PAY FOR HINTS, NOT ANSWERS: LLM SHEPHERDING FOR COST-EFFICIENT INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) deliver state-of-the-art performance on complex reasoning tasks, but their inference costs limit deployment at scale. Small Language Models (SLMs) offer dramatic cost savings yet lag substantially in accuracy. Existing approaches—routing and cascading—treat the LLM as an all-or-nothing resource: either the query bypasses the LLM entirely, or the LLM generates a complete response at full cost. We find that the LLM need not be used this way: even a truncated prefix of its response raises SLM accuracy, with gains that broadly grow with prefix length on all four benchmarks we study, also it hold across model families, tokenizers, and SLM sizes. We therefore introduce LLM Shepherding, which requests only a short prefix (a *hint*) from the LLM and lets the SLM produce its own complete response. Shepherding generalizes both routing and cascading, and it achieves lower cost under oracle decision-making. We develop a two-stage predictor that jointly determines whether a hint is needed and how many tokens to request. On GSM8K, CNK12, HumanEval, and MBPP, we compare shepherding with routing and cascading. Across full cost–accuracy sweeps, Reactive Shepherding attains the largest area under the cost–accuracy curve on all four benchmarks; at a fixed target of 90% of LLM accuracy, it is the cheapest method on all four; and at default operating points it reduces cost by 33–65% relative to LLM-only inference and delivers up to 4$\times$ cost reduction compared to state-of-the-art routing and cascading baselines. To our knowledge, shepherding is the first to formalize and learn a single hand-off prefix hint, controlled only through `max_new_tokens`, under a black-box per-token cost model.

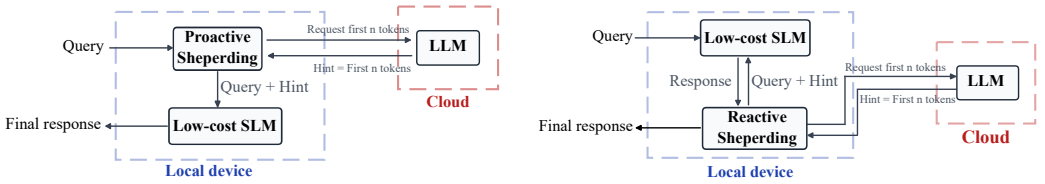
1 INTRODUCTION

LLMs excel on complex reasoning tasks, but their inference costs limit deployment at scale. Meanwhile, the emergence of highly capable open-source Small Language Models (SLMs) has created new opportunities for cost-efficient inference on edge devices, private servers, or shared datacenters. SLMs offer clear advantages: lower latency, improved privacy, and significantly smaller monetary and energy costs. However, for tasks involving logic and mathematics, the quality gap between SLMs and frontier LLMs (e.g., GPT, Gemini, Claude) remains substantial (Subramanian et al., 2025). Users and organizations thus face a difficult trade-off: accept inferior response quality to reduce costs, or pay a premium for LLM access to maintain accuracy.

This cost-quality trade-off motivates a central question: *How can we substantially improve SLM output quality while minimizing reliance on expensive LLMs?* The question is especially pressing in resource-constrained settings—edge devices with limited connectivity, latency-sensitive applications, and cost-conscious deployments—where every LLM call carries significant overhead. The goal is not merely faster inference, but fundamentally less LLM computation for a given quality target.

Two complementary paradigms have emerged to address this challenge: routing and cascading (Behera et al., 2025). *Routing* (Lu et al., 2024; Ding et al., 2024; Ong et al., 2025) uses a learned classifier to direct each query to exactly one model—either the SLM handles the query entirely, or it is forwarded to the LLM based on query complexity. *Cascading* (Chen et al., 2024; Aggarwal et al., 2024; Gupta et al., 2024) takes a sequential approach: the SLM attempts a response first, and the LLM is invoked only if the SLM response fails a confidence or correctness check.

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) Proactive Shepherding (b) Reactive Shepherding

Figure 1: Two modes of LLM Shepherding. Both request the first n LLM tokens as a hint for the SLM. (a) Proactive: decides upfront. (b) Reactive: decides after SLM failure.

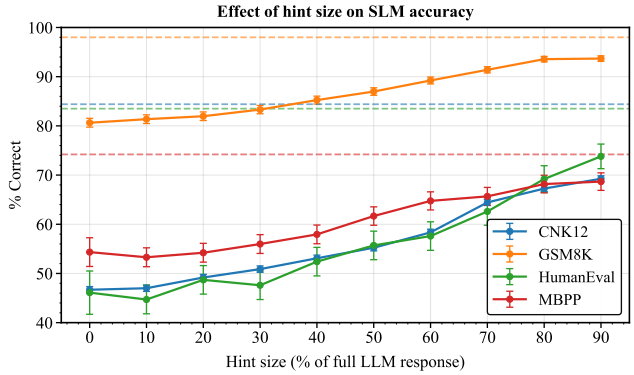


Figure 2: SLM (Llama-3.2-3B-Instruct) accuracy vs. LLM (Llama-3.3-70B-Versatile) hint size. Accuracy broadly rises with hint size, and on the math benchmarks the gains begin with the smallest hints. Dashed lines indicate LLM accuracy.

We observe that both routing and cascading share a fundamental limitation: they treat the LLM as an all-or-nothing resource. Either the query bypasses the LLM entirely, or the LLM generates a complete response, incurring the full LLM inference cost. This binary view overlooks a key opportunity: *most LLM service providers allow users to specify* `max_new_tokens`*, a user-defined limit for the number of output tokens*. This capability can be strategically leveraged to reduce LLM costs while simultaneously improving SLM answer quality.

We propose **LLM Shepherding**, a novel framework that complements routing and cascading by enabling *partial* LLM assistance. Rather than having the LLM produce complete responses, shepherding requests the LLM to output only a short prefix—first n tokens of the LLM response—called a *hint*. The hint is then concatenated with the query and is given to SLM for the final response. To validate whether LLM hints can benefit SLM inference, we evaluate on four widely-used benchmarks spanning mathematical reasoning—GSM8K (Cobbe et al., 2021) and CNK12 (OpenR1, 2025)—and code generation—HumanEval (Chen, 2021) and MBPP (Austin et al., 2021). We use Llama-3.2-3B-Instruct as the SLM and Llama-3.3-70B-Versatile as the LLM (Dubey et al., 2024). As shown in Figure 2, SLM accuracy rises with hint size on all four benchmarks. The rise is steady on the math benchmarks, whereas on the code benchmarks hints below roughly 40% of the response give no reliable gain and larger hints give large ones. We further verify in Appendix A that this broadly increasing, though not strictly monotonic, hint-size effect holds for cross-family SLM-LLM pairs, which suggests it is a property of prefix-guided SLM generation rather than of our specific setup.

As illustrated in Figure 1, we instantiate two modes of shepherding: 1) **Proactive Shepherding** (Figure 1a) is a *routing-based* shepherding, where a learned router directly predicts whether a hint is needed and how many tokens to request, and 2) **Reactive Shepherding** (Figure 1b) is a *cascading-based* shepherding, where the features of the SLM response(s) are used for cascading decision and hint prediction. Observe that proactive shepherding reduces to routing at the extremes: hint size zero (SLM only) or maximal (full LLM response). Similarly, reactive shepherding reduces to cascading when a complete LLM output is requested upon SLM failure. Shepherding thus generalizes both paradigms. By operating in the intermediate regime – requesting hints rather than full responses – shepherding theoretically achieves lower costs than either approach.

Realizing this vision in practice presents substantial challenges. First, the minimum hint size is zero-inflated across queries: on GSM8K, 80% of queries need no hint, while the conditional distribution over positive hint sizes spans 10–90% of the full LLM response. Second, the benefit of additional tokens is non-smooth—critical information may appear at specific token boundaries, causing abrupt quality transitions. Third, SLM accuracy does not increase steadily with hint length: on HumanEval it fluctuates within the error bars at small and intermediate budgets and peaks only at the largest one (Figure 2), so the minimum sufficient size cannot be read off a smooth curve.

These challenges motivate our two-stage prediction framework that jointly determines whether a hint is needed and how many tokens to request.

Contributions:

- We formalize LLM Shepherding and prove that, under oracle decision-making and a mild condition on token prices, it costs no more than routing and cascading (Section 2).
- We develop a two-stage prediction model that jointly predicts whether a query requires a hint and how many tokens to request, addressing key challenges including a non-smooth accuracy response and zero-inflated hint size distributions(Section 3).
- We empirically evaluate shepherding against state-of-the-art routing strategies (RouteLLM (Ong et al., 2025), GraphRouter (Feng et al., 2025)) and cascading strategies (FrugalGPT (Chen et al., 2024), ABC (Kolawole et al., 2025)) on mathematical reasoning (GSM8K, CNK12) and code generation (HumanEval, MBPP) benchmarks. Reactive Shepherding attains the highest AIQ on all four benchmarks, is the cheapest method at a 90% accuracy target on all four, and achieves the highest ACE at default operating points, with four times the cost reduction of routing baselines on the CNK12 and 1.5 times that of cascading baselines on HumanEval, the latter without any code-specific training (Section 4).

2 Problem Formulation

2.1 System and Notation

Let $\mathcal{Q}$ denote the set of all possible queries and let $q \in \mathcal{Q}$ denote an individual query. Let $\mathcal{X}$ denote a finite vocabulary of tokens and $\mathcal{X}^*$ denote the set of all finite token sequences over $\mathcal{X}$. We consider two language models: an open-source SLM $h_s : \mathcal{Q} \to \mathcal{X}^*$ deployed on a local device, such as a smartphone, laptop, or local server, and an LLM $h_l : \mathcal{Q} \to \mathcal{X}^*$ accessible through an LLM API.

For a token sequence $x \in \mathcal{X}^*$, we use $|x|$ to denote its number of tokens. For two sequences $x, y \in \mathcal{X}^*$, we denote their concatenation as $x \oplus y$. Given a sequence x and an integer $n \leq |x|$, $x_{[1:n]}$ denotes the prefix of x consisting of its first n tokens. We use $\mathbb{1}[\cdot]$ to denote the indicator function.

We define a quality function $\phi : \mathcal{Q} \times \mathcal{X}^* \to [0, 1]$ that measures the quality of a response $r \in \mathcal{X}^*$ to a query $q \in \mathcal{Q}$. Higher values indicate better response quality. In practice, ϕ may be instantiated as an exact-match accuracy metric (for tasks with ground-truth answers), a semantic similarity score, or a learned reward model. We define a quality threshold $\tau \in (0, 1]$ and $I_s(q) = \mathbb{1}_{[\phi(q,h_s(q))\geq\tau]}$ to indicate whether the SLM alone produces a satisfactory response.

2.2 Hints and Shepherding Policy

The central concept in LLM shepherding is the *hint*, defined as a prefix of the LLM's response provided to the SLM to guide its generation. Formally, for a query q and hint size $n \geq 0$, the hint is given by $\text{hint}(q, n) = h_l(q)[1 : n]$. The hint-augmented SLM response is then:

$$h_s^{(n)}(q) = h_s\big(q \oplus \text{hint}(q, n)\big), \tag{1}$$

where the SLM generates a response for the original query concatenated with the hint prefix.

