TTS-ROUTE: QUERY-ADAPTIVE TEST-TIME SCALING GRAPHS ON THE FLY

Anonymous authors
Paper under double-blind review

ABSTRACT

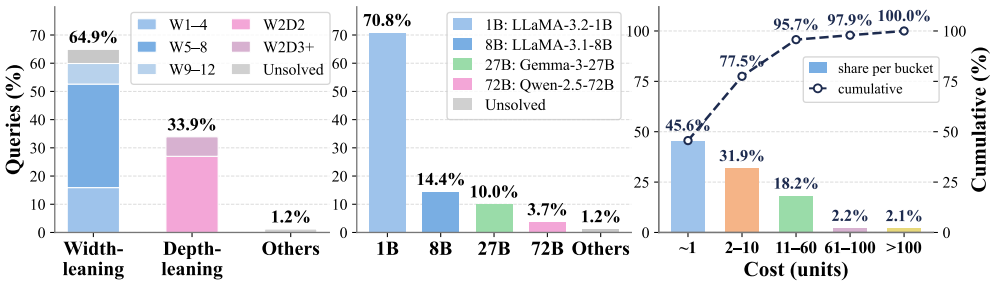
Test-Time Scaling (TTS) improves large language models (LLMs) by allocating additional computation at inference time. However, existing work typically optimizes task-level compute allocation, such as using a single TTS graph across a task, overlooking substantial variation in per-query compute requirements. We therefore study a novel problem of *query-adaptive TTS routing*: given a weak-to-strong LLM pool and a target average compute budget, the goal is to adaptively select the model, scaling strategy, and realized TTS graph for each query. This is challenging because non-Euclidean TTS graphs are difficult to predict directly, informative configuration signals remain unclear, and global compute must be allocated adaptively across queries. To address these challenges, our pilot experiments reveal three insights: (1) TTS graphs should adapt to runtime response quality; (2) per-action success probabilities provide a stable learning target; and (3) low-cost probe responses and verifier feedback improve routing reliability. Based on these insights, we propose TTS-ROUTE, combining a budget-aware TTS-Router for per-action success estimation and Lagrangian action selection with a training-free TTS-Planner that constructs TTS graphs on the fly using verifier feedback. Experiments on three datasets show that TTS-ROUTE achieves the strongest performance–cost trade-off among strong baselines and generalizes to monetary budgets and out-of-distribution queries. Our code is available at link.

1 INTRODUCTION

Test-Time Scaling (TTS) improves the performance of large language models (LLMs) by allocating additional computational resources, such as FLOPs and monetary cost, during inference (Brown et al., 2024; Snell et al., 2025). Existing solution-level TTS methods mainly follow two paradigms: (i) *parallel scaling* (Cobbe et al., 2021; Wang et al., 2023; 2025b), which samples multiple candidate solutions in parallel (Fig. 7 (a) in the Appendix); and (ii) *sequential scaling* (Madaan et al., 2023; Chen et al., 2024b; Gou et al., 2024), which iteratively improves existing solutions through feedback and refinement (Fig. 7 (b)). Recent work further generalizes these paradigms into *graph-structured TTS* (*TTS Graph*) (Wang et al., 2025a), where each model invocation forms a node that may condition on one or multiple preceding solutions (Fig. 7 (c)). During execution, a verifier, such as a reward model, scores candidate solutions and selects the final answer. Wider graphs typically emphasize parallel exploration, whereas deeper graphs enable more extensive sequential refinement.

However, existing methods typically determine TTS configurations at the *task level* (Zheng et al., 2026; Wu et al., 2025; Du et al., 2025), applying a fixed model and scaling strategy uniformly across queries. This *one-size-fits-all* design overlooks substantial variation in per-query inference requirements. To quantify this variation, we perform oracle TTS-Graph search on BEST-Route-Mix training queries (Ding et al., 2025). For each query, we identify the lowest-FLOPs configuration among those that produce a correct response (detailed in Appendix A). Fig. 1(a)–(c) summarizes these cheapest-correct preferences in terms of TTS graph topology, model selection, and compute usage. The results reveal substantial query-level heterogeneity. 64.9% of queries favor width-leaning scaling, while 33.9% favor depth-leaning scaling (Fig. 1(a)); 70.8% are cheapest to solve with the 1B model, while 10.0% favor the 27B model (Fig. 1(b)). Compute requirements also vary sharply: 45.7% of queries require only one normalized cost unit, whereas 22.5% require more than 10 units (Fig. 1(c)). These observations expose a fundamental *per-query compute–necessity mismatch*: effective budget utilization requires adapting both model scale and scaling strategy to each query.

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.



(a) TTS Graph Preference **(b) Model Preference** **(c) Budget Usage**

Figure 1: Per-query preferences in cheapest-correct oracle configurations: (a) TTS graph distribution, (b) model selection distribution, and (c) minimum compute cost required for a correct response.

Motivated by these observations, we study *query-adaptive TTS routing*. Given a set of queries, an LLM pool spanning different model scales, and an average test-time compute budget, the goal is to select an appropriate model–scaling configuration for each query to maximize overall performance under the budget. Ideally, easier queries should use lower-cost configurations, while harder queries receive additional test-time compute. A complete TTS configuration comprises the model, scaling strategy, and realized TTS graph. While graph-structured TTS offers a richer configuration space, it introduces three key challenges: (i) TTS graphs are non-Euclidean discrete structures with a large combinatorial space, making direct graph optimization or prediction difficult; (ii) the signals informative for configuration selection remain unclear; and (iii) a global compute budget must be allocated adaptively across queries. To address them, we conduct pilot experiments that yield three empirical insights. **Insight 1** shows that the realized graph should be constructed on the fly using runtime verifier feedback, motivating a decomposition into *routing*, which selects a model–scaling pair (an *action*), and *planning*, which dynamically instantiates the corresponding graph. For routing, existing query-adaptive methods (Ong et al., 2025; Pan et al., 2026; Zhang et al., 2026) typically learn direct query-to-configuration mappings, implicitly assuming that a stable near-optimal configuration can be inferred from the query alone. **Insight 2** shows that this assumption is unreliable for TTS: repeated oracle searches can yield different model and scaling choices, whereas per-action success probabilities provide a more stable prediction target; we further show theoretically that direct action prediction is bounded by oracle stability. Finally, **Insight 3** shows that incorporating a low-cost probe response and verifier feedback substantially improves routing reliability.

Building on these insights, we propose TTS-ROUTE, a budget-aware framework for query-adaptive TTS with two components: *TTS-Router* and *TTS-Planner*. Motivated by Insight 1, which shows that the realized TTS graph should adapt to runtime response quality, we address Challenge (i) by decoupling TTS configuration into two levels: TTS-Router selects a coarse-grained *action*, i.e., a model–scaling pair $a = (M, S)$ with $S \in \{\text{single}, \text{width-lean}, \text{depth-lean}\}$, while TTS-Planner instantiates the concrete graph online using verifier feedback. Motivated by Insights 2–3, we address Challenge (ii) by estimating the success probability $p_q(a)$ of each action from the query q, a low-cost probe response r_q^p, and its verifier score s_q^p, rather than directly predicting an unstable oracle action. To address Challenge (iii) of adaptively allocating the global budget, given an average compute budget b, the router selects the action maximizing the Lagrangian utility $p_q(a) - \lambda(b)\bar{C}_a$, where $\bar{C}_a$ is the mean profiling cost and $\lambda(b)$ is calibrated on a held-out set to satisfy the target average budget. TTS-Planner then executes the selected action: it performs a single call for $S = \text{single}$, or invokes TTS-PLANNER-W or TTS-PLANNER-D to construct a width-lean or depth-lean graph, respectively, with verifier feedback guiding expansion, stopping, and final readout. Offline, we train a lightweight task-specific proxy verifier, construct an action-profiling library containing $Y_{q,a}$ and $C_{q,a}$, train TTS-Router, and calibrate $\lambda(b)$; at test time, all learned parameters remain fixed while the router selects an action and the planner adaptively realizes the corresponding TTS graph.

Our main **contributions** are threefold: (i) we study *query-adaptive TTS routing*, extending task-level TTS allocation and query-adaptive model routing to jointly consider model selection, scaling strategy, and TTS graph allocation; (ii) inspired by three empirical insights, we propose TTS-ROUTE, which consists of a budget-aware *TTS-Router* for action selection and a *TTS-Planner* that constructs TTS graphs on the fly guided by verifier feedback; and (iii) extensive experiments across multiple datasets and compute budgets show that TTS-ROUTE improves the performance–cost frontier over strong baselines and exhibits strong generalization to OOD tasks and alternative budget metrics.

2 Related Work

Test-time Scaling. Allocating additional inference compute, known as *test-time scaling (TTS)*, can improve LLM performance (Wei et al., 2022; Madaan et al., 2023; Wang et al., 2023; Brown et al., 2024). Solution-level TTS primarily follows two paradigms: *sequential scaling* iteratively refines a response but may propagate early errors, whereas *parallel scaling* aggregates independently generated candidates but lacks depth-wise refinement. Recent work combines both paradigms into *TTS graphs*, representing the scaling process as a tree (Inoue et al., 2025; Miyamoto et al., 2026) or a generalized graph (Wang et al., 2025a). *Compute-optimal TTS* allocates inference compute under a fixed budget and shows that smaller models with reasonable scaling strategies can match or outperform larger models (Wu et al., 2025; Liu et al., 2025; Yue et al., 2025; Snell et al., 2025; Wang et al., 2025b). However, existing methods typically operate on a single model within a restricted TTS search space and apply a uniform task-level strategy across queries. In contrast, we study *query-adaptive TTS graph routing*, which jointly selects the LLM and TTS graph for each query.

Query-adaptive Routing dynamically selects a LLM/config for each query to maximize performance or balance response quality against inference cost (Shnitzer et al., 2024; Hu et al., 2024). Existing methods follow two paradigms. *(i) Best-config routing* trains a query-only router from offline profiling and training to select one configuration (e.g., LLM) from a pool online (Lu et al., 2024; Stripelis et al., 2024; Pan et al., 2026; Zhang et al., 2026), although learned LLM routers often collapse to the globally strongest model (Srivatsa et al., 2024). *(ii) Weak-to-strong routing* balances cost and quality through either cascades that accept or escalate responses (Chen et al., 2024a; Aggarwal et al., 2024; Chang et al., 2026) or learned comparative signals for cost-aware model selection (Ding et al., 2024; 2025; Ong et al., 2025). However, these methods primarily allocate compute across model tiers without considering whether to assign additional inference compute to small models. BEST-Route is the only one, augmenting model routing with a best-of-n strategy; nevertheless, its TTS space is restricted to parallel sampling. In contrast, our TTS graph-augmented routing jointly selects an LLM and a TTS graph, enabling a more general form of query-adaptive test-time scaling.

3 Preliminaries and Problem Formulation

Test-time scaling graphs. Given a query q and an LLM M, TTS generates candidate responses through parallel and sequential operations. As shown in Fig. 7 in Appendix, *Parallel sampling* independently samples multiple candidates, $r_i \sim M(\cdot \mid q)$; *Sequential refinement* instead iteratively refines the preceding response, $r_i \sim M(\cdot \mid q, r_{i-1})$. Combining them, a candidate may integrate and revise a selected set of predecessors $\mathcal{R}^S$, such that $r_i \sim M(\cdot \mid q, \mathcal{R}^S)$. Following Wang et al. (2025a), we represent this process as a directed graph $G = (\mathcal{N}, \mathcal{E})$, where each node $n \in \mathcal{N}$ corresponds to an LLM invocation and each edge $e \in \mathcal{E}$ denotes a dependency between candidate responses. Starting from the query root, the graph expands through parallel sampling, refinement, and their composition; a verifier-based readout ρ returns the best candidate (Algo. 1). We characterize a TTS graph by its *width* W, the maximum nodes per layer, and *depth* D, the longest dependency path, capturing parallel exploration and sequential refinement, respectively.

Verifier. The readout relies on a verifier to select the best candidate response. The verifier may take different forms depending on the task. For example, Brown et al. (2024) uses deterministic unit tests for coding and a reward model for mathematical reasoning, while Snell et al. (2025) adopts learnable process and outcome reward models. We therefore define a general scoring verifier $\mathcal{V}$, trained offline to assign a quality score to each candidate. The readout selects $\rho(q, \mathcal{R}) = \arg\max_{r \in \mathcal{R}} \mathcal{V}(q, r)$.

Model pool and routing space. Let $\mathcal{M} = M_1, \ldots, M_K$ denote a pool of candidate LLMs ordered by model scale. For each query, the system jointly selects a model and allocates test-time compute through a TTS graph. We represent a TTS configuration as $c = (M, S, \mathcal{G})$, where $M \in \mathcal{M}$ is the selected model, $S \in \mathcal{S}$ optionally specifies the scaling strategy, and $\mathcal{G}$ is the realized TTS graph. Together, the model pool and the space of constructible TTS graphs define the routing space.

