A Hybrid SLM and LLM System for Cost-Effective Optimized Inference

Tanay Matta[0000−1111−2222−3333], Kabita Choudhary[0009−0004−1235−3273], and Sujala D. Shetty[0000−0003−3992−1109]

Department of Computer Science, Birla Institute of Technology and Science, Pilani, Dubai Campus, Dubai International Academic City, Dubai, United Arab Emirates
f20220274@dubai.bits-pilani.ac.in, p20230907@dubai.bits-pilani.ac.in, sujala@dubai.bits-pilani.ac.in

Abstract. Large language models (LLMs) achieve strong performance on complex natural language tasks but require substantial computational resources and incur high operational costs. Small language models (SLMs) offer faster inference and lower costs but tend to lack the reasoning capabilities needed for complex tasks. This creates a gap between efficiency and performance, limiting the practical deployment of language models in resource-constrained environments and in costsensitive applications. This work introduces HybridLM, a hybrid system that intelligently combines SLMs and LLMs to achieve an optimal balance between performance, cost, and response time. The strategy adopts a machine learning-based routing mechanism, enhanced with reinforcement learning, that automatically selects the appropriate model based on a complete review of query complexity. The system routes simple tasks such as basic question answering, factual queries, and clear-cut information retrieval to locally deployed SLMs, while directing intricate reasoning, analytical tasks, and specialized queries to cloud-based LLMs. Building on the HybridLM system, the study concentrates on three main objectives: developing a multi-dimensional feature identification and classification system that accurately assesses query complexity in real-time; implementing series-parallel SLM orchestration that retains response quality through consensus validation and progressive improvement; and establishing an online educational model that continuously improves routing choices through accumulated production experience and reward-based feedback. Through the proposed approach, the paper examines how intelligent caching combined with adaptive routing strategies influences overall system performance, explores trade-offs between effectiveness and performance through experimental evaluation using established evaluation metrics, and demonstrates real-world implementation approaches for integrated systems in production environments. Collectively, the present work addresses an urgent need in the AI community for efficient yet capable language processing systems, enabling broader adoption of AI technologies while significantly reducing operational expenditures.

Keywords: Hybrid Language Models · Adaptive Model Routing · Query Complexity Classification · Reinforcement Learning · Cost-Efficient AI Inference

1 Introduction

Large Language Models (LLMs) such as GPT-4, Claude, and Gemini have revolutionalized natural language processing through their ability to understand and generate human-like text across diverse tasks, but their computational demands result in high inference costs along with increased latency due to network bottlenecks. In contrast, Small Language Models (SLMs) such as Phi, Mistral, and Granite offer smaller alternatives with smaller sizes thus providing faster inference times and lower computational costs. Although they often struggle with complex reasoning and multi-step problem solving. Current AI applications face a primary trade-off between performance and efficiency, forcing developers to choose between deploying expensive LLMs for all queries or using SLMs exclusively and sacrificing quality on complex requests. [1]

This paper presents HybridLM, an intelligent routing system that dynamically selects between large and small language models based on query complexity analysis, aiming to reduce overall inference costs all while maintaining comparable quality metrics using a self-made routing algorithm that uses Machine Learning and Reinforcement learning. This paper details the architecture and evaluation of HybridLM, covering existing literature on model routing and query complexity classification (Section 2), system architecture (Section 3), routing strategy (Section 4), experimentation and results (Section 5), and conclusion (Section 6).