Shepherding Policy. A *shepherding policy* $\pi : \mathcal{Q} \to \{0, 1, \ldots, N_{\max}\}$ maps query q to a hint size $n = \pi(q)$, where $N_{\max}$ is the maximum output token limit of the LLM. When $n = 0$, the SLM responds without assistance; when $n > 0$, the system requests an n-token hint before SLM

completion. For each query q, we define the *minimum hint size* as the smallest number of LLM tokens required for the SLM to produce a satisfactory response:

$$n^*(q) = \min\left\{n \in \mathbb{Z}_{\geq 0} : \phi\big(q, h_s^{(n)}(q)\big) \geq \tau\right\}. \tag{2}$$

The *Oracle Shepherding Policy* π^* maps each query q to $n^*(q)$.

2.3 Cost Model and Evaluation Metrics

We adopt a token-based cost model consistent with commercial pricing. Let c_l^{in} and c_l^{out} denote the cost per input token and output token for the LLM, respectively. Similarly, let c_s^{in} and c_s^{out} denote the corresponding costs for the SLM. Both LLM and SLM costs are treated as general non-negative quantities, capturing deployments where the SLM runs on metered local hardware, private infrastructure, or shared datacenters. Let $c_\pi(q)$ denote the cost that a policy π incurs on query q, and let $C_\pi = \sum_{q\in\mathcal{Q}} c_\pi(q)$ denote its total cost over $\mathcal{Q}$. For a shepherding policy, the explicit form of $c_\pi(q)$ differs between proactive and reactive shepherding, and Proposition 2.1 in Section 2.4 gives both forms under oracle decision-making. As reference points, the LLM-only and SLM-only strategies incur per-query costs $c_l(q) = |q|\, c_l^{\text{in}} + |h_l(q)|\, c_l^{\text{out}}$ and $c_s(q) = |q|\, c_s^{\text{in}} + |h_s(q)|\, c_s^{\text{out}}$, with total costs C_l and C_s obtained by summing over $\mathcal{Q}$.

Evaluation metrics. Let A_l and A_s denote the accuracies of the LLM and the SLM, and A_π the accuracy of a policy π. Every strategy we compare exposes hyperparameters that trade cost against accuracy, so we evaluate strategies at three levels, in the order used in Section 4: across the full operating range, at a fixed accuracy target, and at a single default operating point.

Full operating range. Sweeping a strategy's hyperparameters yields a set of policies Π, each giving an operating point (C_π, A_π). The strategy's cost–accuracy curve $A_\Pi(c) = \max\big(\{A_s\} \cup \{A_\pi : \pi \in \Pi,\ C_\pi \leq c\}\big)$ is the highest accuracy it attains within a total budget c, with the SLM-only accuracy as a floor (Section 4.2). We summarize each curve by its Average Improvement in Quality (AIQ) (Hu et al., 2024), the area under the curve normalized over the interval $[C_s, C_l]$ shared by all strategies:

$$\text{AIQ}(\Pi) = \frac{1}{C_l - C_s}\int_{C_s}^{C_l} A_\Pi(c)\, dc. \tag{3}$$

The interval starts at C_s, the cheapest budget at which the SLM-only policy, and hence the floor A_s, is attainable.

Fixed accuracy target. Many deployments must meet a fixed quality level, such as a service-level agreement. For a target accuracy $\bar{\tau}$, we therefore also compare the minimum cost each strategy needs to reach it (Section 4.3):

$$C_\Pi^*(\bar{\tau}) = \min_{\pi\in\Pi} C_\pi \quad \text{s.t.} \quad A_\pi \geq \bar{\tau}, \tag{4}$$

which is undefined when no configuration of the strategy reaches $\bar{\tau}$.

Single operating point. Finally, to compare strategies at their default settings (Section 4.4), we use Accuracy-per-Cost Efficiency (ACE):

$$\text{ACE}(\pi) = \frac{(A_\pi - A_s)/(A_l - A_s)}{(C_\pi - C_s)/(C_l - C_s)},$$

which measures the fraction of the SLM–LLM accuracy gap a policy recovers per unit of the SLM–LLM cost gap it spends. ACE coincides with the IBC metric of AutoMix (Aggarwal et al., 2024) up to a fixed normalization; Appendix B discusses its relation to IBC and PGR/APGR (Ong et al., 2025). ACE is undefined for the SLM-only policy, whose numerator and denominator both vanish; it is the reference point rather than a competitor.

2.4 Cost Analysis (Oracle)

We analyze the monetary cost of routing, cascading, and the two modes of shepherding Proactive and Reactive which under oracle decision-making. We assume the oracle has perfect knowledge of $I_s(q)$

and $n^*(q)$. Routing incurs either the full SLM cost or the full LLM cost (otherwise), but never both. Cascading always incurs the SLM cost first; if the SLM fails, it additionally incurs the full LLM cost.

Proactive Shepherding makes the hint decision upfront: when $n^*(q) = 0$ only the SLM is invoked; otherwise the system pays (i) the LLM input cost for processing the query, (ii) the LLM output cost for generating $n^*(q)$ hint tokens, and (iii) the SLM cost on the query augmented with the hint. Reactive Shepherding follows the cascading structure: it first invokes the SLM, and only when $I_s(q) = 0$ does it additionally pay the LLM input/hint-output cost and re-run the SLM on the augmented input. This yields the following proposition.

Proposition 2.1 (Oracle Costs)**.** *Under oracle decision-making, the monetary cost per query $q \in \mathcal{Q}$ for routing, cascading, Proactive Shepherding, and Reactive Shepherding is:*

$$c^*_{route}(q) = I_s(q)\,(|q|c_s^{in} + |h_s(q)|c_s^{out}) + \big(1 - I_s(q)\big)\,(|q|c_l^{in} + |h_l(q)|c_l^{out})$$

$$c^*_{casc}(q) = |q|c_s^{in} + |h_s(q)|c_s^{out} + \big(1 - I_s(q)\big)\,(|q|c_l^{in} + |h_l(q)|c_l^{out})$$

$$c^*_{proac\text{-}shep}(q) = \mathbb{1}_{[n^*(q)>0]}|q|c_l^{in} + n^*(q)c_l^{out} + (|q| + n^*(q))c_s^{in} + |h_s^{(n^*(q))}(q)|c_s^{out}$$

$$c^*_{reac\text{-}shep}(q) = |q|c_s^{in} + |h_s(q)|c_s^{out} + \big(1 - I_s(q)\big)\big[|q|c_l^{in} + n^*(q)c_l^{out} + (|q| + n^*(q))c_s^{in} + |h_s^{(n^*(q))}(q)|c_s^{out}\big],$$

where $h_s^{(n)}(q)$ is as defined in (1)*.*

The following corollary identifies the condition under which each mode of shepherding dominates its corresponding baseline (proof in Appendix C).

Corollary 2.2. *For every query $q \in \mathcal{Q}$ with $I_s(q) = 0$, both*

$$c^*_{proac\text{-}shep}(q) \le c^*_{route}(q) \quad \textit{and} \quad c^*_{reac\text{-}shep}(q) \le c^*_{casc}(q)$$

hold if and only if

$$\big(|h_l(q)| - n^*(q)\big)\, c_l^{out} \;\ge\; (|q| + n^*(q))\, c_s^{in} + |h_s^{(n^*(q))}(q)|\, c_s^{out}. \tag{5}$$

For queries with $I_s(q) = 1$, all four costs reduce to the SLM-only cost $|q|\, c_s^{in} + |h_s(q)|\, c_s^{out}$ and the inequalities hold with equality.

The condition in (5) means that shepherding pays less than its baseline whenever the savings from generating fewer LLM output tokens, $(|h_l(q)| - n^*(q))\, c_l^{\text{out}}$, outweigh the additional SLM cost incurred from processing the longer (query + hint) input and producing an SLM response. Since LLM output tokens are typically the dominant cost factor (with $c_l^{\text{out}} > c_l^{\text{in}}$ and $c_l^{\text{out}} \gg c_s^{\text{in}}, c_s^{\text{out}}$ in practice), (5) is satisfied for the vast majority of queries.

3 Training the Shepherding System

Our goal is to learn a policy $\pi_\theta : \mathcal{Q} \to \mathbb{Z}_{\ge 0}$ that predicts the minimum hint size $n^*(q)$ defined in Eq. (2). Direct prediction of $n^*(q)$ is challenging, because 1) **Zero-inflated hint sizes:** $n^*(q)$ has a large point mass at zero—on GSM8K, 80.6% of queries require no hint at all—while the conditional distribution over positive hint sizes spans 10% to 90% of the full LLM response with approximately exponential decay (see Appendix F); 2) **Non-smooth benefit:** a small change in n can abruptly determine whether the SLM crosses the quality threshold τ, as critical information may appear at specific token boundaries; 3) **Noisy accuracy curve:** measured accuracy does not increase steadily with n; on HumanEval it fluctuates within the error bars at small and intermediate budgets and peaks only at the largest one (cf. Fig. 2); and 4) **Tokenization sensitivity:** the same semantic content yields different token counts across models and query phrasings, making direct regression to absolute counts brittle. See Appendix D.1 for more discussion.

We address them with a two-step policy: first predict whether a hint is needed via a binary indicator $y(q) = \mathbb{1}_{[n^*(q)>0]}$, then regress the hint size on positive samples only. Note that $y(q) = 1 - I_s(q)$: a query requires a hint if and only if the SLM alone fails to produce a satisfactory response.

3.1 Training Data and Supervision Signals

For each query q, we construct supervision labels by evaluating candidate hints at percentage sizes $p \in \{0, 10, 20, \ldots, 90\}$ of the full LLM response length $|h_l(q)|$, computing $n_p = \lfloor \frac{p}{100} |h_l(q)| \rfloor$ and

querying the LLM with `max_new_tokens` $= n_p$ to obtain $\text{hint}(q, n_p) = h_l^{(n_p)}(q)$. We record

$$n^*(q) = \min\left\{n_p : \phi\big(q, h_s(q \oplus \text{hint}(q, n_p))\big) \geq \tau\right\},$$

falling back to $n^*(q) = |h_l(q)|$ if no partial hint meets the threshold. The full labeling procedure, outlier filtering (Appendix E), discretization rationale, and decoding configuration are detailed in Appendix D.2. The log-transformed regression target is $r(q) = \log(1 + n^*(q))$.

