ONLY one of the following tokens exactly: 1 2 Tie Do not output anything else.
```

H.2 Router-Internal Templates

The preceding templates construct the query–model response matrix used by all routers. Most single-turn routers consume this matrix without another LLM prompt. The following templates are used only by the routers that perform additional language-model calls during routing or aggregation.

kNN-MultiRound and LLM-MultiRound. Both multi-round routers decompose a query, obtain sub-answers, and aggregate them. kNN-MultiRound selects a model for each sub-query with nearest-neighbor lookup, whereas LLM-MultiRound asks an LLM to choose the model. Their shared decomposition template is:

Multi-round decomposition

```
Given the query '{query}', decompose it into as many as 4 meaningful sub-queries
(minimum 1, maximum 4).
Try to cover the full scope of the original query by breaking it down into multiple
specific and distinct sub-tasks whenever possible.

Aim for the maximum number of high-quality sub-queries without introducing redundancy.
Each sub-query should be clear, self-contained, and semantically coherent.

Output only the decomposed sub-queries, one per line. Do not include any other text,
explanations, or headers in your output. Each line should contain only one sub-query.
```

LLM-MultiRound appends the candidate names and descriptions, then replaces the last three lines by the following routing constraint:

Multi-round model selection

```
You will then be provided with descriptions of the following Large Language Models
(LLMs): {model_list}.

{model_descriptions}

For each sub-query, select the single LLM that is most likely to generate the
highest-quality response, regardless of cost or efficiency.
Focus entirely on maximizing effectiveness and providing the most accurate and relevant
output. Base your decision strictly on the descriptions of the models.

Output only the decomposed sub-queries and the full name of the selected LLM for each.
Output exactly one sub-query and one LLM per line.

Each line must be formatted as follows:
<sub-query>: <LLM name>

Use a colon ':' as the separator. Do not include any other text, explanations, or
headers in your output.
```

Each selected model is queried with:

Multi-round specialist

```
You are a helpful assistant. You are participating in a multi-agent reasoning process,
where a base model delegates sub-questions to specialized models like you.

Your task is to do your absolute best to either:
Answer the question directly, if possible, and provide a brief explanation;
Offer helpful and relevant context, background knowledge, or insights related to the
question, even if you cannot fully answer it.

If you are completely unable to answer the question or provide any relevant or helpful
information, you must:
Clearly state that you are unable to assist with this question.
Explicitly instruct the base model to consult other LLMs for further assistance.
Keep your response clear, concise, and informative (preferably under 512 tokens).
Stay strictly on-topic.
Do not include irrelevant or generic content.

Here is the sub-question for you to assist with: {sub_query}
```

For non-multiple-choice tasks, the aggregator receives:

Multi-round aggregation

```
You are given a question along with auxiliary information, which consists of several
sub-questions derived from the original question and their respective answers.
Use this information to answer the original question if relevant, but make your own
reasoning step by step before arriving at the final answer.

Important: Your final answer MUST be clearly marked and enclosed within <answer> and
</answer> tags at the end of your response. No other part of the output should be inside
these tags.

Auxiliary Information: {sub_query_and_response_pairs}
Question: {query}
Let's think step by step.
```

For multiple-choice tasks, it instead receives:

Multi-round multiple-choice aggregation

```
You are given a multiple-choice question and supporting sub-answers. Use the information
only if helpful.

Question: {original_query}
Supporting information: {sub_query_and_response_pairs}

Rules:
Select exactly one option: A, B, C, D, or E.
Output only the letter in <answer> tags.
No explanation.

Format:
<answer>A</answer>
```

Router-R1. Router-R1 appends descriptions of the available models to the following template. Its agent can request a specialist model within `<search>` tags; the returned text is inserted as `<information>` in the next round.

Router-R1

```
Answer the given question.
Every time you receive new information, you must first conduct reasoning inside <think>
... </think>.
After reasoning, if you find you lack some knowledge, you can call a specialized LLM
by writing a query inside <search> LLM-Name:Your-Query </search>.

STRICT FORMAT RULES for <search>:
Use an exact model name from: {candidate_model_names}.
Replace Your-Query with a concrete question.
Never copy model descriptions into <search>.
Never output the placeholder format literally.
Before each LLM call, reason about why external information is needed and which model
is appropriate.
The returned response appears between <information> and </information>.
Different models may be called multiple times.

{model_descriptions}

If no further external knowledge is needed, output the final answer inside <answer> ...
</answer>.
Do not output empty answer tags.

Question: {question}
```

CausalLM. The CausalLM router is trained to complete the selected model name after this prefix:

CausalLM

```
You are an intelligent router that selects the best Large Language Model (LLM) for a
given query.

Available LLMs: {model_1}, {model_2}, ...
Based on the query content, complexity, and requirements, predict which LLM would
provide the best response.