Cost and budget definition. To fairly compare costs of configurations across model scales and TTS graph topologies, we define a cost function $f_{\text{cost}}(q, M, \mathcal{G})$ that measures the inference cost of executing model M with graph $\mathcal{G}$ on query q (Eq. 15 in the Appendix). The cost can be instantiated using FLOPs, monetary cost, or other compute metrics. In this work, we primarily use a standard prefill+decode FLOPs metric, detailed in Appendix B. To facilitate interpretation, we normalize the

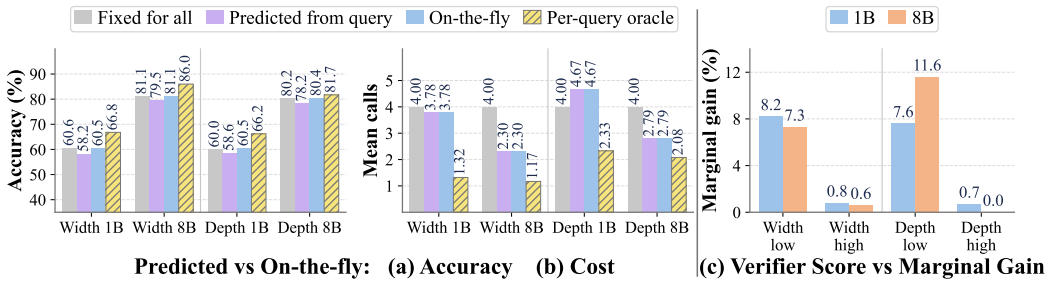


Figure 2: **Adaptive TTS graph construction with verifier guidance.** (a) Accuracy and (b) mean compute cost of fixed, query-predicted, on-the-fly, and per-query oracle graphs; (c) marginal gains from further expansion under low vs. high verifier scores.

cost by a lowest-cost reference configuration:

$$C(q, M, \mathcal{G}) = \frac{f_{\text{cost}}(q, M, \mathcal{G})}{f_{\text{cost}}(q, M_1, \mathcal{G}_{\text{single}})}, \tag{1}$$

where M_1 is the lowest-cost model in the pool and $\mathcal{G}_{\text{single}}$ denotes a single model invocation. Thus, $C(q, M, \mathcal{G}) = 10$ means that the configuration consumes approximately the same compute as 10 single-call generations from M_1 on the same query. Given a user-specified budget b, we define it as an *average per-query compute budget* over the query distribution rather than a strict per-query cap. This enables adaptive allocation of test-time compute, spending less on easier queries while allowing more computation for queries that benefit from additional test-time compute.

Problem definition. Based on the above setting, we define the *query-adaptive TTS routing* problem.

Definition 1. *Given a model pool $\mathcal{M}$, a training set with reference answers $\mathcal{D}_{\text{train}}$, a verifier $\mathcal{V}$, and an average compute budget b, the goal is to learn a policy π that adaptively selects a TTS configuration for each query. Specifically, for a query q, the policy outputs a configuration $c_q = (M_q, S_q, \mathcal{G}_q) = \pi(q)$, whose execution yields a final response r_q. Let r_q^{GT} denote the reference answer and* Eval *the task-specific evaluation function. We define $s_q^\pi = \text{Eval}(r_q, r_q^{\text{GT}})$ as the evaluation score and $C_q^\pi = C(q, M_q, \mathcal{G}_q)$ as the compute cost. The objective is*

$$\pi^\star = \arg\max_\pi \ \mathbb{E}_q\big[s_q^\pi\big] \quad \textit{s.t.} \quad \mathbb{E}_q\big[C_q^\pi\big] \le b. \tag{2}$$

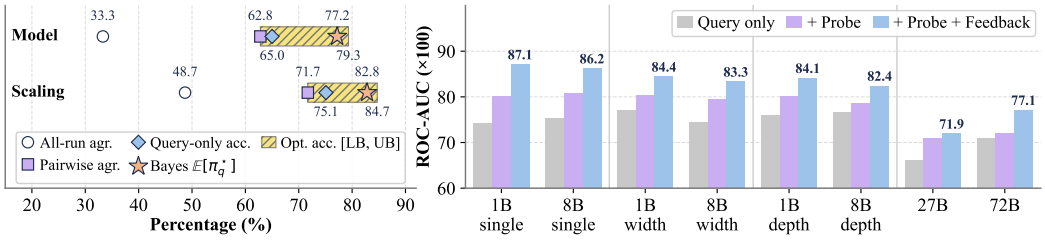
The policy is trained on $\mathcal{D}_{\text{train}}$ and evaluated on $\mathcal{D}_{\text{test}}$. At test time, it uses only information available during inference, such as the query, generated responses, and verifier feedback.

4 Empirical Insights into TTS Routing

Routing to effective TTS configurations poses three challenges: (1) TTS graphs are non-Euclidean discrete structures with a large combinatorial space; (2) the signals most informative for configuration selection remain unclear; and (3) a global compute budget must be allocated adaptively across queries. We therefore conduct pilot experiments to uncover TTS routing patterns and guide the design of a budget-aware, query-adaptive framework for compute-optimal test-time scaling.

Experimental setting. We conduct preliminary experiments on BEST-Route-Mix (Ding et al., 2025) (Appendix C.1) with Llama-3-1B/8B (Grattafiori et al., 2024), Gemma-3-27B (Team, 2025), and Qwen-2.5-72B (Yang et al., 2024). We use FLOPs as the cost metric. Compute is normalized to one 1B invocation, with $b_{\max}$ set to one 72B invocation ($\sim$ 100 units). Final responses are judged by DeepSeek-V4 (Xu et al., 2026) and the proxy verifier is trained offline (Sec. 5.2).

Insight 1: TTS graphs should adapt to runtime response quality, with verifier scores guiding expansion and stopping. We examine whether, for a fixed model and scaling strategy, the TTS graph should be predetermined or adapted online. We compare three strategies: a fixed graph shared across queries, a query-specific graph predicted before execution, and an adaptive graph that expands and stops using verifier scores; the latter two are matched in average compute (details in Appendix C.4). As shown in Fig. 2(a)–(b), adaptive planning achieves accuracy comparable to a large fixed graph with $\sim$ 0.6 fewer calls per query and improves average accuracy by 2% over query-predicted graphs at matched compute, without requiring a parametric graph predictor. Fig. 2(c)



(a) Oracle profiling stability **(b) Per-action success prediction**

Figure 3: (a) Oracle profiling stability over five runs (%); the hatched span bounds optimal prediction accuracy. (b) Per-action success-prediction ROC-AUC ($\times 100$) under different input evidence.

further shows that marginal gains from continued expansion decrease as verifier scores increase, while low-scoring responses benefit more from more computation. These results indicate that TTS graphs should be constructed on the fly, with verifier feedback guiding expansion and stopping.

Insight 2: Per-action success probabilities provide a more stable prediction target. We decompose each TTS configuration into a model–scaling action and an online-instantiated graph, and examine the stability of the former. We define the set of scaling strategies as $\mathcal{S} = \{\text{single}, \text{width-lean}, \text{depth-lean}\}$ and the action space as $\mathcal{A} = \mathcal{M} \times \mathcal{S}$. We conduct five independent oracle searches on 300 queries while keeping all hyperparameters fixed, so that variability arises solely from stochastic LLM generation. Pairwise agreement is only 62.8% for model choice and 71.7% for scaling strategy, dropping to 33.3% and 48.7% across all five runs. This suggests that the cheapest-correct action is not a stable property of the query. To characterize this limitation, let $L(q)$ denote the stochastic oracle action and $\pi_q(a) = \Pr(L(q) = a \mid q)$. For a direct predictor $f : q \mapsto a$, $\mathrm{Acc}(f) = \mathbb{E}_q[\pi_q(f(q))]$, and pairwise oracle stability is $A_{\text{pair}} = \mathbb{E}_q[\sum_{a\in\mathcal{A}} \pi_q(a)^2]$. We theoretically show that the accuracy of direct action prediction is bounded by oracle stability.

Proposition 1 (Stability bounds for direct action prediction)**.** *For any deterministic predictor f, we have* $\mathrm{Acc}(f) \le \mathrm{Acc}^\star := \mathbb{E}_q[\max_{a\in\mathcal{A}} \pi_q(a)]$*, where* $\mathrm{Acc}^\star$ *denotes the Bayes-optimal prediction accuracy. Furthermore,*

$$\underbrace{A_{\text{pair}}}_{\mathrm{Acc}_{\mathrm{LB}}} \le \mathrm{Acc}^\star \le \mathbb{E}_q\left[\sqrt{\textstyle\sum_{a\in\mathcal{A}} \pi_q(a)^2}\right] \le \underbrace{\sqrt{A_{\text{pair}}}}_{\mathrm{Acc}_{\mathrm{UB}}}. \tag{3}$$

The lower bound is attained by a randomized predictor that independently samples an oracle label and uses it as its prediction. Moreover, $\mathrm{Acc}^\star = \mathrm{Acc}_{\mathrm{LB}}$ *if and only if* π_q *is uniform over its nonzero support for almost every query under the query distribution.*

The proof is provided in Appendix D. The measured stability implies upper bounds of 79.3% and 84.7% for direct model and scaling-strategy prediction, respectively, while a query-only MiniLM–MLP achieves only 65.0% and 75.1% (Fig. 3(a)). These results highlight the difficulty of predicting oracle actions from query text alone, as the cheapest-correct label varies with stochastic generation. We therefore estimate, for each $a \in \mathcal{A}$, its success probability $p_q(a) = \Pr(\mathrm{Eval}(r_{q,a}, r_q^{\mathrm{GT}}) = 1)$, which is a well-defined query–action property estimable from repeated observations and not subject to oracle-label instability. Overall, per-action success probabilities provide a more stable and reliable characterization of query–action suitability than the stochastic cheapest-correct action.

Insight 3: Low-cost probe outputs and verifier feedback improve routing reliability. Building on Insight 2, we formulate routing as per-action success prediction, where $Y_{q,a} = \mathrm{Eval}(r_{q,a}, r_q^{\mathrm{GT}}) \in \{0, 1\}$ and a frozen MiniLM (Wang et al., 2020) encoder with action-specific MLP heads estimates $p_q(a)$. Using the same training and test splits, we compare three input variants under an identical backbone: query only, query plus a low-cost 1B probe, and query plus both the probe and its verifier score as feedback. We evaluate prediction quality using ROC-AUC, which measures how well the predictor distinguishes successful from unsuccessful action executions. As shown in Fig. 3(b), probe outputs consistently improve AUC across actions, while adding verifier feedback yields further gains. For example, AUC ($\times 100$) increases from 74.3 to 80.2 to 87.1 for 1B-single and from 75.3 to 80.7 to 86.2 for 8B-single. These results show that probe responses reveal runtime model behavior, while verifier scores provide complementary quality signals for more reliable routing.

5 THE PROPOSED FRAMEWORK–TTS-ROUTE

Guided by the insights in Sec. 4, we propose TTS-ROUTE for budget-aware, query-adaptive TTS in Fig. 4, which consists of *TTS-Router*, which estimates per-action success probabilities and performs budget-aware Lagrangian action selection, and *TTS-Planner*, which executes the selected action by constructing the TTS graph on the fly using verifier feedback and selecting the final response via an offline-trained proxy verifier. Offline, we profile candidate actions, train the proxy verifier and router, and calibrate the Lagrangian coefficient across target budgets. At online stage, all learned parameters remain fixed: TTS-Router selects an action under the target budget, and TTS-Planner adaptively realizes the corresponding TTS graph.

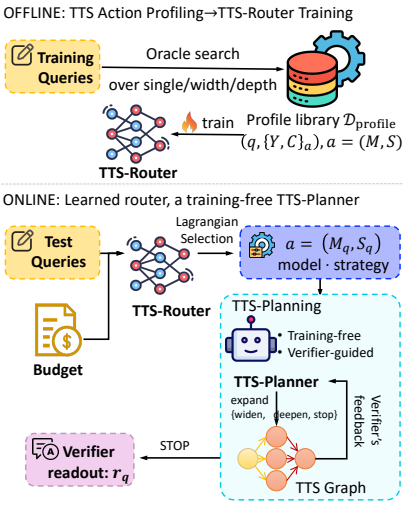


Figure 4: Overview of TTS-ROUTE

5.1 ONLINE STAGE: ACTION ROUTING AND DYNAMIC TTS EXPANSION

Given a query q and a target average budget b, the offline-trained TTS-Router estimates per-action success probabilities and selects an action using the calibrated Lagrangian objective (Sec. 5.2). The corresponding training-free TTS-Planner then constructs the TTS graph on the fly according to the model and scaling strategy specified by the selected action. Fig. 4 (bottom) and Algo. 2 in Appendix show the online procedure.