3.2 Model and Training Objective

We use DeBERTa-v3-large (He et al., 2023) as the text encoder and fuse the `[CLS]` embedding $\mathbf{h}_{\text{CLS}}(q)$ with transformed numeric features $\mathbf{g}(q) = \text{MLP}(\mathbf{f}(q))$ into $\mathbf{u}(q) = [\mathbf{h}_{\text{CLS}}(q); \mathbf{g}(q)]$. Two lightweight heads produce the hint-indicator logit $\widehat{y}(q) = \text{Head}_{\text{hint}}(\mathbf{u}(q))$ and the size estimate $\widehat{r}(q) = \text{Head}_{\text{size}}(\mathbf{u}(q))$, with multi-sample dropout (Inoue, 2019) on the indicator logit to reduce variance (details in Appendix D.3). The two heads are trained jointly with $\mathcal{L} = \lambda\,\mathcal{L}_{\text{hint}} + (1-\lambda)\,\mathcal{L}_{\text{size}}$. Here $\mathcal{L}_{\text{hint}}$ is the binary cross-entropy between $\sigma(\widehat{y}(q))$ and $y(q)$. $\mathcal{L}_{\text{size}}$ is the Smooth L1 (Huber) loss (Huber, 1964) between $\widehat{r}(q)$ and $r(q)$, averaged over the positive set $\mathcal{Q}^+ = \{q \mid y(q) = 1\}$ because the hint size is undefined for no-hint queries. The weight $\lambda \in [0, 1]$ is tuned on the validation set, and class imbalance is handled by weighted minibatch sampling. The AdamW/EMA setup and policy calibration are given in Appendix D.4.

3.3 Inference: Mapping Predictions to Decisions

At test time, let $\mathbb{P}(q) = \sigma(\widehat{y}(q))$ and apply threshold $\alpha \in [0, 1]$:

$$\pi_\theta(q) = \begin{cases} 0, & \text{if } \mathbb{P}(q) < \alpha, \\ \lfloor \text{clip}(\exp(\widehat{r}(q)) - 1,\ 0,\ N_{\max}) \rceil\,, & \text{if } \mathbb{P}(q) \geq \alpha, \end{cases}$$

where $\lfloor x \rceil$ denotes rounding to the nearest integer, $\text{clip}(x, a, b) = \max(a, \min(x, b))$ clamps values to $[a, b]$, and $N_{\max}$ is the maximum LLM output token limit. The parameter θ encompasses all learned components (backbone encoder, MLP, and prediction heads).

Hint Threshold. For queries with $\pi_\theta(q) = 0$, we return the SLM output $h_s(q)$ directly. For queries requiring hints ($\pi_\theta(q) > 0$), we further apply a hint-size threshold η_{hint} to distinguish low- from high-complexity queries:

- If $0 < \pi_\theta(q) \leq \eta_{\text{hint}}$, we query the LLM with `max_new_tokens` $= \pi_\theta(q)$ to obtain $\text{hint}(q, \pi_\theta(q)) = h_l^{(\pi_\theta(q))}(q)$, and return the shepherded response $h_s(q \oplus \text{hint}(q, \pi_\theta(q)))$;
- If $\pi_\theta(q) > \eta_{\text{hint}}$, the query is deemed high-complexity and routed directly to the LLM, returning $h_l(q)$ without invoking the SLM.

This threshold-based routing means that queries requiring extensive hints may be better served by the LLM's full reasoning capacity, avoiding the overhead of generating a lengthy hint only to invoke the SLM. The threshold η_{hint} is tuned on the validation set; see Appendix D.5 for further rationale.

3.4 Reactive Shepherding: Cascading Decision

For reactive shepherding, we add a training-free cascading module before the shepherding model, following the agreement-based approach of (Kolawole et al., 2025). We run the SLM K times ($K \in \{2, 3\}$, listed per dataset in Table 11) with temperature sampling ($T = 0.3$) to generate candidate answers $\{A_1, \ldots, A_K\}$; if at least k answers agree on the final answer, we return that answer directly, otherwise the query is passed to the shepherding model. We additionally incorporate features derived from the K responses—average predictive entropy, average output length, and query length—into the shepherding model's numeric input $\mathbf{f}(q)$. Intuitively, higher entropy signals greater SLM uncertainty and thus a stronger need for a hint, while output-length statistics provide a proxy for the verbosity and reasoning depth required by the query. The number of samples K and the agreement requirement k trade precision against routing rate. The values of K we use are listed in Appendix I, and Appendix J.1 sweeps k. Full feature definitions and design justifications are in Appendix D.6.

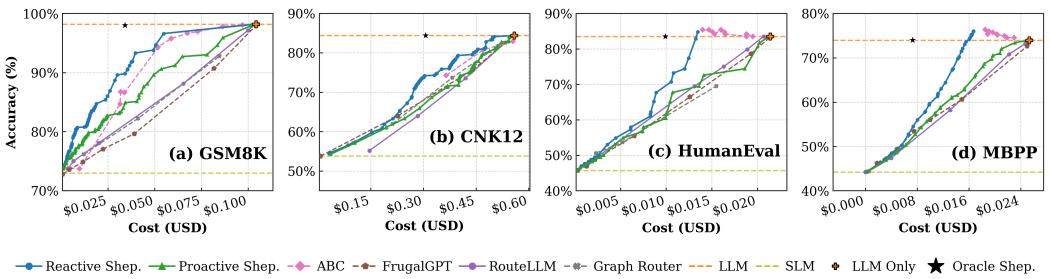


Figure 3: Cost–accuracy Pareto curves on the four benchmarks.

4 Performance Comparison

We compare the cost and accuracy of Proactive and Reactive Shepherding strategies with state-of-the-art LLM routing and cascading strategies. The routing baselines include RouteLLM (Ong et al., 2025) and GraphRouter (Feng et al., 2025), while the cascading baselines include FrugalGPT (Chen et al., 2024) and ABC (Kolawole et al., 2025). We use the vanilla Llama-3.2-3B-Instruct (without finetuning) as the SLM, deployed on a local server with an NVIDIA RTX 5090 GPU. For the LLM, we query Llama-3.3-70B-Versatile via the Groq API at $0.59 per 1M input tokens and $0.79 per 1M output tokens. The SLM is assigned a hardware-based cost of $0.0058 per 1M tokens, derived from RTX 5090 throughput measurements and cloud rental rates (details in Appendix G).

4.1 Datasets and Experimental Setup

Mathematical reasoning. GSM8K (Cobbe et al., 2021) contains 7,473 grade-school word problems for training and 1,319 for testing, each requiring multi-step arithmetic and paired with a numerical answer. We use 776 of the test questions for evaluation and the remaining 543 for validation. CNK12 is a Chinese K-12 mathematics benchmark drawn from OpenR1-Math-220k (OpenR1, 2025), covering algebra, geometry, and calculus from elementary through high school. We randomly sample 21,471 problems and split them into 18,251 for training, 1,073 for validation, and 2,147 for testing.

Code generation. HumanEval (Chen, 2021) has 164 programming problems, each with a function signature, docstring, and unit tests. MBPP (Austin et al., 2021) has 974 entry-level Python tasks, each with three test cases. Both are evaluated with the shepherding model trained on GSM8K, without any code-specific training, to test cross-domain transfer.

Inference and Evaluation. Test sets contain no pre-computed hints, and hints are requested at inference time as described in Section 3. A response counts as correct when its final numerical answer matches the ground truth (GSM8K, CNK12) or when it passes all unit tests (HumanEval, MBPP). Each query is scored by majority vote over 7 sampled trials. Decoding settings, prompt templates, and scoring details are given in Appendix H.2. We compare methods from the widest view to the narrowest: full cost–accuracy curves (Section 4.2), the minimum cost at a fixed accuracy target (Section 4.3), and a single default operating point per method (Section 4.4).

4.2 Pareto Cost–Accuracy Analysis and AIQ

Routing, cascading, and shepherding each expose knobs that trade cost against accuracy. We therefore sweep every method across its hyperparameter space and plot the resulting cost–accuracy curves on all four benchmarks (Figure 3). Reactive Shepherding lies on or above the baselines across most of the cost range, while Proactive Shepherding, which routes without SLM inference, remains above or on par with the routing baselines. On the coding benchmarks, both Reactive Shepherding and ABC exceed the LLM-only baseline due to the cascading mechanism analyzed in Appendix J.2.

Table 1 summarizes each sweep by its AIQ (Eq. (3)), the area under the budget envelope of its points in Figure 3. Reactive Shepherding attains the highest AIQ on all four benchmarks, and Proactive Shepherding the highest among routing-style methods. These comparisons span the whole accuracy range; we next fix a single accuracy level and compare what each method must spend to reach it.

Table 1: AIQ (%) of each method, computed from the operating points in Figure 3.

	Routing			Cascading		
Dataset	RouteLLM	GraphRouter	Proactive Shep.	FrugalGPT	ABC	Reactive Shep.
GSM8K	81.6	81.5	87.9	80.5	87.9	**91.5**
CNK12	62.0	63.3	66.1	62.4	62.0	**69.3**
HumanEval	58.1	56.9	63.0	59.0	59.6	**69.3**
MBPP	53.8	48.3	59.1	55.1	52.9	**63.3**

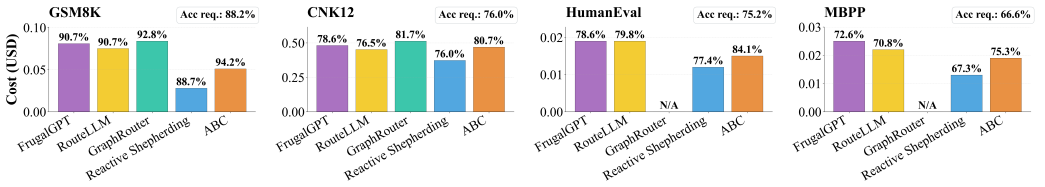


Figure 4: Minimum cost to achieve 90% of LLM accuracy on each dataset. Each method is tuned to its optimal operating point, which meets or exceeds the accuracy requirement (shown above bars). N/A marks methods that do not reach the requirement at any configuration.

4.3 Minimum Accuracy Requirement

For each method, we solve Eq. (4) with $\bar{\tau}$ set to 90% of LLM accuracy on each dataset. Figure 4 compares minimum costs, and Appendix I lists method configuration. Reactive Shepherding is cheapest across all benchmarks. On GSM8K (target 88.2%), it reaches 88.7% at \$0.028, 45% less than ABC (\$0.051), which overshoots at 94.2%. FrugalGPT reaches 90.7% but costs \$0.082, nearly three times as much. On CNK12 (target 76.0%), Reactive Shepherding costs \$0.37; ABC and FrugalGPT cost 27% and 30% more. On HumanEval (target 75.2%), it costs \$0.012, 20% below ABC and 37% below RouteLLM. On MBPP, it costs \$0.013, 32% below ABC (\$0.019).

4.4 Comparison at Default Operating Points

Finally, we report every method at a single default operating point. Tables 2–5 summarize results across four benchmarks. For the baseline routing and cascading strategies, we use default parameter settings released by the authors in their respective GitHub repositories. For Proactive and Reactive Shepherding, we present the values for η_{hint}, α, and K in Appendix I. Reactive Shepherding achieves the highest ACE on all datasets, ranging from 1.09 (CNK12) to 1.89 (GSM8K), while reducing costs by 33–65% relative to LLM-only inference. This ranking agrees with the AIQ and fixed-target comparisons above.

Comparison with routing. Routing baselines (RouteLLM, GraphRouter) struggle when query difficulty varies widely. On CNK12, they achieve only 5–8% cost reduction compared to LLM, whereas Reactive Shepherding delivers 33%, four times the reduction, while still meeting the 90% accuracy target. The gap is smaller on GSM8K, where most queries are SLM-solvable, but shepherding still nearly matches Oracle performance (\$0.036 vs. \$0.034) while RouteLLM incurs 78% higher cost.