Query: {query}
Best LLM:
```

OpenClaw deployment router. When the deployed server uses prompt-based routing, it selects one model with:

OpenClaw deployment router

```
You are an intelligent LLM router. Choose the most suitable model for the user's query.

Available models: {model_descriptions}

Rules:
1. Simple greetings/daily chat -> cheaper models (8b, 9b size)
2. Q&A/knowledge retrieval -> chatqa models
3. Instruction-following/structured output -> Mistral models
4. Code generation/technical questions -> Llama or larger models
5. Complex reasoning/deep analysis -> 70b or larger models

IMPORTANT: Only return the model name, nothing else!

Model names: {model_list}
User query: {query}
```

For image and video inputs, OpenClaw first uses the following preprocessing prompts; the returned text is included in the router query.

OpenClaw image and video preprocessing

Image input:
```
Describe this image concisely in 2-3 sentences.
```
Video input:
```
Describe what you see in these video frames.
```

H.3 Multi-Agent Topology Templates

The multi-agent topologies in §G route every LLM-call node independently from the prompt assigned to that node. Repeated nodes use the same template and differ only in their indexed fields. The final node is shared by all five topologies and is given after the topology-specific templates.

Star. The central planner decomposes the problem for `{n_actors}` actors, then consolidates their reports.

Star planner

```
You are the central planner of a team of {n_actors} agents working on the problem below.
Decompose the problem into exactly {n_actors} subtasks, one per agent, so that their
combined results solve the problem. Output EXACTLY {n_actors} lines, one subtask per
line, no numbering or extra words.

Problem: {user_query}
```

Star actor

```
You are agent {i+1} in a team. Solve the following subtask assigned by your planner.
Be concise and factual.

Original problem: {user_query}
Your subtask: {subtask}
```

Star planner consolidation

```
You are the central planner. Your agents reported the results below. Consolidate them
into a single coherent analysis of the problem.

Problem: {user_query}
Agent reports: Subtask: {subtask_1} Result: {result_1}
Subtask: {subtask_2} Result: {result_2}
...
```

Tree. The root planner assigns two complementary sub-problems to team leads. Each lead refines its branch into one task for a worker, after which the root planner consolidates the two reports.

Tree root planner

```
You are the top-level planner. Split the problem below into exactly 2 complementary
sub-problems for your two subordinate team leads. Output EXACTLY 2 lines, one
sub-problem per line, no numbering.

Problem: {user_query}
```

Tree sub-planner

```
You are team lead {b+1}. Refine the sub-problem below into one concrete task for your
worker agent. Output ONE line only.

Original problem: {user_query}
Your sub-problem: {branch_task}
```

Tree worker

```
You are a worker agent. Solve the task below concisely and factually.

Original problem: {user_query}
Your task: {task}
```

Tree root consolidation

```
You are the top-level planner. Consolidate your two teams' reports into a single
coherent analysis of the problem.

Problem: {user_query}
Team reports: Sub-problem: {branch_task_1} Task: {task_1} Result: {result_1}
Sub-problem: {branch_task_2} Task: {task_2} Result: {result_2}
```

Graph. Each of `{n_actors}` agents first answers independently. In each subsequent communication round, an agent receives the other agents' current answers and revises its own answer.

Graph actor: initial round

```
You are agent {i+1} of {n_actors} independent agents. Solve the problem below on your
own. Show brief reasoning, then your answer.

Problem: {user_query}
```

Graph actor: revision round

```
You are agent {i+1} of {n_actors}. You communicated with all other agents; their current
answers are below. Reconsider and give your revised reasoning and answer.

Problem: {user_query}
Your previous answer: {answers[i]}
Agent {j+1}'s current answer: {answers[j]}
...
```

Chain. The first agent answers independently. Each later agent verifies and improves the immediately preceding decision before passing its result onward.

Chain: first agent

```
You are agent 1 in a chain of {n_agents} agents. Solve the problem below. Show brief
reasoning, then your answer. Your output will be passed to the next agent.

Problem: {user_query}
```

Chain: later agent

```
You are agent {i+1} in a chain of {n_agents} agents. The previous agent passed you
their decision below. Verify it, fix any mistakes, and pass on your improved reasoning
and answer.

Problem: {user_query}
Previous agent's decision: {decision}
```

Plan-Exec-Sum. The GraphPlanner-style topology first decomposes the query into `{width}` atomic sub-queries. Each executor receives its sub-query without an additional instruction, and the summarizer combines the results.

Plan-Exec-Sum planner

```
You are a query decomposition assistant. Your task is to decompose the user's query
into exactly {width} atomic and independent sub-queries.

Keep each sub-query self-contained and non-overlapping.
Prefer factual, directly answerable units.
Output EXACTLY {width} lines, one sub-query per line, no numbering or extra words.