TTS-Router constructs an input vector x_q from the query, a low-cost probe response r_q^p, and its verifier score s_q^p, and estimates $p_q(a) = \Pr(Y_{q,a} = 1 \mid x_q)$ for each $a \in \mathcal{A}$. Given a target budget b, it selects $\hat{a} = \arg\max_{a \in \mathcal{A}}[p_q(a) - \lambda(b)\bar{C}_a]$, where $\bar{C}_a$ is the mean profiling cost and $\lambda(b)$ is calibrated offline to control the accuracy–cost trade-off while satisfying the target budget (Sec. 5.2).

Training-free TTS-Planner. *Motivated by Insight 1, we address the challenge of non-Euclidean graph search space by progressively constructing the TTS graph using runtime verifier feedback.* Given the selected action $\hat{a} = (M, S)$, TTS-Planner executes the corresponding scaling strategy. If $S = \text{single}$, it returns a single-call response from M; otherwise, it dynamically constructs a layer-wise TTS graph, where each layer contains candidate responses and nodes at layer $\ell \geq 2$ fuse and refine selected responses from the preceding layer. Specifically, TTS-PLANNER-W is used for $S = \text{width-lean}$ and TTS-PLANNER-D for $S = \text{depth-lean}$.

*Width-lean planner (*TTS-PLANNER-W*).* The width-lean planner prioritizes parallel exploration and increases depth only after the current layer converges. Starting from layer $\ell = 1$, it generates candidate responses in batches. Let $\mathcal{R}_{\ell,t}$ denote the t-th batch at layer ℓ, and $\mathcal{R}_{\leq(\ell,t)}$ all responses generated up to that point. We define the best verifier score as $s^*_{\ell,t} = \max_{r \in \mathcal{R}_{\leq(\ell,t)}} \mathcal{V}(q, r)$. If $s^*_{\ell,t} - s^*_{\ell,t-1} > \epsilon$, the planner continues expanding the current layer, where ϵ is a small predefined improvement threshold; otherwise, it advances to layer $\ell + 1$ by fusing and refining selected responses from layer ℓ. Let s^*_ℓ denote the best verifier score after layer ℓ. Expansion stops when improvement is negligible or the next expansion would exceed the query-level budget:

$$s^*_\ell - s^*_{\ell-1} \leq \epsilon \quad \text{or} \quad C_{\ell,t} + \widehat{C}_{\text{next}} > B_q, \tag{4}$$

where $C_{\ell,t}$ is the accumulated cost and $\widehat{C}_{\text{next}}$ the estimated cost of the next expansion. For an action using $M_k, k < K$, we set $B_q = \widehat{C}(q, M_{k+1}, \mathcal{G}_{\text{single}})$, so that scaling M_k remains cheaper than escalating directly to the next larger model M_{k+1}. Finally, the planner returns $\hat{r} = \arg\max_{r \in \mathcal{R}} \mathcal{V}(q, r)$.

*Depth-lean planner (*TTS-PLANNER-D*).*The depth-lean planner prioritizes sequential refinement by iteratively revising the best candidate from each layer; further details are provided in Appendix E.

5.2 OFFLINE STAGE: LEARNING AND BUDGET CALIBRATION

The offline stage prepares three components for online inference: a proxy verifier $\mathcal{V}$, an action-success router $\theta_{\text{router}} : (q, a) \mapsto p_q(a)$, and budget-aware Lagrangian coefficients $\lambda(b)$. We train the verifier, build the action-profiling library, then train the router and calibrate $\lambda(b)$ across budgets.

Proxy Verifier Training. Verifier scoring is central to our routing and planning process. However, existing general-purpose verifiers are often large reward models, such as ArmoRM (Wang et al., 2024), or proprietary LLM judges (Kwok et al., 2026). Although effective, their inference cost can exceed that of models in our routing pool, making them impractical for query-adaptive TTS. We therefore distill a strong teacher verifier into a lightweight task-specific proxy verifier $\mathcal{V}(q, r) \in \mathbb{R}$ that estimates the quality of response r for query q. Specifically, we construct pairwise preference data from responses generated for each training query. For a query q, we sample multiple responses from the same model and use a teacher verifier to rank them. For each pair (r^+, r^-), r^+ denotes the teacher-preferred response. We train the proxy verifier using the Bradley–Terry objective:

$$\mathcal{L}_{\text{ver}} = -\sum\nolimits_{(q,r^+,r^-)} \log \sigma\left(\mathcal{V}(q, r^+) - \mathcal{V}(q, r^-)\right), \tag{5}$$

where $\sigma(\cdot)$ is the sigmoid function. After training, the teacher verifier is no longer required online; the proxy verifier provides quality signals for probe scoring, graph expansion and stopping, and final response readout. In practice, we use ArmoRM (Wang et al., 2024) as the teacher verifier.

Action Profiling. For each training query $q \in \mathcal{D}_{\text{train}}$ and action $a = (M, S) \in \mathcal{A}$, where $M \in \mathcal{M}$ and $S \in \mathcal{S} = \{\text{single}, \text{width-lean}, \text{depth-lean}\}$, we execute the action using the same planning procedure as in online inference. Execution is label-free; only after completion do we evaluate the returned response $\hat{r}_{q,a}$ against the reference r_q^{GT}. We define $Y_{q,a} = \text{Eval}(\hat{r}_{q,a}, r_q^{\text{GT}}) \in \{0, 1\}$ and record its realized normalized compute cost $C_{q,a}$. This yields the action-profiling library

$$\mathcal{D}_{\text{profile}} = \{(q, \{Y_{q,a}, C_{q,a}\}_{a\in\mathcal{A}}) \mid q \in \mathcal{D}_{\text{train}}\}. \tag{6}$$

Reference answers are used only to evaluate actions and never to guide TTS graph expansion.

TTS-Router Training. *Motivated by Insight 2 that the cheapest-correct oracle action is stochastic, we model action success probabilistically rather than directly predicting a single oracle action.* Specifically, $Y_{q,a}$ is a Bernoulli outcome under stochastic generation, and the router θ_{router} estimates $p_q(a) = \Pr(Y_{q,a} = 1 \mid x_q)$ for each action $a \in \mathcal{A}$. *Motivated by Insight 3 that probe outputs and verifier feedback improve routing reliability, we augment each query with a low-cost probe response and its verifier score.* We split the profiling queries into disjoint router-training and calibration sets, $\mathcal{D}_{\text{router}}$ and $\mathcal{D}_{\text{cal}}$. For each query q, the lowest-cost model M_1 generates a probe response r_q^p with score $s_q^p = \mathcal{V}(q, r_q^p)$. Using a frozen MiniLM encoder $e(\cdot)$, we construct $x_q = [e(q); e(r_q^p); \bar{s}_q^p]$, where $\bar{s}_q^p$ is standardized using router-training statistics. The router uses action-specific MLP heads with sigmoid outputs to predict $p_q(a)$ and jointly trains them with binary cross-entropy:

$$\mathcal{L}_{\text{router}} = -\sum\nolimits_{q\in\mathcal{D}_{\text{router}}} \sum\nolimits_{a\in\mathcal{A}} \left[Y_{q,a} \log p_q(a) + (1 - Y_{q,a}) \log(1 - p_q(a))\right]. \tag{7}$$

This multi-label objective allows multiple actions to succeed and supports budget-aware selection.

Budget-aware Lagrangian Action Selection. To overcome the challenge of allocating compute adaptively across queries under a global budget, we employ a Lagrangian rule for action selection. Given the predicted success probability $p_q(a)$ and mean profiling cost $\bar{C}_a$ of each action, we select the action with the highest cost-adjusted utility:

$$\hat{a}(q; \lambda) = \arg\max_{a\in\mathcal{A}} \left[p_q(a) - \lambda \bar{C}_a\right], \tag{8}$$

where $\bar{C}_a$ is the average realized cost of action a over the router-training set. This provides a practical Lagrangian surrogate for the budget-constrained objective in Eq. 2, with $\lambda \geq 0$ controlling the accuracy–cost trade-off. For each target average budget b, we calibrate λ on the held-out set $\mathcal{D}_{\text{cal}}$:

$$\lambda(b) = \arg\max_{\lambda\in\Lambda} \frac{1}{|\mathcal{D}_{\text{cal}}|} \sum_{q\in\mathcal{D}_{\text{cal}}} Y_{q,\hat{a}(q;\lambda)} \quad \text{s.t.} \quad \frac{1}{|\mathcal{D}_{\text{cal}}|} \sum_{q\in\mathcal{D}_{\text{cal}}} C_{q,\hat{a}(q;\lambda)} \leq b, \tag{9}$$

where Λ is a predefined grid of candidate coefficients. Since the profiled outcomes $Y_{q,a}$ and costs $C_{q,a}$ are already available for all actions, calibration requires no additional LLM execution. The resulting $\lambda(b)$ is fixed before test evaluation, with no post-hoc adjustment based on realized costs.

6 Experiments

We compare TTS-Route with diverse baselines under a unified budget protocol and ablate its components, monetary-cost performance, cross-dataset generalization, and budget-allocation behavior.

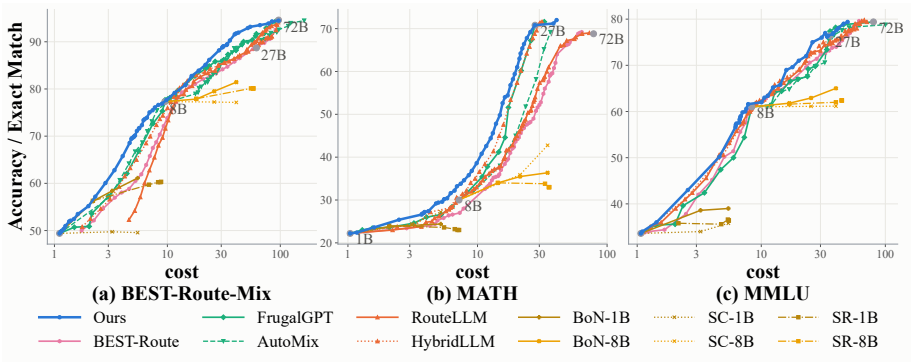


Figure 5: Performance–cost frontiers on (a) BEST-Route-Mix, (b) MATH, and (c) MMLU. Cost is on a logarithmic FLOPs scale, united by a single 1B call; gray markers are single-model baselines.

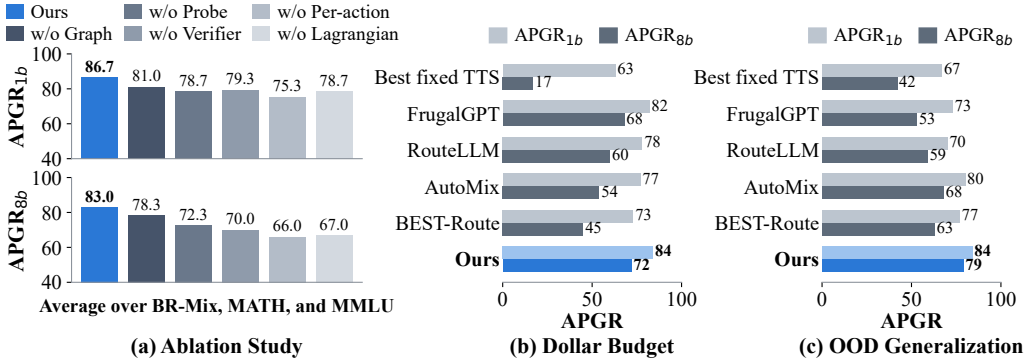


Figure 6: (a) Average ablation across BEST-Route-Mix, MATH, and MMLU (APGR_{1b}/APGR_{8b}). (b) Dollar-budget APGR on BEST-Route-Mix. (c) OOD APGR on MT-Bench with calibration on BEST-Route-Mix and no adaptation. Boldface marks the best results.

Experimental setup. We evaluate on BEST-Route-Mix, MATH (Hendrycks et al., 2021b), and MMLU (Hendrycks et al., 2021a) using the same model pool as above, spanning 1B–72B. BEST-Route-Mix is graded by an LLM judge, while MATH and MMLU use exact match; dataset splits are provided in Appendix C.1. Baselines fall into four categories: (i) *Vanilla*: one call to each model; (ii) weak-to-strong routing/cascades (RouteLLM, Hybrid LLM, AutoMix, and FrugalGPT); (iii) fixed TTS (BoN, Self-Consistency (SC), Self-Refinement (SR), and fixed width×depth graphs); and (iv) adaptive TTS (BEST-Route and BG-MCTS), with details in Appendix C.6. All methods share the same model pool, proxy verifier, and FLOPs accounting. We report performance for each target budget $b \in \{5, 10, \ldots, 100\}$ and APGR, defined as $\text{APGR} = K^{-1} \sum_{k=1}^{K} (\text{acc}(b_k) - A_W)/(A_S - A_W)$ over $K = 100$ budgets (grid search on b_k) between the weak- and strong-model costs, where A_W and A_S are the corresponding single-call accuracies, using 1B/8B as weak anchors and 72B as the strong anchor (Appendix C.7) and we note both as $\text{APGR}_{1b}/\text{APGR}_{8b}$.