Comparison with cascading. Cascading baselines (FrugalGPT, ABC) achieve higher accuracy than routing but at a greater cost. On GSM8K, ABC reaches 94.3% accuracy, 5.2 points above Reactive Shepherding—but costs 42% more, yielding lower ACE (1.75 vs. 1.89). This illustrates a key trade-off: cascading pays for accuracy that may exceed application requirements, while shepherding optimizes cost-efficiency by requesting only partial LLM assistance.

Cross-domain generalization. HumanEval tests whether shepherding transfers to unseen domains: all methods use GSM8K-trained models without code-specific fine-tuning. Reactive Shepherding achieves 77.4% accuracy while delivering 44% cost reduction, yielding the highest ACE (1.51).

Table 2: Cost–accuracy on GSM8K.
LLM ($0.104, 98.0%) | SLM ($0.0008, 73.0%).

Strategy	Cost ($)	Acc. (%)	Red. (%)	ACE
Routing				
RouteLLM	0.064	88.1	38.5	0.99
GraphRouter	0.041	82.2	60.6	0.94
Proactive Shep.	0.036	80.9	65.4	0.93
Cascading				
FrugalGPT	0.039	79.6	62.5	0.71
ABC	0.051	**94.3**	51.0	1.75
Reactive Shep.	**0.036**	89.1	**65.4**	**1.89**
Oracle Shep.	0.034	98.0	67.3	3.11

Table 3: Cost–accuracy on CNK12.
LLM ($0.559, 84.4%) | SLM ($0.006, 53.8%).

Strategy	Cost ($)	Acc. (%)	Red. (%)	ACE
Routing				
RouteLLM	0.533	**82.6**	4.7	0.99
GraphRouter	0.513	81.7	8.2	0.99
Proactive Shep.	0.489	80.6	12.5	1.00
Cascading				
FrugalGPT	0.440	75.6	21.3	0.91
ABC	0.469	80.7	16.1	1.05
Reactive Shep.	**0.373**	76.0	**33.3**	**1.09**
Oracle Shep.	0.309	84.4	44.7	1.83

Table 4: Cost–accuracy on HumanEval.
LLM ($0.0214, 83.5%)| SLM($0.00019, 45.7%).

Strategy	Cost ($)	Acc. (%)	Red. (%)	ACE
Routing				
RouteLLM	0.020	83.5	6.5	1.07
GraphRouter	0.015	69.5	29.9	0.90
Proactive Shep.	**0.011**	67.7	**48.6**	1.14
Cascading				
FrugalGPT	0.019	78.66	11.2	0.98
ABC	0.015	**84.15**	29.9	1.46
Reactive Shep.	0.012	77.4	43.9	**1.51**
Oracle Shep.	0.009	83.5	57.9	2.41

Table 5: Cost–accuracy on MBPP.
LLM ($0.025, 74.0%) | SLM ($0.00019, 44.2%).

Strategy	Cost ($)	Acc. (%)	Red. (%)	ACE
Routing				
RouteLLM	0.013	58.2	48.0	0.91
GraphRouter	**0.005**	49.0	**80.0**	0.83
Proactive Shep.	0.018	68.0	28.0	1.11
Cascading				
FrugalGPT	0.015	60.6	40.0	0.92
ABC	0.019	**75.3**	24.0	1.38
Reactive Shep.	0.014	69.8	44.0	**1.54**
Oracle Shep.	0.007	74.0	72.0	3.64

4.5 Latency Overhead: Proactive vs Reactive

We measured inference latency on CNK12 using Llama-3.2-3B-Instruct on an RTX 5090 GPU. The shepherding policy adds only 7.32 ms per query, negligible versus SLM generation (384.62 ms). Proactive Shepherding makes hint decisions before SLM inference with minimal latency. Reactive Shepherding first generates $K = 3$ SLM responses for consensus and features (1,163.48 ms), then runs the decision module (7.32 ms), a 0.6% overhead. Thus, while Reactive Shepherding achieves better cost–accuracy trade-offs, Proactive Shepherding makes decisions $159\times$ faster locally.

5 Conclusion

We have challenged the prevailing binary paradigm of routing and cascading, which treats LLMs as all-or-nothing resources. By introducing LLM Shepherding, we demonstrate that the most efficient path toward high-quality inference is not to choose between models, but to enable targeted collaboration. Our framework leverages token-level budget control, requesting only a short hint (prefix) from a frontier LLM to improve the accuracy of the SLM.

Our findings establish LLM Shepherding as a new approach for cost-efficient deployment. Across full cost–accuracy sweeps, Reactive Shepherding attains the highest AIQ on all four benchmarks, and at a fixed target of 90% of LLM accuracy it is the cheapest method on all four. At default operating points it achieves the highest ACE on every benchmark and reduces cost by 33–65% relative to LLM-only inference, up to four times the reduction of routing baselines and 1.5 times that of cascading baselines while meeting the 90% accuracy target. The predictor also generalizes: trained solely on math data, it transfers zero-shot to code generation, achieving the highest ACE on HumanEval and MBPP, and it transfers to a second model pair without retraining (Appendix A.2).

Our evaluation has focused on verifiable reasoning tasks where correctness can be automatically assessed; extending shepherding to open-ended tasks such as summarization or creative writing would require replacing the binary correctness function with learned reward models or human evaluation, which we leave for future work.

AI USE STATEMENT

In this work, we used generative AI tools for language polishing of author-written text and for assistance with LaTeX formatting. We have not used generative AI tools for research ideation, experimental design, data generation, or analysis of results. All AI-assisted text was reviewed and edited by the authors, and all technical claims, proofs, and experimental results were produced and verified by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide anonymized source code, training data generation scripts, and evaluation pipelines at `https://anonymous.4open.science/r/LLM_Sheperding-77C3/`. The problem formulation and all assumptions behind our cost analysis are stated in Section 2, with the complete proof of the dominance result in Appendix C. The construction of the training dataset, including hint labeling, outlier filtering, and decoding configurations, is detailed in Section 3 and Appendices D.2 and E. Prompt templates are listed in Appendix H, baseline configurations in Appendix I, and cost accounting details in Appendix G.

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. *Proc. NeurIPS*, 37:131000–131034, 2024.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

Adarsh Prasad Behera, Jaya Prakash Champati, Roberto Morabito, Sasu Tarkoma, and James Gross. Towards efficient multi-llm inference: Characterization and analysis of llm routing and hierarchical techniques. *arXiv preprint arXiv:2506.06579*, 2025.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023.

Guoxin Chen, Minpeng Liao, Peiying Yu, Dingmin Wang, Zile Qiao, Chao Yang, Xin Zhao, and Kai Fan. C-3po: Compact plug-and-play proxy optimization to achieve human-like retrieval-augmented generation, 2025.

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*, 2024. ISSN 2835-8856. URL `https://openreview.net/forum?id=cSimKw5p6R`. Featured Certification.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 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 V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *Proc. ICLR*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. 2024.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *Proc. ICLR*, pp. 26186–26203, 2025.

Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International journal of computer vision*, 129(6):1789–1819, 2021.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *Proc. ICLR*, 2024.

Neha Gupta, Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Aditya Krishna Menon, and Sanjiv Kumar. Language model cascades: Token-level uncertainty and beyond. In *Proc. ICLR*, 2024.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing, 2023.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Proc. ACL*, pp. 8003–8017, 2023.

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

Peter J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35 (1):73–101, 1964.

Hiroshi Inoue. Multi-sample dropout for accelerated training and better generalization. *arXiv preprint*, 2019.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. pp. 14165–14178, 2023.

Steven Kolawole, Don Dennis, Ameet Talwalkar, and Virginia Smith. Agreement-based cascading for efficient inference. *TMLR*, 2025. ISSN 2835-8856.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proc. ICML*, pp. 19274–19286, 2023.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *Proc. ICML*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proc. ICLR*, 2019.

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 *Proc. ACL*, pp. 1964–1974, 2024.

Daniel Morales-Brotons, Thijs Vogels, and Hadrien Hendrikx. Exponential moving average of weights in deep learning: Dynamics and benefits. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

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 *Proc. ICLR*, 2025.

OpenR1. Openr1-math-220k: A large-scale dataset for math reasoning tasks, 2025. URL https://huggingface.co/datasets/open-r1/OpenR1-Math-220k.

Shreyas Subramanian, Vikram Elango, and Mecit Gungor. Small language models (slms) can still pack a punch: A survey. *arXiv preprint arXiv:2501.05465*, 2025.

Mengwei Xu, Wangsong Yin, Dongqi Cai, Rongjie Yi, Daliang Xu, Qipeng Wang, Bingyang Wu, Yihao Zhao, Chen Yang, Shihe Wang, et al. A survey of resource-efficient llm and multimodal foundation models. *arXiv preprint arXiv:2401.08092*, 2024.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proc. ACL*, pp. 11263–11282, 2024.

A CROSS-MODEL VALIDATION OF THE HINT-SIZE EFFECT

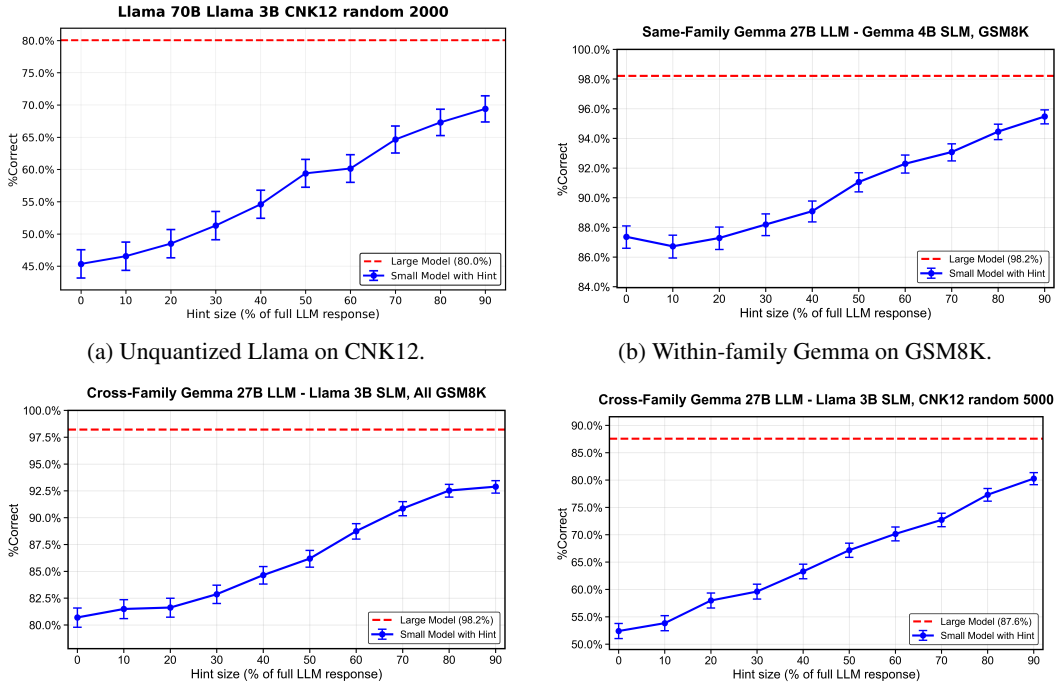
The motivating observation behind LLM Shepherding—that short LLM prefixes substantially improve SLM accuracy (Figure 2)—is established using a single Llama SLM-LLM pair. To check whether this effect is specific to that pair, we reproduce the hint-size vs. accuracy curve under three additional configurations on the mathematical reasoning benchmarks. These experiments validate the hint-size effect only, without training or evaluating the shepherding model. Appendix A.1 additionally varies the SLM size, and Appendix A.2 evaluates the full shepherding pipeline on a second model pair. The configurations are listed in Table 6.