User query: {user_query}
```

Plan-Exec-Sum executor

```
{sub_query}
```

Plan-Exec-Sum summarizer

```
You are a professional summarizer. Summarize the following content into a concise,
coherent paragraph without bullet points. Make it fluent and logically connected.

Content: Sub-query: {sub_query_1} Answer: {answer_1}
Sub-query: {sub_query_2} Answer: {answer_2}
...
Summary:
```

Shared final node. Every topology uses the same final node. Its system instruction is restored from the original task's `[System Instruction]` block, so the required answer format remains task-specific. When there is team context, the final node receives the following user prompt:

Shared final node: user prompt with team context

```
{user_query}
Below is supporting analysis from your team. Use it if helpful, but follow the answer
format required above.
[Team Analysis] {context}
```

Without team context, the final node receives `{user_query}` alone.

I THE COMFYUI VISUAL INTERFACE

As shown in Figure 12, `RouterFactory` exposes its full routing pipeline as a graph on the ComfyUI canvas, where every node is a library component and every edge is an artifact that flows between components. Two input nodes supply the source benchmarks and the candidate pool $\mathcal{M}$, a data-engine node consumes them and emits the query–model matrix as a single edge; and each router node consumes that matrix and emits its evaluation. The matrix records the per-query performance and cost of every candidate, the supervision that every router in §2.1 is trained on. The graph traces the evaluation protocol of §2.2 from left to right, and the router nodes appear in menu groups named after the three router families, so the taxonomy of §2.1 is visible in the node menu. The canvas therefore renders the architecture of Figure 3 as a diagram a user reads and edits, and replacing a router replaces one node while the data-engine and evaluation nodes stay in place, so the modularity of the library becomes visible on the canvas.

The data nodes realize the three stages of the data engine. The Select Datasets and Select LLMs nodes declare the query source and the candidate pool, and the Generate Data node runs query curation, response collection, and scoring in one call and writes the query–model matrix to a self-contained data directory. The router nodes cover every built-in router and are grouped in the menu by router family, and each router node reads its defaults from the same YAML configuration that the command line uses and renders every hyperparameter as a typed widget whose value, range, and options come from that file, so a canvas node and its scripted counterpart run one configuration. On execution a router node writes a runtime configuration, dispatches training and evaluation through the shared router registry, and returns a summary of the query count, the success count, the average performance, and the routing distribution over candidates. The Generate Data node hashes the selected datasets,

candidate pool, and sample size into a metadata record and reuses an existing data directory when the record and its files are unchanged, which skips the costly response-collection stage on repeated runs.

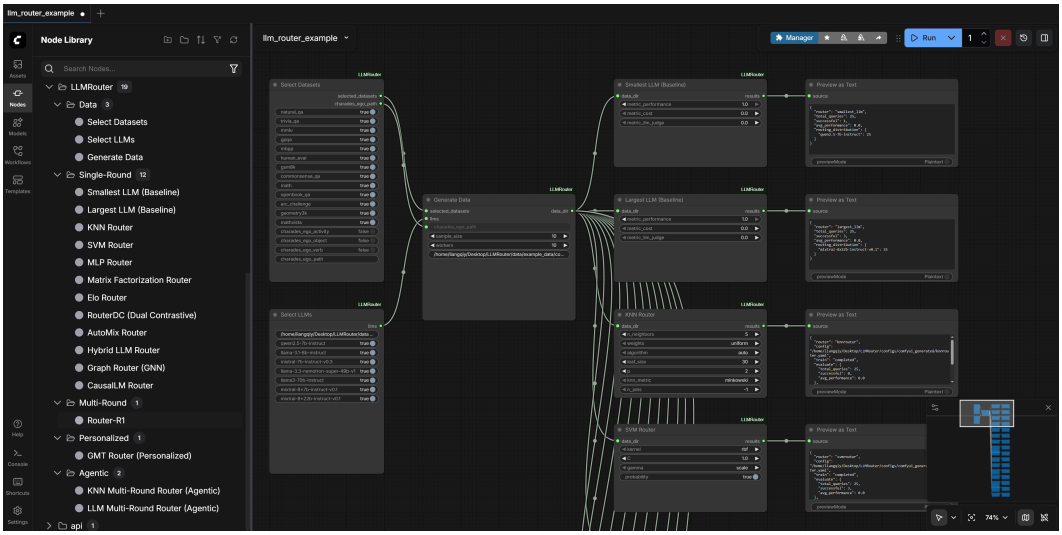


Figure 12: **The ComfyUI interface of `RouterFactory`.** The source benchmarks and the candidate pool enter on the left; the data-engine node produces the query–model matrix, and each router node consumes the matrix and reports its evaluation, so the graph traces the routing pipeline from data construction to evaluation. Each router node exposes the hyperparameters of its library configuration as typed widgets.