Main results. Tables 1, 5 and Fig. 5 summarize results across three datasets. We observe: (1) TTS-ROUTE achieves the highest APGR overall: 86/76 on BEST-Route-Mix, 88/94 on MATH, and 86/79 on MMLU. On BEST-Route-Mix, it leads at four of six budgets and approaches 72B accuracy within 1 point at 87% of its cost; On MATH, it surpasses 72B accuracy (72 vs. 69) at lower cost; on MMLU, it matches 72B at 58% of the cost, showing that our design motivated by Insights 1–3 effectively exploits the richer TTS routing space. (2) TTS-ROUTE consistently outperforms adaptive-TTS, model-routing and cascade baselines, improving APGR_{1b} over BEST-Route by 10, 20, and 4 points across the three datasets. BEST-Route considers only parallel scaling, BG-MCTS uses a fixed LLM, and conventional routers adapt only model selection. In contrast, TTS-ROUTE jointly adapts model choice, scaling strategy, and the realized TTS graph. (3) Fixed-TTS methods are weaker because fixed computation patterns cannot adapt expansion and stopping to query-specific needs. (4) Adaptive TTS yields smaller gains on MMLU, likely because performance relies more on parametric knowledge and model scale, leaving less room for gains from richer TTS routing.

Ablation Studies. We evaluate the contribution of each component by comparing the full version with five variants: *w/o Verifier Guidance* (w/o Insight 1), *w/o Per-action Success Prediction* (w/o

Table 1: BEST-Route-Mix results across target budgets, reported as accuracy@cost (%). Best is bolded; "–" means infeasible points where the minimum required cost exceeds the target budget.

	Method	b=10	b=20	b=30	b=50	b=75	b=100	APGR$_{1b}$	APGR$_{8b}$
Vanilla	1B	49@1	49@1	49@1	49@1	49@1	49@1	–	–
	8B	77@10	77@10	77@10	77@10	77@10	77@10	–	–
	27B	–	–	–	–	89@62	89@62	–	–
	72B	–	–	–	–	–	**95**@97	–	–
Fixed TTS	BoN-1B	62@10	65@20	66@30	66@50	68@75	68@99	36	−62
	BoN-8B	77@10	78@19	80@28	82@47	82@75	83@94	63	21
	SC-1B	49@10	48@20	49@30	49@50	49@75	49@99	1	−159
	SC-8B	77@10	77@19	77@28	77@47	78@75	77@94	57	2
	SR-1B	60@9	60@9	60@9	60@9	60@9	60@9	23	−98
	SR-8B	77@10	78@18	78@18	78@18	80@58	80@58	54	9
	Graph[1,1]	55@8	55@8	58@30	58@30	61@70	61@70	18	−111
	Graph[1,2]	50@3	50@3	56@22	56@22	59@66	59@66	13	−124
	Graph[2,1]	56@2	56@2	61@23	61@23	61@23	62@79	23	−98
Router	RouteLLM	72@10	82@19	85@28	88@49	92@70	94@92	78	59
	HybridLLM	74@9	83@18	85@28	86@35	90@70	94@95	78	55
	AutoMix	77@9	80@20	85@30	89@46	90@71	91@85	79	57
	FrugalGPT	**78**@10	81@20	84@29	91@48	93@61	94@95	81	65
Adaptive TTS	BEST-Route	72@9	81@18	83@27	87@44	89@64	93@91	76	51
	BG-MCTS	57@9	57@9	78@30	79@41	81@71	82@100	56	−2
Ours	TTS-ROUTE	77@9	**85**@20	**88**@29	**92**@43	**94**@66	94@84	**86**	**76**

Insight 2), *w/o Probe Response and Verifier Feedback* (w/o Insight 3), *w/o Graph Actions*, and *w/o Lagrangian Action Selection*. As shown in Fig. 6(a), all components contribute to performance. Averaged across three datasets, removing verifier guidance, per-action success prediction, and probe evidence reduces APGR$_{1b}$/APGR$_{8b}$ by $7.3/13.0$, $11.3/17.0$, and $8.0/10.7$ points, respectively, supporting Insights 1–3 from the pilot experiments. Removing graph actions lowers APGR by $5.7/4.7$ points, confirming the benefit of a richer routing space beyond single-model selection. Finally, replacing Lagrangian selection with threshold-based selection reduces APGR by $8.0/16.0$ points, showing that explicitly balancing predicted success and compute cost enables more effective budget allocation across queries. Further details and analysis are provided in Appendix F.1.

Generalization to Dollar Cost as Budget. Beyond FLOPs, API users often care directly about monetary cost, whose relative scale can differ substantially across models. We therefore use dollar cost as an alternative budget metric, computed from input and output tokens using OpenRouter prices (OpenRouter, 2026). We retain the trained router, replace each action's mean FLOPs cost $\bar{C}_a$ with its mean dollar cost $\bar{C}_a^{\$}$, and recalibrate $\lambda(b)$ and the baseline thresholds on the calibration set. As shown in Fig. 6(b) and Appendix Table 7, TTS-ROUTE achieves the highest accuracy at five of six budgets and the best APGR$_{1b}$/APGR$_{8b}$ ($84/72$ versus $82/68$ for FrugalGPT). It further reaches 89 accuracy at only 37% of the always-72B cost, showing robust generalization across cost metrics.

Generalization to Out-of-distribution Queries. To evaluate generalization to OOD queries, we apply routing methods calibrated on BEST-Route-Mix directly to the 80 turn-one MT-Bench questions without retraining. As shown in Fig. 6(c) and Appendix Table 8, TTS-ROUTE achieves the highest APGR$_{1b}$/APGR$_{8b}$ ($84/79$), outperforming AutoMix ($80/68$) and BEST-Route ($77/63$). Its frozen $\lambda(b)$ also satisfies every target budget, indicating robust transfer of budget calibration, whereas RouteLLM exceeds several target budgets after transfer. Details are provided in Appendix F.2. Appendix F has more experimental analyses including action allocation across target budgets.

7 CONCLUSION

We study a novel problem of budget-aware, query-adaptive TTS graph routing. Pilot experiments yield three insights: (1) TTS graphs should adapt to runtime response quality; (2) per-action success probabilities provide a more stable learning target; and (3) low-cost probes improve routing reliability. Based on these insights, we propose TTS-ROUTE, combining a TTS-Router with Lagrangian action selection for balancing performance and cost and a training-free TTS-Planner that constructs TTS graphs on the fly guided by verifier feedback. Experiments on three datasets show that TTS-ROUTE achieves the best performance–cost trade-off, generalizes to monetary budgets and OOD queries, and approaches the performance of the strongest model at substantially lower cost.

AI USE DISCLOSURE STATEMENT

Large language models (LLMs) were used for language refinement, including rephrasing, grammar correction, and readability improvement, as well as limited assistance with editing and debugging code. LLMs were not used for research ideation, hypothesis formulation, methodological or experimental design, implementation of the proposed methods, data analysis, result interpretation, or the development of theoretical or conceptual contributions. All research ideas, methodological decisions, analyses, interpretations, and conclusions were developed by the authors. All LLM-assisted text and code were carefully reviewed and verified by the authors for accuracy and correctness. The authors take full responsibility for the final content of the manuscript and for its compliance with applicable ethical and academic-integrity standards.

ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics. No human or animal subjects were involved. Datasets (BEST-Route-Mix, MATH, MMLU) were used in compliance with guidelines, with no privacy violations or personally identifiable information. We ensured fairness, avoided bias, and upheld transparency and integrity throughout the research.

REPRODUCIBILITY STATEMENT

We provide an anonymous code package and configuration at link to ensure reproducibility of all experiments; Training/inference details are provided in Section 4, 6, and C; The datasets we used are public datasets, with sources, task setups, and pre-processing steps provided in Appendix C.1.

REFERENCES

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.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Raeyoung Chang, Dongwook Kwon, Jisoo Lee, and Nikhil Verma. Cascadedebate: Multi-agent deliberation for cost-aware llm cascades. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL 2026)*, pp. 1329–1340, 2026.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024a. ISSN 2835-8856. URL https://openreview.net/forum?id=cSimKw5p6R. Featured Certification.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=KuPixIqPiq.

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.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks Lakshmanan, and Ahmed H Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, volume 2024, pp. 41348–41366, 2024.

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks VS Lakshmanan, Qingyun Wu, and Victor Rühle. Best-route: adaptive llm routing with test-time optimal compute. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 13870–13884, 2025.

Weihua Du, Yiming Yang, and Sean Welleck. Optimizing temperature for language models with multi-sample inference. In *International Conference on Machine Learning*, pp. 14648–14668. PMLR, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Sx038qxjek.

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.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTav3: Improving deBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=sE7-XhLxHA.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

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. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021b. URL https://openreview.net/forum?id=7Bywt2mQsCe.

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. In *Agentic Markets Workshop at ICML 2024*, 2024. URL https://openreview.net/forum?id=IVXmV8Uxwh.

Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper? scaling LLM inference-time compute with adaptive branching tree search. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=jAsr5GHt3P.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. Llm-as-a-verifier: A general-purpose verification framework. *arXiv preprint arXiv:2607.05391*, 2026.

Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. Can 1b LLM surpass 405b LLM? rethinking compute-optimal test-time scaling. In *Workshop on Reasoning and Planning for Large Language Models*, 2025. URL https://openreview.net/forum?id=CvjX9Lhpze.

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 Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *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, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.109. URL https://aclanthology.org/2024.naacl-long.109/.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Sora Miyamoto, Daisuke Oba, and Naoaki Okazaki. Aligning tree-search policies with fixed token budgets in test-time scaling of LLMs. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=TgaVuBX8dx.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition. https://openai.com/index/gpt-5-6/, July 2026. Accessed: 2026-09-26.

OpenRouter. Openrouter model pricing. https://openrouter.ai/models, 2026. Accessed: September 22, 2026.

Melissa Z Pan, Negar Arabzadeh, Mathew Jacob, Fiodar Kazhamiaka, Esha Choukse, and Matei Zaharia. Natural language query to configuration for retrieval agents. *arXiv preprint arXiv:2605.27361*, 2026.

Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. Large language model routing with benchmark datasets. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Zb0ajZ7vAt.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Kv Aditya Srivatsa, Kaushal Kumar Maurya, and Ekaterina Kochmar. Harnessing the power of multiple minds: Lessons learned from LLM routing. In Shabnam Tafreshi, Arjun Akula, João Sedoc, Aleksandr Drozd, Anna Rogers, and Anna Rumshisky (eds.), *Proceedings of the Fifth Workshop on Insights from Negative Results in NLP*, pp. 124–134, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.insights-1.15. URL https://aclanthology.org/2024.insights-1.15/.

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 Franck Dernoncourt, Daniel Preoţiuc-Pietro, and Anastasia Shimorina (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 452–462, Miami, Florida, US, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-industry.34. URL https://aclanthology.org/2024.emnlp-industry.34/.

Gemma Team. Gemma 3. 2025. URL https://arxiv.org/abs/2503.19786.

Fali Wang, Jihai Chen, Shuhua Yang, Runxue Bao, Tianxiang Zhao, Zhiwei Zhang, Xianfeng Tang, Hui Liu, Qi He, and Suhang Wang. Generalizing test-time compute-optimal scaling as an optimizable graph. *arXiv preprint arXiv:2511.00086*, 2025a.

Fali Wang, Hui Liu, Zhenwei DAI, Jingying Zeng, Zhiwei Zhang, Zongyu Wu, Chen Luo, Zhen Li, Xianfeng Tang, Qi He, and Suhang Wang. AgentTTS: Large language model agent for test-time compute-optimal scaling strategy in complex tasks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=BuYtcTUMyA.

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

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for llm problem-solving. In *The Thirteenth International Conference on Learning Representations*, 2025.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

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.

Zhenrui Yue, Honglei Zhuang, Aijun Bai, Kai Hui, Rolf Jagerman, Hansi Zeng, Zhen Qin, Dong Wang, Xuanhui Wang, and Michael Bendersky. Inference scaling for long-context retrieval augmented generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=FSjIrOm1vz.

Zhiheng Zhang, Yuanzhe Zhang, Bohan Yu, Daojian Zeng, Kang Liu, and Jun Zhao. Heterodesigner: Automated design of multi-agent systems with heterogeneous llms. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27553–27574, 2026.

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.

Tong Zheng, Haolin Liu, Chengsong Huang, Huiwen Bao, Sheng Zhang, Rui Liu, Runpeng Dai, Ruibo Chen, Chenxi Liu, Tianyi Xiong, et al. Llms improving llms: Agentic discovery for test-time scaling. *arXiv preprint arXiv:2605.08083*, 2026.