Table 6: Additional SLM-LLM configurations used to validate the hint-size effect. All models are used off-the-shelf without any alignment fine-tuning.

Setting	SLM	LLM	Evaluation Data
Unquantized Llama	Llama-3.2-3B-Instruct	Llama-3.3-70B-Instruct†	2,000 CNK12 samples
Within-family Gemma	Gemma-3-4B-IT	Gemma-3-27B-IT	GSM8K (full)
Cross-family	Llama-3.2-3B-Instruct	Gemma-3-27B-IT	GSM8K (full)
Cross-family	Llama-3.2-3B-Instruct	Gemma-3-27B-IT	5,000 CNK12 samples

† Default open-source weights served via Amazon Bedrock, without quantization.

The *Unquantized Llama* setting replaces the Groq-served quantized LLM with the official open-source weights, isolating the effect of provider-side quantization. The *Within-family Gemma* setting swaps both models for a different family with a different tokenizer. The *Cross-family* setting pairs a Llama SLM with a Gemma LLM, so the hint prefix and the SLM use different tokenizers and were trained independently. For each configuration, we sweep the hint size from 0% to 90% of the full LLM response in 10% increments and measure SLM accuracy. Results are shown in Figure 5.



(a) Unquantized Llama on CNK12.

(b) Within-family Gemma on GSM8K.

(c) Cross-family on GSM8K.

(d) Cross-family on CNK12.

Figure 5: SLM accuracy as a function of hint size across alternative SLM-LLM configurations. The broadly increasing trend from Figure 2 persists across quantization, model family, and tokenizer changes.

In all four configurations, SLM accuracy increases approximately monotonically with hint size, matching the pattern in Figure 2. We therefore conjecture that the hint-size effect is a general property

of prefix-guided SLM generation, largely invariant to model family, tokenizer, and quantization, as long as both models are reasonably capable on the target task. This suggests LLM Shepherding can in principle be applied to heterogeneous SLM-LLM pairs. Appendix A.2 evaluates the full pipeline on the within-family Gemma pair, and we leave the full pipeline evaluation under cross-family pairs to future work.

A.1 Varying the SLM Size

The hint-size effect may also depend on the capability of the SLM itself. We repeated the hint-size sweep on the GSM8K test set (776 queries) with three SLMs of increasing size. All three read the same Llama-3.3-70B hints. Table 7 reports the results.

Table 7: SLM accuracy (%) as a function of hint size for three SLM sizes on GSM8K. All SLMs read the same Llama-3.3-70B hints.

Hint size	Llama-3.2-1B	Llama-3.2-3B	Llama-3.1-8B
0% (no hint)	42.8	77.5	84.8
10%	44.2	81.1	87.8
20%	49.5	81.2	86.2
30%	50.9	81.7	89.2
40%	52.1	84.8	89.8
50%	57.7	85.4	91.2
60%	61.9	86.6	93.3
70%	66.8	89.8	94.2
80%	74.0	93.0	94.8
90%	76.0	94.1	95.2
Gain (0→90%)	**+33.2**	**+16.6**	**+10.4**

Hints help at all three scales. The absolute gain shrinks as the SLM improves. Relative to each model's headroom to the LLM's 98.0%, however, the ordering changes: the 1B recovers about 60% of its gap, while the 3B and 8B recover 81% and 79%. The weakest model needs hints most but exploits them least, and even a 90% hint leaves it below what the 3B reaches unaided. At the other end, the 8B passes 89% at a 30% budget where the 3B needs 70%: a stronger SLM needs a shorter hint for the same accuracy. Shepherding therefore does not depend on a narrow capability window.

A.2 Full-Pipeline Validation on a Second Model Pair

We ran the full pipeline on a second model pair: Gemma-3-4B-IT as the SLM and Gemma-3-27B-IT as the LLM. We reused the GSM8K-trained predictor from the Llama pair, with no retraining.

Table 8 follows the protocol of Section 4.3: each method is reported at its cheapest configuration that reaches 90% of the LLM's accuracy. This target follows RouteLLM (Ong et al., 2025), which frames the routing objective as minimizing cost while reaching a set fraction of the stronger model's performance. Gemma-3-27B-IT scores 96.0% on the 776-query GSM8K test set, the same split used for every method in this table, so the target accuracy is 86.4%. The table groups methods by family: routing-style with Proactive Shepherding, then cascading-style with Reactive Shepherding. SLM-only and LLM-only are reference points.

Every method reaches the target, and Reactive Shepherding does so at the lowest cost, 43% below the next cheapest (ABC). Proactive Shepherding also reaches the target. Proactive mode acts on the predictor's output alone, with no consensus check to correct it, so meeting the target on an unseen model family shows that the hint-size decisions themselves transfer. Tables 4–5 apply the same GSM8K-trained predictor to a different task domain. Together, these results suggest the predictor captures query difficulty rather than properties of one model pair or one dataset.

Table 8: Minimum cost to reach 90% of the LLM's accuracy on GSM8K with the Gemma-3-4B-IT/Gemma-3-27B-IT pair, using the GSM8K-trained predictor from the Llama pair without retraining.

Method	Cost ($)	Accuracy
SLM-only	0.0012	86.21%
RouteLLM	0.0104	87.37%
GraphRouter	0.0226	88.79%
Proactive Shepherding	0.0280	88.02%
ABC	0.0102	90.04%
FrugalGPT	0.0229	88.14%
Reactive Shepherding	**0.0058**	89.50%
LLM-only	0.0563	96.0%

B Relation of ACE to Existing Cost-Efficiency Metrics

ACE (Section 2) summarizes each method at one operating point in the cost–accuracy plane. This appendix relates ACE to the metrics used in prior work and reports a complementary metric that does not depend on the choice of operating point. We do not claim ACE as a contribution of this work.

Relation to IBC. ACE coincides with the IBC metric of AutoMix (Aggarwal et al., 2024) up to a fixed normalization. Writing both definitions out,

$$\mathrm{ACE}(\pi) = \frac{(A_\pi - A_s)/(A_l - A_s)}{(C_\pi - C_s)/(C_l - C_s)} = \frac{\mathrm{IBC}_\pi}{\mathrm{IBC}_{\mathrm{BASE}}}, \qquad \mathrm{IBC}_{\mathrm{BASE}} = \frac{A_l - A_s}{C_l - C_s},$$

so the two metrics rank methods identically. IBC therefore cannot serve as an independent check on ACE, and it grows without bound near the SLM cost exactly as ACE does.

Relation to PGR and APGR. The RouteLLM metrics (Ong et al., 2025) are related but measure different things. PGR is the fraction of the weak-to-strong accuracy gap that a policy recovers, which is exactly the accuracy part (numerator) of ACE. APGR averages PGR as the routing threshold is swept, and its cost axis is the fraction of queries sent to the strong model. That axis does not fit our problem setup: a 25-token hint and a 124-token full response both count as one strong-model call, so the token-level saving that shepherding targets is invisible there.

Behavior near the SLM cost. ACE grows without bound as a method's cost approaches the SLM cost C_s, where its denominator vanishes. We never tune our methods to maximize ACE, and every operating point reported in Tables 2–5 sits far from that boundary: Reactive Shepherding costs 45× the SLM cost on GSM8K, and 62×, 63×, and 74× on CNK12, HumanEval, and MBPP. ACE is also normalized within each dataset, so we never compare ACE values across datasets.

AIQ as an operating-point-free check. As a check that does not depend on any chosen operating point, we compute AIQ (Hu et al., 2024), the normalized area under the cost–accuracy curve, over the Pareto sweeps of Appendix J, which cover every method across its full hyperparameter range. The oracle policy is an upper bound rather than a runnable method, so we omit it.

Reactive Shepherding has the highest AIQ on all four datasets, in agreement with the ACE comparison at the default operating points and with the Pareto curves of Appendix J. The three views are complementary: ACE summarizes the default operating points reported in the main tables, the Pareto curves compare methods at every quality level, and AIQ aggregates each curve into a single operating-point-free number.

C Proof of Corollary 2.2

We treat the two cases $I_s(q) = 1$ and $I_s(q) = 0$ separately.

Case 1: $I_s(q) = 1$**.** The SLM alone produces a satisfactory response, so $n^*(q) = 0$. Substituting into Proposition 2.1 and using $\mathbb{1}_{[n^*(q)>0]} = 0$ and $h_s^{(0)}(q) = h_s(q)$:

$$\begin{aligned} c^*_{\text{route}}(q) &= |q|\, c_s^{\text{in}} + |h_s(q)|\, c_s^{\text{out}}, \\ c^*_{\text{casc}}(q) &= |q|\, c_s^{\text{in}} + |h_s(q)|\, c_s^{\text{out}}, \\ c^*_{\text{proac-shep}}(q) &= |q|\, c_s^{\text{in}} + |h_s(q)|\, c_s^{\text{out}}, \\ c^*_{\text{reac-shep}}(q) &= |q|\, c_s^{\text{in}} + |h_s(q)|\, c_s^{\text{out}}. \end{aligned}$$

All four costs coincide, and both inequalities hold with equality.

Case 2: $I_s(q) = 0$**.** The SLM alone fails, so $n^*(q) > 0$ and the indicator $\mathbb{1}_{[n^*(q)>0]} = 1$. Substituting into Proposition 2.1:

$$\begin{aligned} c^*_{\text{route}}(q) &= |q|\, c_l^{\text{in}} + |h_l(q)|\, c_l^{\text{out}}, \\ c^*_{\text{casc}}(q) &= |q|\, c_s^{\text{in}} + |h_s(q)|\, c_s^{\text{out}} + |q|\, c_l^{\text{in}} + |h_l(q)|\, c_l^{\text{out}}, \\ c^*_{\text{proac-shep}}(q) &= |q|\, c_l^{\text{in}} + n^*(q)\, c_l^{\text{out}} + (|q| + n^*(q))\, c_s^{\text{in}} + |h_s^{(n^*(q))}(q)|\, c_s^{\text{out}}, \\ c^*_{\text{reac-shep}}(q) &= |q|\, c_s^{\text{in}} + |h_s(q)|\, c_s^{\text{out}} + |q|\, c_l^{\text{in}} + n^*(q)\, c_l^{\text{out}} \\ &\quad + (|q| + n^*(q))\, c_s^{\text{in}} + |h_s^{(n^*(q))}(q)|\, c_s^{\text{out}}. \end{aligned}$$

Proactive vs. Routing. The term $|q|\, c_l^{\text{in}}$ appears in both expressions. Subtracting it,

$$c^*_{\text{proac-shep}}(q) \le c^*_{\text{route}}(q) \iff n^*(q)\, c_l^{\text{out}} + (|q| + n^*(q))\, c_s^{\text{in}} + |h_s^{(n^*(q))}(q)|\, c_s^{\text{out}} \le |h_l(q)|\, c_l^{\text{out}}.$$

Rearranging gives exactly the dominance condition (5).