Algorithm 1 TTS Graph Execution

Require: Query q, selected model M, TTS graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, proxy verifier $\mathcal{V}$
Ensure: Final response $\hat{r}$

$\mathcal{R} \leftarrow \varnothing$
for each node $n_i \in \mathcal{N}$ in topological order **do**
  $\mathrm{Pred}(n_i) \leftarrow \{n_j : (n_j, n_i) \in \mathcal{E}\}$
  if $\mathrm{Pred}(n_i) = \varnothing$ **then**
    $r_i \sim M(\cdot \mid q)$
  else
    $\mathcal{R}_i^S \leftarrow \{r_j : n_j \in \mathrm{Pred}(n_i)\}$
    $r_i \sim M(\cdot \mid q, \mathcal{R}_i^S)$
  end if
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{r_i\}$
end for
$\hat{r} \leftarrow \rho(q, \mathcal{R}) = \arg\max_{r \in \mathcal{R}} \mathcal{V}(q, r)$
return $\hat{r}$

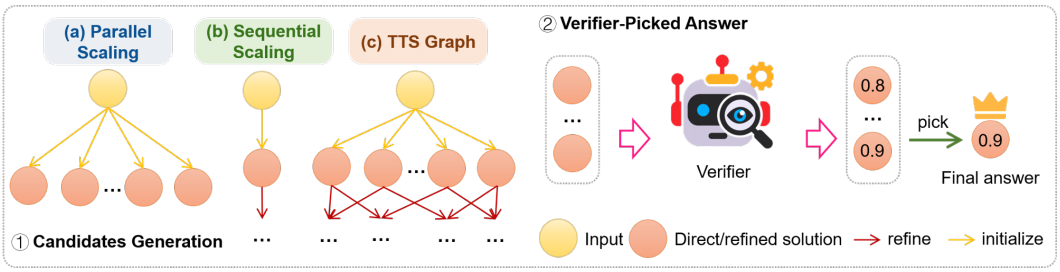


Figure 7: Existing test-time scaling paradigms.

A Motivating Experiments

To characterize query-level variation in model choice, TTS graph topology, and compute requirements, we perform an oracle configuration search on the BEST-Route-Mix (Ding et al., 2025) training set and analyze the resulting most cost-effective configurations. For each model (1B, 8B, 27B, and 72B), we evaluate multiple TTS strategies, including single-call inference, width expansion, and depth expansion, under a per-query budget B_q equal to the FLOPs cost of one 72B invocation. For each TTS strategy given an LLM, we progressively increase test-time compute according to its predefined expansion rule and perform readout and correctness evaluation after every expansion. Width expansion incrementally adds parallel samples, while depth expansion follows a depth-first procedure that progressively fuses and refines candidate responses. Expansion stops once the readout produces a correct response, yielding the cheapest correct configuration for each query, model, and TTS strategy. We then select, across all models and strategies, the globally lowest-cost correct model–TTS topology configuration (M, S, G) for each query. Fig. 1 summarizes the resulting oracle preferences along three dimensions. Fig. 1(a) shows TTS topology preferences, Fig. 1(b) shows model preferences, with W and D denoting width and depth, and Fig. 1(c) shows the compute-cost distribution. We group costs into $[0, 1.5)$, $[1.5, 10)$, $[10, 60)$, $[60, 100)$, and ≥ 100 units, roughly corresponding to the cost of at most one invocation of the 1B, 8B, 27B, and 72B models, and higher-compute regimes, respectively, and overlay the cumulative distribution.

B Cost Model and Budget Units

FLOPs accounting. We use floating-point operations (FLOPs) as the primary measure of test-time compute. For each invocation of model M, let n_p and n_d denote the numbers of prompt (prefill) and completion (decode) tokens, respectively. Let P_M, L_M, and D_M denote the number of non-embedding parameters, transformer layers, and hidden dimension of M. Following standard prefill–decode accounting, we count one multiply–add as two FLOPs and assume causal self-attention with

key–value caching during autoregressive decoding. The graph-level cost $f_{\text{cost}}(q, M, \mathcal{G})$ is obtained by summing the FLOPs of all model invocations in $\mathcal{G}$.

For projection and MLP operations, each non-embedding parameter is applied approximately once per token, yielding $2P_M$ FLOPs per token. Their prefill and decode costs are therefore $2P_M n_p$ and $2P_M n_d$, respectively. For self-attention, causal masking yields a triangular attention pattern during prefill, while each completion token attends to all preceding prompt and completion tokens. The corresponding attention costs are

$$f_{\text{attn}}^{\text{pre}}(M, n_p) = 2L_M D_M n_p(n_p + 1), \tag{10}$$

and

$$f_{\text{attn}}^{\text{dec}}(M, n_p, n_d) = 2L_M D_M n_d(2n_p + n_d + 1). \tag{11}$$

Combining projection/MLP and attention costs, the FLOPs of a single model invocation are

$$f_{\text{pre}}(M, n_p) = 2P_M n_p + 2L_M D_M n_p(n_p + 1), \tag{12}$$

$$f_{\text{dec}}(M, n_p, n_d) = 2P_M n_d + 2L_M D_M n_d(2n_p + n_d + 1), \tag{13}$$

giving

$$f_{\text{inv}}(M, n_p, n_d) = f_{\text{pre}}(M, n_p) + f_{\text{dec}}(M, n_p, n_d). \tag{14}$$

In our model pool, the model specifications are listed in Table 2.

Table 2: Model specifications used in our model pool. P_M, L_M, and D_M denote the non-embedding parameter count, number of transformer layers, and hidden dimension, respectively.

Model	Params. P_M	Layers L_M	Hidden dim. D_M
Llama-3.2-1B-Instruct	1.0B	16	2048
Llama-3.1-8B-Instruct	7.5B	32	4096
Gemma-3-27B-IT	25B	62	4608
Qwen2.5-72B-Instruct	71B	80	8192

Graph-level cost. A TTS graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ consists of a set of executed LLM invocations, where each node $v \in \mathcal{N}$ corresponds to one model invocation. Each node is associated with model M_v, realized prompt length n_p^v, and completion length n_d^v. The prompt length depends on the graph structure: parallel nodes typically condition only on the original query, whereas refinement or composition nodes additionally condition on predecessor responses. We therefore compute graph cost using the realized token lengths of each executed node rather than approximating it solely by the number of nodes. The total FLOPs of executing $\mathcal{G}$ on query q are

$$f_{\text{cost}}(q, M, \mathcal{G}) = \sum_{v \in \mathcal{N}} f_{\text{inv}}\left(M_v, n_p^v, n_d^v\right). \tag{15}$$

For the single-model TTS graphs considered in our planner, $M_v = M$ for all $v \in \mathcal{N}$. This formulation accounts for both additional model invocations and the longer contexts introduced by multi-parent composition and sequential refinement.

Normalized budget units. Raw FLOPs are difficult to interpret across heterogeneous model sizes and TTS graphs. We therefore normalize computation on a per-query basis using the realized single-call probe from the smallest model as the reference unit. Let $\mathcal{G}_{\text{single}}$ denote the one-node graph corresponding to the 1B probe generation and $M_1 = 1\text{B}$ the smallest model. For query q, we define

$$C(q, M, \mathcal{G}) = \frac{f_{\text{cost}}(q, M, \mathcal{G})}{f_{\text{cost}}(q, M_1, \mathcal{G}_{\text{single}})}. \tag{16}$$

Thus, one budget unit corresponds to the realized FLOPs of the 1B probe on the same query, and $C(q, M, \mathcal{G}) = 10$ means that the executed configuration consumes approximately ten times this compute. Per-query normalization controls for variation in query length and makes costs comparable across queries, models, and TTS graph structures.

Because the denominator uses the realized probe generation, another 1B single call need not cost exactly one unit if its completion length differs. On the BEST-Route-Mix test set, the mean realized single-call costs are 1.11, 9.49, 61.5, and 96.5 units for 1B, 8B, 27B, and 72B, respectively, with corresponding medians of 1.00, 7.77, 48.8, and 78.3. The mean–median differences primarily reflect query-dependent completion lengths.

Analytical cost estimates. While all reported costs are computed from realized token lengths, the planner must estimate the cost of additional generations before execution. We therefore estimate the single-call cost of each candidate model using the probe lengths of the current query. Specifically, let $n^p_{p,q}$ and $n^p_{d,q}$ denote the realized prompt and completion lengths of the probe for query q. For model M, we define

$$\widehat{C}_M(q) = \frac{f_{\mathrm{inv}}(M, \hat{n}_{p,q}, \hat{n}_{d,q})}{f_{\mathrm{inv}}\Big(M_1, n^p_{p,q}, n^p_{d,q}\Big)}. \tag{17}$$

where $\hat{n}_{p,q}, \hat{n}_{d,q}$ are the estimated prompt and generation length of the current node.

Average user budget. A user-specified budget b denotes an *average per-query compute allowance*, rather than a strict cap on each query:

$$\frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} C\big(q, M^\pi_q, \mathcal{G}^\pi_q\big) \le b, \tag{18}$$

where M^π_q and $\mathcal{G}^\pi_q$ denote the routed model and realized TTS graph for query q. This formulation enables query-adaptive compute allocation: easy queries may terminate after approximately one budget unit, whereas harder queries may consume substantially more compute through larger models or additional TTS computation. The routing policy can therefore redistribute compute across queries while satisfying the target budget on average.

Cost accounting protocol. We report the test-time compute of the selected TTS execution and exclude auxiliary routing overhead, including probe generation and verifier scoring. This protocol isolates the compute efficiency of TTS allocation and measures how effectively additional LLM generation compute is converted into performance gains. The excluded overhead is comparatively small: probe generation costs approximately one normalized unit and can be reused as an initial candidate without additional generation (when reused, its generation cost is counted toward the total budget), while the proxy verifier, a DeBERTa-v3-large model (He et al., 2023) with approximately 0.44B parameters, costs less than 0.5 normalized units per candidate. We therefore focus the primary budget on LLM generation cost incurred by the selected TTS execution.

Alternative cost metrics. Although FLOPs are our primary cost metric, the framework can accommodate other deployment-relevant metrics by replacing the invocation-level cost function. For example, API cost can be computed from model-specific input and output token prices and aggregated over all executed invocations. We evaluate this setting using dollar cost. Other metrics, such as wall-clock latency, can be incorporated with metric-specific cost estimation and aggregation, while the routing and budget-selection framework remains unchanged.

C Datasets, Verifier, Baselines, and Detailed Experiment Setting

C.1 Datasets

We evaluate query-adaptive TTS routing on multiple benchmarks covering open-ended instruction following, mathematical reasoning, knowledge-intensive multiple-choice questions, and out-of-distribution conversational tasks. Among them, BEST-Route-Mix is our primary benchmark for empirical analysis and evaluation. MATH and MMLU are used to train and evaluate task-specific routers, while MT-Bench evaluates the OOD generalization of routing policies trained on the large-scale BEST-Route-Mix benchmark. Table 4 summarizes the dataset splits and evaluation protocols.

BEST-Route-Mix (Ding et al., 2025). BEST-Route-Mix is a mixed-domain benchmark of open-ended queries adapted from BEST-Route, covering general instruction following, reasoning and

preference-based prompts, code generation, and safety-related queries. Starting from the original mixed-domain dataset, which does not provide reference answers, we use GPT-5.6 (OpenAI, 2026) as the labeling LLM to generate reference responses and DeepSeek-V4 (Xu et al., 2026) as the correctness judge during offline profiling. We retain only queries for which at least one candidate action produces a correct response. During data construction, we additionally remove queries that remain unsolved by all profiled actions and are content-filtered during profiling. After applying these filtering, we obtain a fixed pool of $5,746$ queries. We perform an 80/20 split within each source using a fixed random seed, resulting in $4,604$ queries for offline profiling and router training and $1,142$ queries for evaluation. As BEST-Route-Mix contains open-ended queries for which exact-match evaluation is generally inappropriate, we assess response correctness using an LLM-based judge. Table 3 summarizes the benchmark composition across data sources.

Table 3: Composition of the BEST-Route-Mix benchmark after preprocessing.

Source	Train	Test	Share	Content
MixInstruct	3,189	798	69.4%	General instruction following
RewardBench	687	171	14.9%	Chat, reasoning, code, and safety preferences
CodeUltraFeedback	402	101	8.8%	Code generation
BeaverTails	326	72	6.9%	Safety and helpfulness prompts

MATH (Hendrycks et al., 2021b). MATH is a mathematical reasoning benchmark spanning seven problem types and five difficulty levels. We profile $1{,}250$ problems in total, using 750 problems from the official training split for offline profiling and router learning and 500 problems from the official test split for evaluation. The training subset contains 52, 125, 182, 163, and 228 problems from Levels 1–5, respectively, while the test subset contains 42, 88, 122, 118, and 130. Both subsets cover all seven problem types. We evaluate the generated final answers using exact match. For router and cascade selection, we use 5-fold out-of-fold predictions over the 750 training problems.

MMLU (Hendrycks et al., 2021a) is a multiple-choice benchmark covering 57 subjects across STEM, humanities, social sciences, and other domains. We use $1{,}385$ questions in total, with 885 for training and 500 for evaluation. The training set consists of all 285 development questions (5 per subject) and 600 questions sampled from the official test set, while another 500 sampled test questions are held out for evaluation. Specifically, we first select $1{,}100$ questions from the full $14{,}042$-question test set using a fixed pseudo-random shuffle, and then split them into 600 training and 500 evaluation examples. This procedure preserves broad coverage of the 57 subjects. Router and cascade selection use 5-fold out-of-fold predictions over the training questions. We evaluate predictions by exact match on the answer option.

MT-Bench (Zheng et al., 2023). MT-Bench is a multi-turn conversational benchmark containing diverse questions involving reasoning, knowledge, coding, and other capabilities. We use all 80 first-turn questions exclusively for out-of-distribution (OOD) evaluation and do not use any MT-Bench examples for profiling, router training, or calibration. This setting evaluates whether routing policies learned from the in-distribution benchmarks generalize to previously unseen query distributions.

C.2 Model Pool

Our model pool consists of Llama-3.2-1B, Llama-3.1-8B (Grattafiori et al., 2024), Gemma-3-27B (Team, 2025), and Qwen-2.5-72B (Yang et al., 2024), denoted as 1B, 8B, 27B, and 72B, respectively.

LLM generation setting. We query LLMs through OpenRouter (OpenRouter, 2026) using the same zero-shot prompt. Independent samples for probes and width/depth-based methods use temperature 0.7 and the LLM judge uses 0. Probe responses are capped at $2{,}048$ tokens ($4{,}096$ on MATH/MMLU); subsequent calls use a cap based on the probe length plus 512 tokens. All generations and judge verdicts are cached to ensure identical samples across methods and ablations.

C.3 Grading

Open-ended responses are evaluated by an LLM judge (DeepSeek-V4) given the query, reference answer, and candidate response. The judge is instructed to accept substantively correct and on-topic

Table 4: Detailed dataset splits used in our experiments. **Train** denotes queries available for offline profiling and router learning, while **Test** denotes held-out queries used only for evaluation.

Dataset	Train	Test	Split construction	Evaluation
BEST-Route-Mix	4,604	1,142	Fixed 80/20 split	LLM judge
MATH	750	500	Fixed subsets of the official train and test splits	Exact match
MMLU	885	500	285 dev + 600 sampled test questions for profiling; 500 sampled test questions for evaluation.	Exact match
MT-Bench	–	80	All 80 first-turn questions	LLM judge

responses regardless of length or style, while rejecting over-refusals. Verdicts are cached for each (judge, query, response) tuple. MATH and MMLU are evaluated by exact match on the boxed answer and option letter, respectively. For MT-Bench, we likewise use DeepSeek-V4 as the judge together with the benchmark-provided FastChat single-answer evaluation prompts. For the 30 questions with GPT-4 reference answers, we use the reference-guided `single-math-v1` prompt, which instructs the judge to compare the assistant response against the reference answer and identify any errors. For the remaining 50 questions, we use the standard `single-v1` prompt, which evaluates helpfulness, relevance, accuracy, depth, creativity, and level of detail. The judge outputs both an explanation and a `Rating`; the final bracketed number is taken as the 1–10 score. We consider responses with a score of at least 7 to be correct.

C.4 Detailed Preliminary Experiment Setting

We use the BEST-Route-Mix dataset, a name we adopt for convenience to refer to the dataset introduced in Best-Route (Ding et al., 2025). It comprises $5{,}746$ open-ended queries from four sources, covering general instruction following, chat and reasoning preferences, code generation, and safety-related tasks (Appendix C.1). We split the dataset into $4{,}604$ training queries and $1{,}142$ test queries.

For each training query, we evaluate three scaling strategies on each model under a common global search budget B_q: single call, width-leaning search, and depth-leaning revision chain. The searches are performed without access to ground-truth labels. After execution, we use DeepSeek-V4 (Xu et al., 2026) as an LLM judge to assess answer correctness (Appendix C.3) and record the realized compute cost, yielding correctness labels $Y_{q,a}$ and costs $C_{q,a}$ for 12 candidate actions per query.

We define the *cheapest-correct oracle configuration* as the lowest-cost action that returns a correct answer: $c(q) = \arg\min_{a:Y_{q,a}=1} C_{q,a}$. We analyze its distribution along three dimensions: (i) *model choice*; (ii) *scaling preference* (single, width-leaning, or depth-leaning); and (iii) *graph structure*, characterized by width–depth combinations. For example, W5D1 denotes five parallel candidates without revision, whereas W2D2 denotes two parallel candidates followed by two refinement rounds. To assess the stability of oracle configurations, we further stratify a subset of 300 training queries and repeat the profiling procedure five times with independent samples.

Predicted vs. On-the-Fly TTS Graph Construction. We conduct the graph-expansion study on the BEST-Route-Mix test queries using 1B and 8B models with verifier-argmax readout. For width-lean expansion, TTS-PLANNER-W generates up to five parallel responses per query at temperature 0.7; for depth-lean expansion, TTS-PLANNER-D starts from two responses and performs feedback-and-refinement rounds, with up to three rounds for 1B and one for 8B. Ground-truth correctness is evaluated only after execution and is never exposed during graph construction. We compare three label-free strategies: *fixed graph*, which applies the same graph size to all queries; *query-predicted graph*, which predicts the graph size from the query using a frozen MiniLM encoder and a two-layer MLP and keeps it fixed during execution; and *on-the-fly graph*, which adaptively expands and stops according to verifier scores. For a fair comparison, the query-predicted and on-the-fly strategies are matched in average compute. We additionally report a *per-query oracle* that uses ground-truth labels to select the smallest successful graph as a reference. To assess verifier feedback as a stopping signal, we measure the marginal gain from further expansion under low- and high-verifier-score regimes.

Hyperparameters. We set the improvement threshold to $\epsilon = 0.35$ and allow each node to have at most four predecessors. The maximum width is 16, with an expansion batch size of 4. For budget calibration, we search over a λ grid defined on three linear intervals: 61 points in $[0, 0.003]$, 54

points in $(0.003, 0.03]$, and 27 points in $(0.03, 0.3]$. The router input is 769-dimensional, consisting of two 384-dimensional embeddings and one probe score. The router is implemented as an MLP with two 256-dimensional hidden layers, ReLU activations, and a dropout rate of 0.1. We train the router using Adam with a learning rate of 10^{-3}, a batch size of 64, and 120 epochs. The verifier is initialized from DeBERTa-v3-large (He et al., 2023) and fine-tuned using AdamW with a learning rate of 10^{-5}, a batch size of 8, and 2 epochs.

Prompts. We use the following prompt templates for generation, fusion, and self-refinement.

Generation Prompt

You are a careful expert assistant. Answer the request correctly and completely. If the request is unsafe or harmful, refuse briefly.

Fusion Prompt

You are a careful expert assistant. Compare the candidate answers, resolve their disagreements, fix errors, and produce a single improved, correct final answer. If the request is unsafe, refuse.
User turn:

```
REQUEST: query
CANDIDATE ANSWERS:
[Candidate 1] ...
[Candidate k] ...
Produce a reconciled, improved, correct final answer.
```

Refinement Prompt

You are a careful expert assistant. Generating the FEEDBACK, produce an improved, correct and complete final answer to the REQUEST. Output only the final answer. If the request is unsafe or harmful, refuse briefly.
User turn:

```
REQUEST: ...
PRIOR ANSWER: ...
FEEDBACK:
IMPROVED FINAL ANSWER:
```

C.5 More Preliminary Experiment Analysis

This instability is particularly pronounced for configurations involving additional TTS. As shown in Fig. 8, while single-call labels exhibit 84% pairwise agreement, width-leaning and depth-leaning configurations are reproduced in only 19% and 24% of cases, respectively, often becoming single-call configurations upon re-profiling. This variation can be traced to the generation outcomes themselves: even for a fixed model and a single-call strategy, correctness labels across repetitions exhibit Cohen's κ values of only 0.4–0.55. Thus, oracle-configuration instability is not solely a consequence of configuration search but is also closely associated with stochastic generation outcomes.

C.6 Baselines

We compare four categories of baselines: (1) *Vanilla*, which applies a fixed model to all test queries: always-1B, always-8B, always-27B, and always-72B; (2) *routers and cascades*, which select a model or escalation path for each query, including RouteLLM (Ong et al., 2025), Hybrid LLM (Ding et al., 2024), AutoMix (Aggarwal et al., 2024), and FrugalGPT (Chen et al., 2024a); (3) *fixed TTS*, which applies a predetermined scaling configuration to all queries under a given budget, including best-of-n, self-consistency (Wang et al., 2023), Self-Refine (Madaan et al., 2023), and budget-scaled fixed graphs (Wang et al., 2025a); and (4) *adaptive TTS*, which adjusts the test-time computation for each query, including BEST-Route (Ding et al., 2025) and BG-MCTS (Miyamoto et al., 2026). All

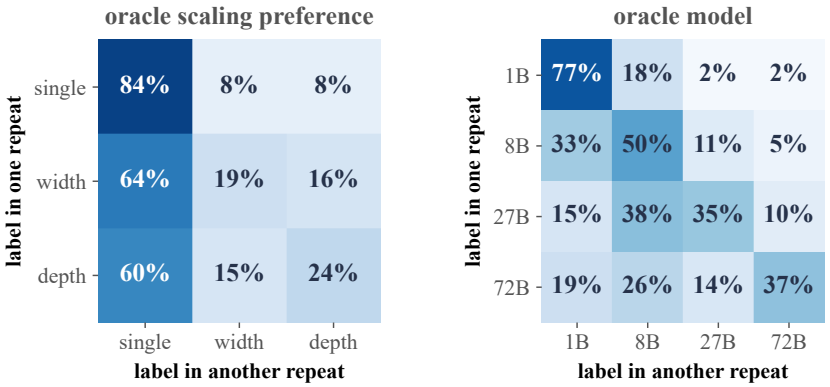


Figure 8: Reproducibility of the oracle configuration: row-normalised probability that a query labelled with the row category in one re-profiling receives the column category in another.

baselines use the same model pool, cached generations where applicable, verifier $\mathcal{V}$, and judge as TTS-ROUTE. Every answer-generating call is charged in 1B-call units. In contrast, router, scorer, and verifier forward passes are treated as routing overhead and excluded from the cost, consistent with TTS-ROUTE. All configurations selected on the training or validation split are frozen before the test set. We detail the baseline adaptations below.

Vanilla. Each baseline calls a fixed model once per query using the task prompt. The 1B/8B resulting operating points also define the model anchors used to compute APGR.

RouteLLM (Ong et al., 2025) learns a binary routing policy between a weak and a strong model from human preference data. Among four released variants, we use the causal-LM router, a fully fine-tuned Llama-3-8B model that assigns each query a rating from 1 to 5. Following the official implementation, we score each query once using its prompt template and compute the routing score from the rating-token distribution as $1 - P(\text{rating} \geq 4)$, without retraining the router. To accommodate our model pool, we partition this score using three jointly calibrated thresholds and route each query to 1B, 8B, 27B, or 72B. Thresholds are selected on the calibration split for each target budget. RouteLLM uses only the query text to select a single model and does not perform TTS.

HybridLLM (Ding et al., 2024) trains a DeBERTa router to estimate whether a smaller model will perform at least as well as a larger model, invoking the larger model when this probability falls below a calibrated threshold. Rather than using deterministic single-sample labels, HybridLLM defines the training target as the fraction of sampled responses for which the smaller model matches or outperforms the larger model. We implement its pairwise ranker with DeBERTa-v3-small, using judge-assessed correctness as the quality signal and five responses from each smaller model to estimate its probabilistic target relative to 72B. We extend the original binary formulation to four single-call candidates (1B, 8B, 27B, and 72B) and optimize a multi-candidate probabilistic loss that predicts the quality gap between each smaller model and 72B. At inference time, the decoder selects the least expensive model whose predicted gap does not exceed a threshold calibrated on the validation split. HybridLLM is a query-only model selector and does not perform TTS. Unlike the pretrained router used by RouteLLM, its router is trained separately for each target task.

AutoMix (Aggarwal et al., 2024) is a model cascade in which a smaller model first generates an answer and estimates its confidence through repeated few-shot self-verification. A POMDP-based meta-verifier then determines whether to accept the answer or escalate the query to a larger model. It discretizes confidence into bins and uses training data to estimate whether queries in each bin are solvable by the smaller model, only by the larger model, or by neither. The action for each bin maximizes expected reward under a tunable cost–quality trade-off. We extend the original two-model cascade to 1B→8B→27B→72B. To ensure a consistent confidence signal across methods, we replace prompt-based self-verification with the sigmoid-transformed score of our proxy verifier. Following the original setup, we discretize confidence into ten bins and fit a POMDP action table on the training split for each trade-off coefficient, specifying whether to accept the current answer or escalate to a larger model at each confidence level. The costs escalating from small to large models are accumulated. AutoMix adaptively escalates across models but does not allocate additional test-time computation within a model.