Reactive vs. Cascading. The term $|q|\, c_s^{\text{in}} + |h_s(q)|\, c_s^{\text{out}} + |q|\, c_l^{\text{in}}$ appears identically in both expressions. Cancelling it leaves the same inequality as in the previous case, which is again equivalent to (5).

Therefore, both $c^*_{\text{proac-shep}}(q) \le c^*_{\text{route}}(q)$ and $c^*_{\text{reac-shep}}(q) \le c^*_{\text{casc}}(q)$ hold if and only if (5) is satisfied.

D Training Shepherding System: Additional Details

D.1 Challenges in Predicting $n^*(q)$

Predicting $n^*(q)$ directly as an exact token count is challenging for several reasons.

Heavy-tailed hint sizes. $n^*(q)$ varies significantly across queries—on GSM8K, 80.6% of queries require no hint at all, while the remaining 19.4% span a wide range from 10% to 90% of the full LLM response (see Appendix F for a detailed discussion).

Non-smooth benefit. A small change in n can abruptly determine whether the SLM crosses the quality threshold τ, as critical information (e.g., a key constraint or intermediate step) may appear at specific token boundaries.

Noisy accuracy curve. Measured accuracy does not increase steadily with n (cf. Fig. 2). On HumanEval it dips at the 10%, 30%, and 60% budgets, each within the error bars, and peaks at the largest budget (75.6% at 90%). The data show a non-smooth increase, not that large hints hurt, so the minimum sufficient size cannot be read off a smooth trend.

Tokenization sensitivity. The same semantic content yields different token counts depending on the model and query phrasing, making direct regression to absolute token counts brittle.

These properties jointly motivate the two-step design (classify-then-regress) and the log-transformed regression target $r(q) = \log(1 + n^*(q))$ used in Section 3.

D.2 Training Dataset Construction Details

This appendix provides the complete methodology for constructing the training dataset used to supervise the Shepherding model, including the discretized token budget approach and labeling procedure.

Supervision via Discretized Token Budgets

We obtain supervision for the hint sizes by evaluating candidate hints at multiple granularities via token-budgeted LLM calls. For each training query q, we first query the LLM once to obtain a reference completion $h_l(q)$ and use its LLM-token length $|h_l(q)|$ to define percentage sizes $p \in \{0, 10, 20, \ldots, 90\}$. We choose 10% increments as they provide sufficient granularity to capture the non-smooth benefit curve of hint tokens while maintaining computational tractability—finer granularities (e.g., 5% steps) would double the labeling cost with diminishing returns in identifying the minimal sufficient size, whereas coarser granularities (e.g., 25% steps) risk missing critical transition points where hints become effective.

For each p, we compute the target token size:

$$n_p = \left\lfloor \tfrac{p}{100} |h_l(q)| \right\rfloor$$

and then re-query the LLM by setting the `max_new_tokens` parameter to n_p. This parameter acts as a hard constraint on the generation length, ensuring the model outputs a completion $h_l^{(n_p)}(q)$ that conforms to the pre-defined size n_p.

We treat this size-constrained completion as the candidate hint $\text{hint}(q, n_p) = h_l^{(n_p)}(q)$, and evaluate it by running the SLM on $q \oplus \text{hint}(q, n_p)$ and computing $\phi\big(q, h_s(q \oplus \text{hint}(q, n_p))\big)$.

We record the discretized minimal sufficient size

$$n^*(q) = \min \left\{ n_p : \phi\big(q, h_s(q \oplus \text{hint}(q, n_p))\big) \geq \tau \right\},$$

which is a grid-based approximation to the continuous optimum. The case $p = 0$ corresponds to $\text{hint}(q, 0) = \emptyset$, i.e., the router chooses not to query the LLM for any hint and directly returns the SLM output $h_s(q)$. If none of the partial sizes up to $p = 90$ meets the threshold, we set $n^*(q) = |h_l(q)|$ and use the full LLM response as the output.

Rationale for Design Choices

10% Granularity. The choice of 10% increments balances labeling efficiency with precision. Each training query requires $|\{0, 10, 20, \ldots, 90\}| = 10$ LLM calls to construct the supervision signal. Finer granularities would provide marginally better approximations to $n^*(q)$ but at high computational cost—5% steps would require 20 calls per query, doubling the labeling budget. Conversely, coarser granularities (e.g., 25% steps) risk missing critical transition points where the hint becomes sufficient, particularly for queries with sharp quality transitions.

Deterministic LLM Decoding. For all hint-labeling LLM calls, we use deterministic decoding (`temperature`$= 0$, `top_p`$= 1$) to ensure reproducible supervision signals and eliminate variance from sampling across different training runs. This design choice prioritizes consistency in the training labels over diversity in hint content.

Grid-based Approximation. The discretized target $n^*(q)$ serves as a practical surrogate for a continuous optimum. While this introduces quantization error, the 10% grid is sufficient to capture meaningful differences in hint requirements across queries, and the log-transformation of the regression target (Section 3) further smooths over small discretization artifacts.

D.3 Model Architecture Details

The numeric feature vector $\mathbf{f}(q)$ is passed through a lightweight MLP to produce $\mathbf{g}(q) = \text{MLP}(\mathbf{f}(q))$, which is concatenated with the `[CLS]` embedding $\mathbf{h}_{\text{CLS}}(q)$ to form the fused representation $\mathbf{u}(q) = [\mathbf{h}_{\text{CLS}}(q); \mathbf{g}(q)]$. $\text{Head}_{\text{hint}}$ and $\text{Head}_{\text{size}}$ are small feed-forward networks attached to $\mathbf{u}(q)$: $\text{Head}_{\text{hint}}$

maps $\mathbf{u}(q)$ to a single scalar logit for binary classification (hint-needed vs. no-hint), while $\mathrm{Head}_{\text{size}}$ maps $\mathbf{u}(q)$ to a single scalar for the regression target $r(q)$. To reduce variance in the hint-indicator logits, we apply multi-sample dropout and average logits across multiple passes Inoue (2019).

D.4 Optimization and Implementation Details

We train with AdamW Loshchilov & Hutter (2019) and maintain an exponential moving average (EMA) of parameters for evaluation and checkpoint selection Morales-Brotons et al. (2024).

Class Balancing. As noted in Section 3, we use weighted minibatch sampling to address class imbalance when the proportion of queries requiring hints deviates from 50%. Concretely, we employ `WeightedRandomSampler` during training to balance the minibatch distribution, ensuring that both positive and negative examples are adequately represented in each gradient update and yielding robust decision boundaries.

Policy Calibration. Post-training, we perform a grid search on the validation set to calibrate the policy for a target cost size. Specifically, given a fixed cost constraint C_{target}, we search over the probability threshold α for the binary classification decision to identify the configuration that maximizes validation accuracy while satisfying the cost constraint. This allows us to adapt the trained model to different cost-accuracy operating points without retraining.

D.5 Inference Routing Rationale

The hint-size threshold η_{hint} separates low-complexity from high-complexity queries among those predicted to need a hint. Queries with $0 < \pi_\theta(q) \leq \eta_{\text{hint}}$ are passed to the LLM with $\texttt{max_new_tokens} = \pi_\theta(q)$ to obtain $\text{hint}(q, \pi_\theta(q)) = h_l^{(\pi_\theta(q))}(q)$, and the shepherded response $h_s(q \oplus \text{hint}(q, \pi_\theta(q)))$ is returned. Queries with $\pi_\theta(q) > \eta_{\text{hint}}$ are deemed high-complexity and routed directly to the LLM, returning $h_l(q)$ without invoking the SLM.

This threshold-based routing captures the intuition that queries requiring extensive hints may be better served by the LLM's full reasoning capacity, avoiding the overhead of generating a lengthy hint only to invoke the SLM. The threshold η_{hint} is a hyperparameter tuned on the validation set to balance cost and accuracy.

D.6 Reactive Shepherding: Feature Engineering and Hyperparameter Choices

This appendix supplements Section 3.4 with the rationale behind the cascading hyperparameters and the precise definitions of the features fed into the shepherding model. Recall that for each query we run the SLM K times with temperature sampling ($T = 0.3$) to obtain candidate answers $\{A_1, \ldots, A_K\}$ and predictive entropies $\{e_1, \ldots, e_K\}$. We use stochastic sampling rather than deterministic decoding to obtain diversity in the responses, which lets us measure the SLM's epistemic uncertainty through both response variation and predictive entropy. Since we work with math and coding datasets, agreement is determined by exact match on final answers.

Choice of K and k. We set $K = 3$ in our experiments as a practical balance between signal quality and computational cost. The choice of k reflects a precision-cost trade-off: $k = 3$ (unanimous agreement) is more conservative, reducing false negatives at the expense of higher routing rates and LLM costs, while $k = 2$ (majority voting) is more aggressive, allowing more queries to bypass the LLM but risking more errors in direct SLM outputs. We explore both settings in our experiments (Section 4).

Numeric Features. Given the K responses, we compute the following features and incorporate them into the shepherding model's numeric input $\mathbf{f}(q)$:

- **Average entropy:** $\bar{e}(q) = \frac{1}{K}\sum_{i=1}^{K} e_i$, capturing the model's overall uncertainty;
- **Average output length:** $\bar{L}_{\text{out}}(q) = \frac{1}{K}\sum_{i=1}^{K} |A_i|$, reflecting the typical response complexity;
- **Query length:** $|q|$, measured in tokens via the backbone model's tokenizer.

We use token-level rather than character-level length features because tokenization aligns with the models' internal representations and provides a more semantically meaningful proxy for reasoning complexity; longer outputs typically indicate multi-step solutions requiring more intermediate reasoning. Token-based features also enable efficient batched processing without additional tokenization overhead at inference time. These features are standardized and fused with the text representation through a lightweight MLP.

E Outlier Analysis

E.1 Outlier Generation Strategy

To identify and filter outliers in our training datasets, we employ a consensus-based approach that leverages the agreement patterns across different hint sizes. For each sample, we generate SLM responses across 11 different hint-size configurations and compare them with the ground truth. At each hint size level, we classify the outcome as an outlier if the minority class (correct or incorrect) occurs, and as normal otherwise. We then aggregate these binary outlier flags to compute k, the total number of outliers per sample across all hint size settings.

Samples with high outlier counts (e.g., $k \geq$ threshold) exhibit inconsistent behavior across varying hint sizes and may introduce noise during training. We therefore filter out such samples to refine the dataset quality. Figures 6 and 7 illustrate the distribution of outlier counts for the CNK12 and GSM8K datasets, respectively.