FrugalGPT (Chen et al., 2024a) learns an ordered model cascade and a stage-specific acceptance threshold under a target budget. After each model generates an answer, a scorer estimates its correctness from the query–answer pair; the answer is returned if its score exceeds the corresponding threshold, and the query is otherwise escalated to the next model in the sequence. The final-stage answer is returned unconditionally. We reimplement the cascade over our model pool by enumerating all model sequences of length at most three, allowing intermediate tiers to be skipped, and searching stage-specific thresholds over an empirical quantile grid. For each target budget, the highest-performing feasible combination of model sequence and thresholds is selected on the training split. To ensure a consistent confidence signal across methods, we replace the original DistilBERT scorer with our proxy verifier. The costs of all models invoked along the cascade are accumulated.

Best-of-n and self-consistency (Wang et al., 2023) generate n independent responses from a model. Best-of-n returns the response with the highest proxy-verifier score, whereas self-consistency normalizes the extracted answer keys and returns their plurality answer without using the verifier. We apply both methods to 1B and 8B, with candidate pools of up to 90 and 10 responses, respectively. For each target budget, we select the largest feasible n according to the mean sampling cost. Both methods scale test-time computation exclusively through width expansion within a fixed model.

Self-Refine (Madaan et al., 2023) iteratively critiques and revises a model's own response. We implement it using our depth-oriented planner TTS-PLANNER-D on 1B and 8B. The procedure begins with two independently sampled responses and selects the one with the highest proxy-verifier score as the incumbent. At each subsequent step, the same model generates feedback on the incumbent and produces a revised response, which replaces the incumbent only if its verifier score improves. The process terminates after one non-improving revision or upon reaching the revision limit $K \in \{0, 1, 2, 3, \infty\}$, which serves as the budget-control parameter. All feedback and revision calls are included in the accumulated cost. In our method, each self-refinement node combines feedback generation and response refinement within a single model call.

Fixed graphs (Wang et al., 2025a) apply the same width$\times$depth collaboration graph to every query. A graph with width W and depth D contains W candidate responses per layer and D layers; each layer after the first fuses the preceding layer's responses to generate W new candidates. The final response is selected by verifier-argmax over all graph nodes. Following the original budget-scaled shape families, we consider the width-to-depth ratios $[1, 1]$, $[1, 2]$, and $[2, 1]$ on 1B. Given an integer scale factor k, these ratios correspond to $(W, D) = (k, k)$, $(k, 2k)$, and $(2k, k)$, respectively. For each target budget, we select the largest feasible k whose mean graph cost satisfies the budget. The selected graph is fixed across queries and always executes in full without early stopping.

BEST-Route (Ding et al., 2025) extends HybridLLM by jointly selecting a model and a fixed amount of test-time computation. For each candidate (M, N), where M is the model and N is the number of sampled responses, the router predicts its quality gap relative to the strongest candidate. The decoder selects the least expensive candidate whose predicted gap falls below a calibrated threshold, using a single call to the strongest model as the fallback. We apply its implementation to twelve candidates: 1B and 8B with $N \in \{1, \ldots, 5\}$, and single calls to 27B and 72B. For multi-sample candidates, the response with the highest proxy-verifier score is returned. Judge-assessed correctness provides the training signal, the gap threshold is calibrated on the validation split, and each candidate incurs the cost of its N generations. BEST-Route selects both the model and sampling width from a fixed action set using only the query text; it does not use probe evidence, early stopping, or iterative revision.

BG-MCTS (Miyamoto et al., 2026) dynamically expands a TTS graph based on a fixed model under a fixed per-query budget. Its budget-aware search policy controls node selection and the trade-off between width and depth according to the fraction of budget consumed: width expansion generates an independent response at the root, whereas depth expansion revises a selected response to create a child node. As the remaining budget decreases, the policy reduces width exploration and favors completing existing depth trajectories. We adapt the original reasoning-step search to our answer-level setting, where each node represents a complete response and its value Q is the sigmoid-transformed proxy-verifier score. The search budget aligns with our budget metrics. We retain the original hyperparameters and run the planner separately on 1B and 8B. BG-MCTS uses the assigned budget to adapt graph width and depth but does not select the backbone model.

C.7 AVERAGE PERFORMANCE-GAIN RATIO (APGR).

To summarize performance across a range of compute budgets, we report the average performance-gain ratio (APGR), which measures the fraction of the accuracy gap between weak and strong single-call anchors recovered by a method on average. Let A_W and A_S denote the accuracies of the weak and strong anchors, respectively, and let $\mathrm{acc}(b_k)$ denote the method accuracy at budget b_k. We define

$$\mathrm{APGR} = \frac{1}{K}\sum_{k=1}^{K}\frac{\mathrm{acc}(b_k) - A_W}{A_S - A_W}, \tag{19}$$

where $K = 100$ budgets are uniformly spaced between the compute costs of the weak and strong anchors. Higher APGR indicates a better overall performance–cost trade-off across the budget range: 0 corresponds to the weak-anchor accuracy on average, while 1 or 100% corresponds to the strong-anchor accuracy. APGR is not bounded to $[0, 1]$; it can be negative when performance falls below the weak anchor and exceed 1 when it surpasses the strong anchor. We use Llama-3.2-1B and Llama-3.1-8B as the weak anchor A_W, and Qwen-2.5-72B as the strong anchor A_S.

D PROOFS AND DERIVATIONS FOR STABILITY BOUNDS

Proof of Proposition 1. Fix a query q and let $\pi_q(a) = \Pr(L(q) = a \mid q)$ denote the oracle-action distribution over $\mathcal{A}$. Define $\pi_q^\star = \max_{a\in\mathcal{A}} \pi_q(a)$ and $\alpha_q = \sum_{a\in\mathcal{A}} \pi_q(a)^2$. For any deterministic predictor f, $\pi_q(f(q)) \le \pi_q^\star$. Taking expectations over the query distribution gives

$$\mathrm{Acc}(f) = \mathbb{E}_q[\pi_q(f(q))] \le \mathbb{E}_q[\pi_q^\star] = \mathrm{Acc}^\star, \tag{20}$$

with equality attained by the Bayes-optimal predictor $f^\star(q) = \arg\max_{a\in\mathcal{A}} \pi_q(a)$. For the lower bound, since $\sum_a \pi_q(a) = 1$,

$$\alpha_q = \sum_a \pi_q(a)^2 \le \pi_q^\star \sum_a \pi_q(a) = \pi_q^\star. \tag{21}$$

Taking expectations yields

$$A_{\mathrm{pair}} = \mathbb{E}_q[\alpha_q] \le \mathrm{Acc}^\star. \tag{22}$$

For the upper bound, $(\pi_q^\star)^2 \le \sum_a \pi_q(a)^2 = \alpha_q$, so $\pi_q^\star \le \sqrt{\alpha_q}$. Therefore,

$$\mathrm{Acc}^\star \le \mathbb{E}_q[\sqrt{\alpha_q}] \le \sqrt{\mathbb{E}_q[\alpha_q]} = \sqrt{A_{\mathrm{pair}}}, \tag{23}$$

where the second inequality follows from Jensen's inequality. To show that the lower bound is attainable, consider a hypothetical randomized predictor that independently samples an action $L_1(q) \sim \pi_q$ and uses it as its prediction. Against an independent oracle action $L_2(q) \sim \pi_q$,

$$\Pr(L_1(q) = L_2(q) \mid q) = \sum_{a\in\mathcal{A}} \pi_q(a)^2 = \alpha_q. \tag{24}$$

Its expected accuracy is therefore $\mathbb{E}_q[\alpha_q] = A_{\mathrm{pair}} = \mathrm{Acc}_{\mathrm{LB}}$. This construction establishes attainability for a randomized predictor with access to the oracle-action distribution, rather than a deployable query-only predictor. Finally, $\alpha_q = \pi_q^\star$ if and only if

$$\sum_{a\in\mathcal{A}} \pi_q(a)\big(\pi_q^\star - \pi_q(a)\big) = 0, \tag{25}$$

which holds exactly when all actions with nonzero probability have equal probability. Since $\pi_q^\star - \alpha_q \ge 0$ for every q, we have $\mathrm{Acc}^\star = A_{\mathrm{pair}}$ if and only if π_q is uniform over its nonzero support for almost every query under the query distribution. □

Estimating stability and Bayes accuracy. Suppose n independent oracle searches on query q produce action counts $\{k_a\}_{a\in\mathcal{A}}$, where $\sum_{a\in\mathcal{A}} k_a = n$. The pairwise-agreement estimator

$$\widehat{\alpha}_q = \frac{\sum_{a\in\mathcal{A}} k_a(k_a - 1)}{n(n-1)} \tag{26}$$

is unbiased for $\alpha_q = \sum_{a\in\mathcal{A}} \pi_q(a)^2$, since it equals the fraction of agreeing pairs among the $\binom{n}{2}$ pairs of independent oracle searches. Averaging over the repeated-query set $\mathcal{Q}_{\mathrm{rep}}$ gives $\widehat{A}_{\mathrm{pair}} = \frac{1}{|\mathcal{Q}_{\mathrm{rep}}|}\sum_{q\in\mathcal{Q}_{\mathrm{rep}}} \widehat{\alpha}_q$. We estimate the Bayes-optimal accuracy using the plug-in estimator $\widehat{\mathrm{Acc}}^\star = \frac{1}{|\mathcal{Q}_{\mathrm{rep}}|}\sum_{q\in\mathcal{Q}_{\mathrm{rep}}} \max_{a\in\mathcal{A}}(k_a/n)$, which is generally upward-biased for finite n.

Instability of the cheapest-correct label under independent outcomes. To illustrate why the cheapest-correct action can be unstable, consider K actions $\{a_1, \ldots, a_K\}$ ordered by increasing cost for a fixed query q, such that $C(q, a_1) < \cdots < C(q, a_K)$. Let $Y_{q,a_k} \sim \text{Bernoulli}(p_q(a_k))$ denote the correctness of action a_k, where $p_q(a_k) = \Pr(Y_{q,a_k} = 1 \mid q)$. Under the simplifying assumption that action outcomes are conditionally independent given q, define the cheapest-correct oracle label as $L(q) = a_k$ if a_k is the first successful action, and $L(q) = \varnothing$ if all actions fail. The induced oracle-action distribution is

$$\pi_q(a_k) = \Pr(L(q) = a_k \mid q) = p_q(a_k) \prod_{j<k} (1 - p_q(a_j)), \tag{27}$$

with

$$\pi_q(\varnothing) = \Pr(L(q) = \varnothing \mid q) = \prod_{k=1}^{K} (1 - p_q(a_k)). \tag{28}$$

Accordingly, the per-query pairwise stability is

$$\alpha_q = \sum_{k=1}^{K} \pi_q(a_k)^2 + \pi_q(\varnothing)^2 = \sum_{k=1}^{K} \left[p_q(a_k) \prod_{j<k} (1 - p_q(a_j)) \right]^2 + \left[\prod_{k=1}^{K} (1 - p_q(a_k)) \right]^2. \tag{29}$$

Under this model, the cheapest-correct label is deterministic only if all actions have zero success probability or the first action with nonzero success probability succeeds with probability 1; otherwise, the oracle label remains stochastic even when the underlying success probabilities $p_q(a)$ are fixed. As an illustration, substituting the benchmark mean success rates $(0.498, 0.611, 0.762, 0.803, 0.876, 0.945)$ yields an approximate oracle-label distribution of $(0.498, 0.307, 0.149, 0.037, 0.008, 0.001)$ and pairwise stability of 0.366. The measured stability of 0.628 is higher, reflecting the gap between this population-mean independence approximation and the actual query-dependent, potentially correlated action outcomes. Nevertheless, this analysis illustrates the underlying mechanism: when lower-cost actions have intermediate success probabilities, the cheapest-correct label can vary across executions even though the underlying per-action success probabilities remain unchanged.

E Depth-Lean Planner (TTS-Planner-D)

The depth-lean planner prioritizes sequential refinement while maintaining a fixed width of w responses per layer (we use $w = 2$). Starting from layer $\ell = 1$, it generates an initial set $\mathcal{R}_1 = \{r_{1,i}\}_{i=1}^{w}$ and selects the highest-scoring response $r_1^* = \arg\max_{r \in \mathcal{R}_1} \mathcal{V}(q, r)$ as the incumbent, with $s_1^* = \mathcal{V}(q, r_1^*)$. At each subsequent layer $\ell \geq 2$, the model generates w refined responses $\mathcal{R}_\ell = \{r_{\ell,i}\}_{i=1}^{w}$, where each response critiques and revises the incumbent $r_{\ell-1}^*$ conditioned on the query. We define the best verifier score through layer ℓ as

$$s_\ell^* = \max \left\{ s_{\ell-1}^*, \max_{r \in \mathcal{R}_\ell} \mathcal{V}(q, r) \right\}.$$

If $s_\ell^* - s_{\ell-1}^* > \epsilon$, the highest-scoring response $r_\ell^* = \arg\max_{r \in \mathcal{R}_\ell} \mathcal{V}(q, r)$ becomes the new incumbent and refinement continues; otherwise, the chain is considered converged. Expansion stops when improvement is negligible or the next refinement would exceed the query-level budget:

$$s_\ell^* - s_{\ell-1}^* \leq \epsilon \quad \text{or} \quad C_\ell + \widehat{C}_{\text{next}} > B_q, \tag{30}$$

where C_ℓ is the accumulated cost through layer ℓ and $\widehat{C}_{\text{next}}$ is the estimated cost of the next refinement. As in TTS-Planner-W, for an action using M_k, we set $B_q = \widehat{C}(q, M_{k+1}, \mathcal{G}_{\text{single}})$, so that scaling M_k remains cheaper than escalating directly to the next larger model M_{k+1}. Finally, the planner returns $\hat{r} = \arg\max_{r \in \mathcal{R}} \mathcal{V}(q, r)$.

F More Experimental Results

F.1 Detailed Ablation Studies.

Table 6 evaluates the contributions of verifier guidance, per-action prediction, probe evidence, Lagrangian selection, and the TTS action space. First, verifier guidance is important for compute-

Algorithm 2 Online inference with TTS-ROUTE

Require: Query q, target average budget b, model pool $\mathcal{M}$, action set $\mathcal{A}$, proxy verifier $\mathcal{V}$, router θ_{router}, calibrated coefficient $\lambda(b)$
Ensure: Final response $\hat{r}$
Generate probe response $r_q^p \leftarrow M_1(q)$
Compute verifier score $s_q^p \leftarrow \mathcal{V}(q, r_q^p)$
Construct router input $x_q \leftarrow [e(q); e(r_q^p); \bar{s}_q^p]$
$p_q(a) \leftarrow \theta_{\text{router}}(x_q, a)$ for all $a \in \mathcal{A}$
$\hat{a} = (\hat{M}, \hat{S}) \leftarrow \arg\max_{a \in \mathcal{A}} \left[p_q(a) - \lambda(b)\bar{C}_a\right]$
if $\hat{S} = \text{single}$ **then**
    return single-call response from $\hat{M}$
end if
if $\hat{S} = \text{width-lean}$ **then**
    return TTS-PLANNER-W$(q, \hat{M}, \mathcal{V})$
else
    return TTS-PLANNER-D$(q, \hat{M}, \mathcal{V})$
end if

Table 5: MATH and MMLU results at target average budgets. Each test set contains 500 examples. Entries report accuracy (%)@realized cost, with cost measured in units of a single 1B call. APGR is reported in percentage points using 1B and 8B as weak anchors. Dashes denote infeasible operating points, and boldface marks the best result in each column based on unrounded values.

Method	b=10	b=20	b=30	b=50	b=100	APGR$_{1b}$	APGR$_{8b}$
MATH (vanilla 8B: 30@7; 27B: 71@28; 72B: 69@79)							
TTS-ROUTE	**36**@9	**53**@16	**71**@28	**72**@41	**72**@41	**88**	**94**
BEST-Route	34@13	45@21	52@30	64@41	69@77	68	68
BG-MCTS	20@6	31@19	31@28	32@48	34@95	17	2
RouteLLM	34@10	42@18	52@26	61@35	69@73	68	68
HybridLLM	32@8	48@16	69@28	72@48	72@48	80	82
FrugalGPT	32@9	52@17	**71**@28	71@40	70@50	80	83
AutoMix	32@9	45@20	58@28	69@43	68@54	72	74
Best fixed TTS	30@7	34@14	36@21	43@35	43@35	33	23
MMLU (vanilla 8B: 61@8; 27B: 77@36; 72B: 79@80)							
TTS-ROUTE	61@10	69@16	**75**@26	**79**@46	**79**@80	**86**	**79**
BEST-Route	**63**@11	**69**@23	73@33	75@48	**79**@79	82	70
BG-MCTS	38@9	62@16	62@28	62@47	63@95	52	-5
RouteLLM	62@10	**69**@20	73@28	76@44	**79**@80	82	71
HybridLLM	61@9	67@16	73@27	78@50	**79**@80	82	73
FrugalGPT	61@8	67@17	70@27	**79**@46	**79**@80	81	69
AutoMix	–	65@17	70@27	77@45	**79**@100	75	63
Best fixed TTS	61@8	62@17	63@25	65@40	65@40	60	15

efficient graph expansion. Removing it reduces APGR$_{1b}$/APGR$_{8b}$ by $13/29$ points on BEST-Route-Mix and $8/10$ on MATH, as unproductive expansions consume compute without commensurate accuracy gains. Its effect on MMLU is negligible, consistent with MMLU benefiting less from additional TTS expansion. Second, per-action success prediction is critical for reliable action selection. Replacing it with direct optimal-action prediction decreases APGR by $11/17$ points on average, supporting Insight 2 that per-action success provides a more stable prediction target. Third, removing probe evidence decreases APGR by $7/18$, $9/12$, and $8/2$ points on BEST-Route-Mix, MATH, and MMLU, respectively, confirming that probe responses and verifier feedback provide useful query-specific evidence beyond query text. Fourth, replacing Lagrangian selection with threshold-based selection reduces APGR by $8/16$ points on average, demonstrating the benefit of explicitly balancing predicted success and compute cost when allocating budget across queries. Finally, removing

Table 6: Ablation results across three datasets. Each cell reports APGR_{1b}/APGR_{8b} in percentage; Avg. is the average across datasets, and Drop denotes the decrease from the full model.

Variant	BR-Mix	MATH	MMLU	Avg.	Drop
TTS-ROUTE (full)	86/76	88/94	86/79	87/83	–
Action-space ablations					
w/o depth actions	85/75	82/86	83/80	83/80	3/3
w/o width actions	83/73	82/89	81/78	82/80	5/3
w/o graph actions	82/72	81/86	80/77	81/78	6/5
Routing and planning ablations					
w/o Verifier Guidance	73/47	80/84	85/79	79/70	7/13
w/o Per-action Prediction	83/70	64/62	79/66	75/66	11/17
w/o Probe Evidence	79/58	79/82	78/77	79/72	8/11
w/o Lagrangian Selection	82/67	79/82	75/52	79/67	8/16

Table 7: BEST-Route-Mix results under dollar-budget constraints. Entries report test accuracy (%) at each budget, measured in USD per 1,000 queries using OpenRouter list prices; APGR is also reported in percentage points relative to the dollar-priced 1B/8B anchors. Single-call operating points are 1B (49 at \$0.004), 8B (77 at \$0.016), 27B (89 at \$0.124), and 72B (95 at \$0.137). Best results are bolded.

Method	Budget (\$/1k queries)						APGR_{1b}	APGR_{8b}
	.01	.02	.03	.05	.08	.12		
Best fixed TTS	56	77	77	80	82	82	63	17
FrugalGPT	66	**80**	83	87	91	94	82	68
RouteLLM	–	76	81	84	91	93	78	60
AutoMix	67	77	81	86	87	91	77	54
BEST-Route	59	72	81	83	87	90	73	45
TTS-ROUTE	**71**	78	**83**	**89**	**93**	**94**	**84**	**72**

either width- or depth-lean actions weakens performance, while removing both reduces APGR by $4/4$, $7/8$, and $6/2$ points across the three datasets. Depth expansion is particularly beneficial on MATH, whereas width expansion contributes more on BEST-Route-Mix and MMLU, indicating complementary scaling behaviors.

F.2 DETAILED GENERALIZATION TO OUT-OF-DISTRIBUTION QUERIES.

Following BEST-Route, we evaluate methods trained on BEST-Route-Mix directly on the 80 turn-one MT-Bench questions, defining correctness as a single-answer judge score of at least 7. For each target budget, all operating points selected on the BEST-Route-Mix calibration set are frozen; no parameters are tuned on MT-Bench. As shown in Table 8, TTS-ROUTE achieves the highest APGR ($84/79$), outperforming AutoMix ($80/68$) and BEST-Route ($77/63$). Its transferred λ also satisfies every target budget, whereas RouteLLM overshoots several low-budget targets. At individual budgets, BEST-Route leads at b=10, while TTS-ROUTE leads at b=20 and remains within one point of the best feasible learned method at b=30. At larger budgets, a fixed 27B refinement chain performs best because, on this dataset, 27B already matches 72B accuracy at less than half the cost. Given n=80, differences below approximately five percentage points fall within one standard error.

F.3 ACTION ALLOCATION ACROSS TARGET BUDGETS

Fig. 9 shows the action distribution over BEST-Route-Mix test queries at the operating point selected for each target average budget. We make two observations. *(1) Approximately* 14% *of queries are assigned to graph actions on average, although this fraction varies substantially across budgets.* The fraction of graph actions follows a U-shaped pattern, decreasing from 17% at b=5 to 8% at b=30, before increasing to 24% at b=100. Their utility also varies across budget regimes. For $b \leq 25$, width- and depth-lean actions are 7–13 percentage points more accurate than single actions using the same model, while costing less than half of a single call to the next larger model.

Table 8: Out-of-distribution results on MT-Bench. All methods are calibrated on BEST-Route-Mix and applied without adaptation. Entries report test accuracy (%)@realized cost, with cost measured in 1B-call units; APGR is reported in percentage points using the MT-Bench 1B/8B anchors. A † indicates that the realized cost exceeds the target budget after transfer. Boldface marks the best budget-feasible result. Single-call operating points are 1B (35@1.1), 8B (63@8.0), 27B (85@36.8), and 72B (85@78.6).

Method	Target budget						$APGR_{1b}$	$APGR_{8b}$
	5	10	20	30	50	75		
Best fixed TTS	35@2.1	35@2.1	68@16.2	68@16.2	**86**@47.1	**86**@47.1	67	42
FrugalGPT	**48**@4.6	63@8.0	64@9.6	65@14.4	75@41.7	79@56.2	73	53
RouteLLM	49@14.0†	63@18.8†	71@24.6†	80@31.0†	80@50.2†	**86**@71.2	70	59
AutoMix	**48**@4.1	61@8.3	63@11.1	71@17.4	80@32.0	81@56.2	80	68
BEST-Route	46@5.5†	**65**@9.7	72@18.8	**78**@28.8	78@49.8	83@68.0	77	63
TTS-ROUTE	45@4.3	61@7.3	**73**@13.4	**78**@18.8	83@41.0	84@62.1	**84**	**79**

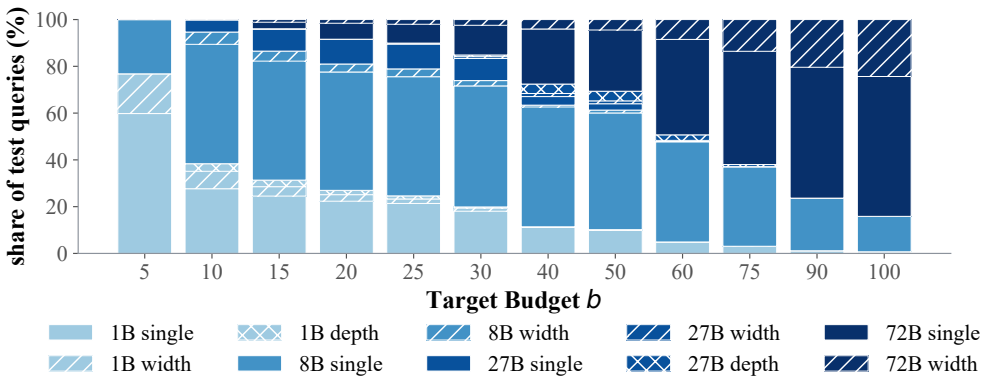


Figure 9: Action allocation on the BEST-Route-Mix test set across target average budgets. Each bar shows the fraction of queries assigned to each model–scaling action $a = (M, S)$. Hatched segments denote width- and depth-lean actions, while solid segments denote single actions.

Replacing these graph actions with the corresponding single actions reduces overall accuracy by 0.7–2.3 percentage points. For $b \geq 40$, graph actions are dominated by width-lean scaling with 72B but provide limited additional accuracy, indicating that the main TTS gains are concentrated in the low- and intermediate-budget regimes. At higher budgets, width-lean scaling primarily consumes the remaining available budget, while depth-lean actions remain rare, serving at most 4% of queries. *(2) As the budget increases, action allocation shifts toward larger models, while smaller models remain important.* At b=5, 1B and 8B serve 77% and 23% of queries, respectively. For b=10–50, 8B becomes the dominant model tier, serving 51–56% of queries, while the share of 1B gradually decreases and larger models are reserved for a smaller subset of queries. For b>50, allocation increasingly shifts toward 72B, whose share reaches 49%, 62%, and 84% at b=60, 75, and 100, respectively. The share of 27B peaks at approximately 11% for b=25–30, decreases to 3% at b=60, and reaches 0% at b=100. This is because its single-call cost is relatively close to that of 72B (61.5 versus 96.5 cost units), despite substantially lower accuracy; therefore, 27B becomes less attractive once 72B is affordable. Even at b=100, 8B still serves 15% of queries, showing a larger average budget does not justify uniformly selecting the largest model; effective compute allocation should remain query-adaptive.