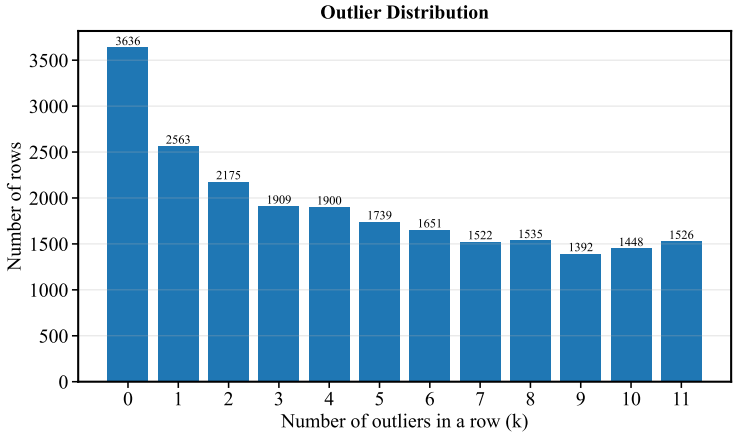


Figure 6: Outlier distribution for CNK12 dataset.

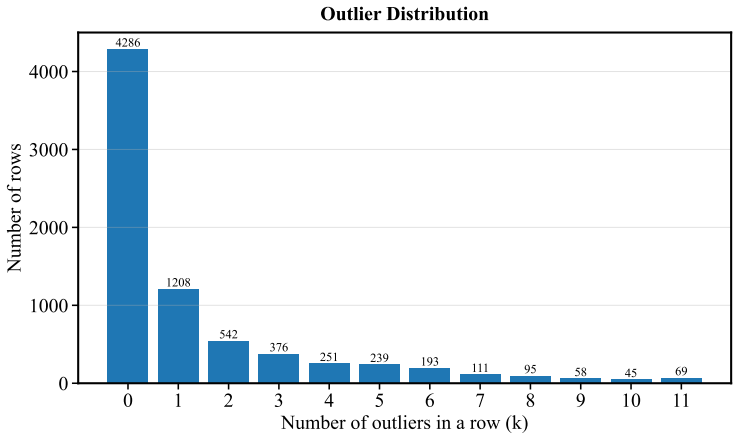


Figure 7: Outlier distribution for GSM8K dataset.

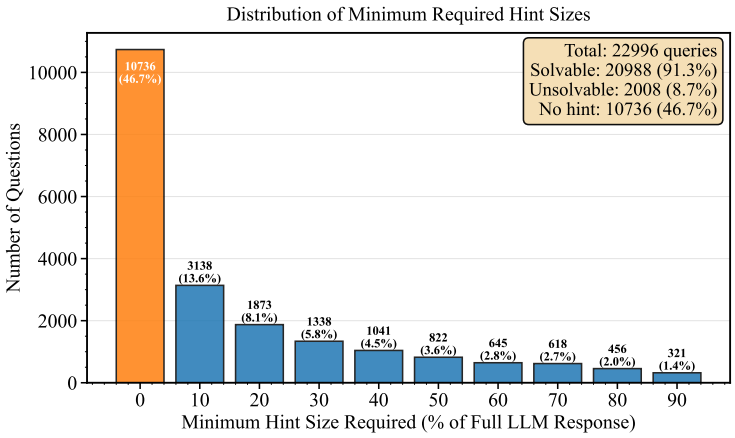


Figure 8: Distribution of minimum required hint sizes on CNK12 dataset (22,996 queries).

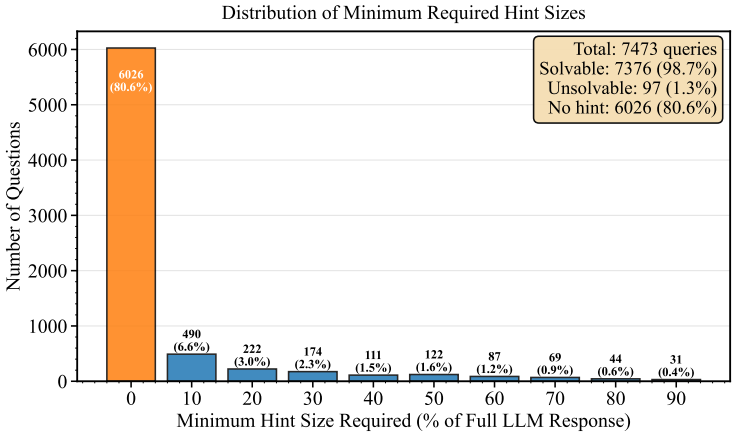


Figure 9: Distribution of minimum required hint sizes on GSM8K dataset (7,473 queries).

Note that we apply this outlier filtering strategy only to CNK12 and GSM8K, as these datasets contain sufficient training samples to justify statistical filtering. For HumanEval, which consists of only 165 problems designed specifically for evaluation purposes, we do not perform outlier analysis.

F Zero-Inflated Distribution of Hint Sizes

As mentioned in Section 3, the distribution of required hint sizes is zero-inflated: a large point mass at $n^*(q) = 0$ together with a conditional distribution over positive hint sizes that decays approximately exponentially. Figure 8 and Figure 9 illustrate this distribution across our training datasets CNK12 and GSM8K, respectively.

The distributions across both datasets demonstrate several key properties that motivate our two-stage prediction approach:

- **Dataset-dependent class imbalance:** The proportion of queries requiring hints varies significantly between datasets. GSM8K shows extreme imbalance with 80.6% (6,026 out of 7,473) requiring no hint, while CNK12 is more balanced with 46.7% (10,736 out of 22,996) requiring no hint. This dataset-dependent behavior necessitates a flexible approach that can handle varying levels of class imbalance.
- **Consistent exponential decay in the positive tail:** Despite different absolute proportions, both datasets exhibit similar exponential decay patterns in their conditional hint size distributions. In CNK12, the distribution decays from 13.6% (10% hints) to 1.4% (90% hints),

while in GSM8K it decays from 6.6% (10% hints) to 0.4% (90% hints). The consistent decay ratio (approximately 0.6–0.8$\times$ per step) confirms the exponential character of the conditional distribution.

- **High variance:** The standard deviation of hint sizes (computed only for queries requiring hints, i.e., $> 0\%$) is substantial in both datasets. Directly regressing on raw token counts would be highly sensitive to these large values, motivating our log-transformation strategy: $r(q) = \log(1 + \tilde{n}^*(q))$ to compress the target range.
- **Variable unsolvability rates:** The proportion of queries unsolvable even with 90% hints varies substantially: 8.7% (2,008) in CNK12 versus only 1.3% (97) in GSM8K. This indicates CNK12 contains more inherently challenging queries that exceed the SLM's capabilities even with substantial guidance. These queries are excluded from hint-size prediction because they require full LLM deployment.
- **Exponential decay characteristic:** Both datasets show clear exponential rather than linear or uniform decay. For example, in CNK12: 13.6% $\rightarrow$ 8.1% $\rightarrow$ 5.8% $\rightarrow$ 4.5%, and in GSM8K: 6.6% $\rightarrow$ 3.0% $\rightarrow$ 2.3% $\rightarrow$ 1.5%. This power-law-like behavior distinguishes the distribution from normal or uniform patterns and validates the need for specialized handling.

This zero-inflated distribution poses significant challenges for direct regression:

1. A single regression model would struggle to accurately predict both the numerous no-hint cases (which constitute 47-81% depending on the dataset) and the rare large-hint cases (which are critical for maintaining quality on difficult queries).
2. Standard ℓ_2 loss would be dominated by reconstruction error on the tail, potentially sacrificing accuracy on the more frequent short-hint cases.
3. The varying class balance between datasets (47-53 split in CNK12 vs 81-19 split in GSM8K) requires flexible handling via weighted sampling to ensure robust decision boundaries across different data distributions.
4. The presence of unsolvable queries (1.3-8.7% depending on dataset) necessitates explicit handling to avoid the model attempting to predict hints for inherently infeasible cases.

Our two-stage approach—first classifying whether any hint is needed, then predicting the size only for positive cases—naturally decomposes this distribution into two manageable sub-problems aligned with its zero-inflated structure: a balanced binary classification for the point mass at zero (via weighted sampling that adapts to dataset-specific imbalance) and a regression on the conditional distribution $p(n^* \mid n^* > 0)$, which excludes the dominant zero mass and focuses modeling capacity on the approximately exponential tail of positive hint sizes. This decomposition proves robust across datasets with varying difficulty profiles, as evidenced by the consistent performance on both the balanced CNK12 and the imbalanced GSM8K datasets.

G SLM Cost Calibration

To make the cost comparison between locally hosted SLM inference and API-based LLM inference consistent, we assign the SLM a nonzero token-level cost based on measured hardware throughput. We benchmark `Llama-3.2-3B-Instruct` in BF16 using vLLM with continuous batching on the same RTX 5090 used in our experiments. We replay 7,473 evaluation queries and measure an end-to-end throughput of 17.7k tokens/s, corresponding to approximately 63.8M tokens/hour. Using an RTX 5090 cloud rental rate of $0.37/hour (Vast.ai, May 2026), this yields an effective SLM cost of $0.0058 per 1M tokens.

Sensitivity to the SLM price. The $0.0058 per 1M token SLM price is an estimate, and it sets the bottom of the ACE ratio. To test how much the rankings depend on it, we priced the SLM at 1/100, 1/50, 1/20, and 1/10 of the LLM's input and output prices and recomputed both metrics on GSM8K, with everything else unchanged: ACE at the Table 2 operating points, and AIQ over the sweeps of Appendix J. The 1/100 point is close to the paper's estimate, since $0.0058 is roughly 1/100 of the LLM's input price. The 1/10 point is close to Groq's listed price for the stronger Llama-3.1-8B ($0.05 input and $0.08 output per 1M tokens), so the band runs from our hardware-based estimate to what renting through the same API would cost.

Table 9: AIQ (%) on GSM8K as the SLM price varies from 1/100 to 1/10 of the LLM's input and output prices.

SLM price ($\times$ LLM price)	Reactive Shepherding	ABC	FrugalGPT
1/100	91.5	87.9	80.5
1/50	91.5	87.6	80.4
1/20	91.3	86.7	80.1
1/10	91.1	85.0	79.6

Table 10: ACE on GSM8K at the Table 2 operating points (cost and accuracy unchanged) as the SLM price varies from 1/100 to 1/10 of the LLM's input and output prices.

Strategy	Cost ($)	Acc. (%)	1/100	1/50	1/20	1/10
FrugalGPT	0.039	79.6	0.73	0.74	0.77	0.84
ABC	0.051	94.3	1.76	1.79	1.83	1.92
Reactive Shep.	0.036	89.1	1.91	1.94	2.06	2.23

The ordering between Reactive Shepherding and the cascading baselines does not change under either metric anywhere in this tenfold range. Most of the cost at these operating points comes from LLM output tokens, and all three methods pay the same SLM price, so raising it shifts them together instead of reordering them.

H Prompting and Evaluation Details

This section gives the prompt templates used for hint generation, SLM inference, and shepherding supervision, followed by the inference and evaluation protocol of Section 4.

H.1 Prompt Template

Across all datasets, the small or large model receives the query with the same instructions and, when available, a generated large-model hint as additional contextual input. The general prompt structure is:

> **Instruction:** Produce a final answer following the task-specific constraints.
> **Question:** {query}
> **Hint:** {generated large-model output}

When no hint is provided, the prompt is identical except that the `Hint:` line is omitted.

GSM8K: We use a fixed few-shot prompt consisting of two example question–answer pairs, followed by the target query. When available, a generated large-model hint is appended as additional context.

MBPP: The small and large models receive the function specifications, function signature, and provided test cases, along with the generated large-model hint if available. The model is instructed to generate only executable Python code, without additional explanation. The prompt structure otherwise follows the general template above.

HumanEval: Follows the same prompting procedure as MBPP; however, the test cases and function signature are already included in the dataset's original prompt.

CNK12: Follows the same prompt template as GSM8K but uses single-shot prompting without example question–answer pairs.

H.2 Inference and Evaluation Protocol

Test sets contain no pre-computed hints. At inference time the policy predicts a hint size $\pi_\theta(q)$. When $\pi_\theta(q) > 0$, we query the LLM with `max_new_tokens` $= \pi_\theta(q)$ and append the returned prefix to

the original prompt for SLM completion (Section 3). All test-time SLM and LLM generations use temperature 0.3 and top-p 0.95.

For GSM8K and CNK12, we extract the final numerical answer from the response and compare it with the ground truth by exact match. For HumanEval and MBPP, we execute the generated program against the benchmark's unit tests and count a response as correct only when all tests pass. To reduce sampling variance, we repeat each query in 7 independent trials and report the majority-vote outcome.

I Method Configurations for Minimum Accuracy Experiments

Table 11 details the hyperparameter settings used for each method across all datasets in the minimum accuracy requirement experiments (Section 4.3). Each baseline is tuned to achieve accuracy closest to the 90% LLM accuracy threshold on each dataset. For RouteLLM, the values shown in the table are for the routing threshold. For FrugalGPT, the values represent the confidence threshold below which queries are escalated to the LLM. GraphRouter uses either its "Performance First" (P.F.) mode, which prioritizes accuracy, or an additional bias term added to increase cost orientation. ABC uses its default configuration with $N = 2$, checking agreement between two SLM responses before escalating to the LLM. For Reactive Shepherding, η_{hint} denotes the hint token budget; K denotes the number of SLM responses used in consensus evaluation, and α denotes the probability threshold for the binary hint classification decision.

Table 11: Method configurations for minimum accuracy experiments. P.F. = Performance First mode; Bias = additive term to increase cost orientation.

Method	**GSM8K**	**CNK12**	**HumanEval**	**MBPP**
RouteLLM	0.48	0.57	0.70	0.58
GraphRouter	P.F.	P.F.	P.F. + 4×10^{-7} Bias	P.F.
FrugalGPT	0.0006	0.001	0.001	0.00016
ABC	$N = 2$	$N = 2$	$N = 2$	$N = 2$
	$\eta_{\text{hint}} = 58$	$\eta_{\text{hint}} = 60$	$\eta_{\text{hint}} = 110$	$\eta_{\text{hint}} = 130$
Reactive Shepherding	$K = 3, \alpha = 0.228$	$K = 2, \alpha = 0.349$	$K = 2, \alpha = 0.228$	$K = 2, \alpha = 0.372$

J Pareto Sweep Details

Figure 3 in Section 4.2 plots the cost–accuracy curves of every method on all four benchmarks. This appendix details the sweep configurations behind those curves and analyzes the cascading methods that exceed the LLM-only baseline.

J.1 Sweep Configurations

For each baseline and for Proactive and Reactive Shepherding, we vary the primary cost–accuracy trade-off parameter over a fine grid on the test set of each benchmark and record the resulting total cost and accuracy:

- **RouteLLM** (Ong et al., 2025): routing threshold swept over $[0, 1]$.
- **GraphRouter** (Feng et al., 2025): cost-bias term swept, with Performance-First mode as the high-accuracy endpoint.
- **FrugalGPT** (Chen et al., 2024): cascading confidence threshold swept over its full operating range.
- **ABC** (Kolawole et al., 2025): ensemble size N and agreement requirement k swept.
- **Proactive and Reactive Shepherding**: probability threshold α and hint budget η_{hint} swept jointly; Reactive additionally varies the consensus requirement k over $\{2, 3\}$.

J.2 Cascading Methods Exceeding the LLM-Only Baseline

On HumanEval and MBPP, both Reactive Shepherding and ABC attain accuracy above the LLM-only baseline. This is initially counterintuitive, since the SLM is individually weaker than the LLM on both benchmarks. The behavior arises from the cascading mechanism. When the SLM achieves majority agreement across K samples (with $K = 2$ on both benchmarks in Table 11), its consensus output is returned directly without invoking the LLM. On code generation tasks, this consensus output is occasionally correct on queries where the LLM's single-pass output fails to compile or contains subtle bugs. The system therefore combines two complementary sources of correct answers: SLM consensus on queries where agreement signals reliability, and LLM hints on harder queries. ABC, which uses SLM agreement as its gating mechanism, exhibits the same phenomenon for the same reason. The effect is specific to code generation, where stochastic single-shot LLM outputs can fail while stable SLM consensus succeeds; on GSM8K and CNK12, the LLM's single-pass accuracy is high enough that no method exceeds it.

K More Related Works

Existing techniques for efficient, accurate, or low-cost SLM-LLM collaborative inference include: 1) LLM routing, 2) LLM cascading, 3) speculative decoding, and 4) knowledge distillation.

K.1 LLM Routing

LLM routing addresses cost-quality trade-offs by using a learned router to decide *a priori* which model should handle a given query. Early work by **Lu et al.** Lu et al. (2024) introduced **Zooter**, a reward-guided routing method that distills rewards from training queries to learn a routing function, achieving top performance on 44% of tasks across 26 benchmark subsets while maintaining computational efficiency. Building on this foundation, **Hybrid-LLM** Ding et al. (2024) proposed quality-aware query routing that dynamically trades quality for cost based on predicted query difficulty, achieving up to 40% fewer calls to large models with no quality degradation. **RouterBench** Hu et al. (2024) subsequently established a standardized evaluation framework comprising over 405k inference outcomes from representative LLMs, enabling systematic comparison of routing strategies. **RouteLLM** Ong et al. (2025) advanced the field by introducing a principled framework for learning routers from human preference data, demonstrating cost reductions of over 85% on MT-Bench and 35% on GSM8K while maintaining 95% of GPT-4's performance. Most recently, **GraphRouter** Feng et al. (2025) proposed an inductive graph framework that models contextual interactions among tasks, queries, and LLMs as a heterogeneous graph, achieving at least 12.3% improvement over prior routers and enhanced generalization to unseen LLMs.

Despite these advances, routing exhibits a fundamental structural limitation: the decision is inherently binary (SLM or LLM), and when the LLM is selected, the system incurs the full inference cost. Routing does not facilitate partial reuse of LLM knowledge, rendering it insufficiently flexible for strict cost constraints.

K.2 LLM Cascading

LLM cascading arranges models hierarchically, escalating queries to more capable models when weaker models fail quality criteria. **FrugalGPT** Chen et al. (2024) pioneered this approach, demonstrating up to 98% cost reduction by learning which LLM combinations to use per query. **LLM-Blender** Jiang et al. (2023) introduced an ensemble perspective, using pairwise ranking to select among candidate outputs and generative fusion to merge top candidates. **AutoMix** Aggarwal et al. (2024) advanced cascading through few-shot self-verification and POMDP-based routing, reducing costs by over 50%. Recent work has explored both learned and training-free approaches: **C3PO** Chen et al. (2025) provides label-free cascading with formal cost guarantees, while **Agreement-Based Cascading (ABC)** Kolawole et al. (2025) proposes a simple, training-free strategy where escalation decisions are based on agreement among an ensemble of model responses. In our single-SLM setting, we adapt ABC by measuring consensus across multiple responses from the same SLM.

While cascading reduces average cost, it remains ill-suited for strict budget constraints: SLMs frequently fail on complex tasks, triggering full LLM inference. Each escalation requires a complete LLM forward pass, limiting cost savings when SLM reliability is low.

K.3 Speculative Decoding

Speculative decoding Leviathan et al. (2023); Chen et al. (2023) is a widely adopted technique for accelerating LLM inference without sacrificing output quality. The approach uses a smaller *draft model* to generate candidate token sequences, which are then verified in parallel by the larger *target model*. Tokens that match the target model's distribution are accepted, amortizing the cost of large-model inference across multiple tokens. **Medusa** Cai et al. (2024) extends this idea by adding multiple decoding heads to the LLM itself, eliminating the need for a separate draft model. **Draft & Verify** Zhang et al. (2024) achieves self-speculative decoding by selectively skipping intermediate layers during drafting, providing up to $2\times$ speedup without additional training. **Online Speculative Decoding** Liu et al. (2024) continuously adapts the draft model during inference to improve token acceptance rates.

Speculative decoding and shepherding solve fundamentally different problems. Speculative decoding accelerates LLM inference *latency* by parallelizing token verification, but the LLM remains in the critical path for every token—the total LLM computation is unchanged. Shepherding reduces LLM *monetary cost* by minimizing the number of LLM tokens generated. The two are complementary: shepherding could use speculative decoding to accelerate the hint-generation step itself. A direct empirical comparison would require equating their different objectives (latency vs. cost), which we leave as future work.

K.4 Knowledge Distillation

Knowledge distillation (KD) Hinton et al. (2015) transfers capabilities from a large teacher model to a smaller student model during training. Recent work has extended KD to LLMs: **MiniLLM** Gu et al. (2024) proposes reverse KL divergence as the distillation objective, which better suits generative language models by preventing the student from overestimating low-probability regions. **Distilling Step-by-Step** Hsieh et al. (2023) extracts chain-of-thought rationales from LLMs to train smaller models more data-efficiently, enabling a 770M-parameter model to outperform few-shot prompted 540B PaLM. Comprehensive surveys Gou et al. (2021); Xu et al. (2024) provide extensive coverage of KD techniques for LLMs, including skill-specific and domain-specific distillation.

Knowledge distillation and shepherding are complementary but distinct. KD is a *training-time* technique: once distillation is complete, the student model operates independently at inference. Shepherding, in contrast, is an *inference-time* collaboration that dynamically injects LLM hints based on each query's difficulty. KD improves the SLM's baseline capability, while shepherding provides targeted assistance when that baseline proves insufficient.

K.5 Positioning of LLM Shepherding

Our work differs fundamentally from all the above approaches. Unlike routing and cascading, which make binary decisions about *which* model generates the complete response, shepherding controls *how much* LLM output to request, enabling a continuous trade-off between cost and quality. Unlike speculative decoding, which accelerates LLM inference while keeping the LLM in the critical path for every token, shepherding removes the LLM from final response generation entirely. Unlike knowledge distillation, which operates during training, shepherding provides dynamic, query-specific guidance at inference time.

Shepherding complements existing approaches: proactive shepherding enhances routing by replacing full LLM calls with partial hints, reactive shepherding enhances cascading by substituting full escalations with targeted guidance, and shepherding can be combined with distilled SLMs for further improvements. To our knowledge, this is the first work to exploit token-level budget control for SLM-LLM collaboration.