2 Literature Review

2.1 Small Language Models and Efficiency

Belcak et al. [2] argue that small language models represent the future of agentic AI systems, demonstrating that models with fewer than 10 billion parameters can achieve comparable performance to larger counterparts on specific tasks when properly trained and fine-tuned. The analysis reveals that SLMs offer 3–5× faster inference speeds and 70–80% lower computational costs compared to large models, making them particularly suitable for edge deployment and high-throughput applications where latency and resource constraints are critical. Zhang et al. [8] provide a comprehensive survey of small language models in the era of large language models, categorizing techniques for enhancing SLM capabilities, including knowledge distillation, efficient fine-tuning, and architectural optimizations. Gunasekar et al. [7] introduced the "Textbooks Are All You Need" paradigm, demonstrating that small models trained on high-quality, curated datasets can achieve remarkable performance despite limited parameter counts. Their phi-1 model (1.3B parameters) matches or exceeds GPT-3.5 performance on Python coding tasks while requiring 100× fewer training tokens, achieving pass@1 scores above 50% on HumanEval benchmarks—only 5% below GPT-4—thereby validating data quality over quantity for specialized domains. Building upon this approach, Abdin et al. [1] present Phi-3, a family of models (3.8B to 14B parameters) designed for on-device deployment. Phi-3-mini (3.8B)

achieves performance comparable to Mixtral-8×7B and GPT-3.5 on common benchmarks including MMLU (69.4%accuracy vs. 70.2%) and MT-bench (8.38 vs. 8.52 scores) while fitting within mobile device memory constraints (14 GB) and delivering sub-100ms inference latencies on commodity hardware. Chowdhery et al. [4] introduced PaLM (Pathways Language Model), scaling to 540 billion parameters and demonstrating state-of-the-art performance across diverse tasks including mathematical reasoning, code generation, and multi-step problem solving. The paper establishes performance baselines that highlight the capabilities gap between large and small models. However, PaLM's deployment requires specialized hardware (TPU v4 pods) with inference costs exceeding $0.01 per 1K tokens—orders of magnitude higher than efficient SLMs.

2.2 Cost Optimization and Hybrid Architectures

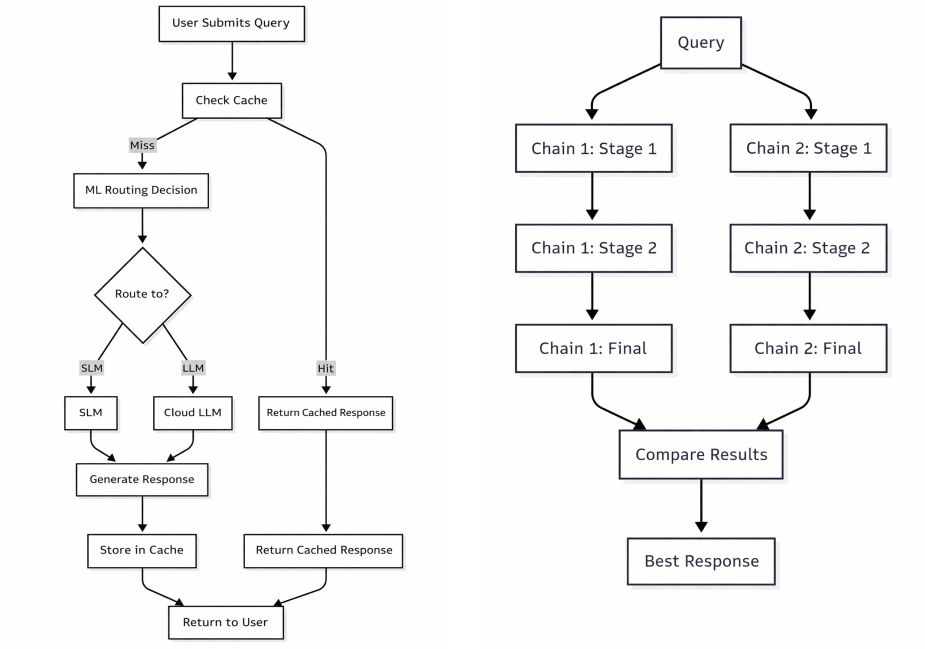
Chen et al.[3] propose FrugalGPT, a cascade-based system that dynamically routes queries across multiple language models to optimize cost-performance trade-offs. Their approach employs a learned scoring function that predicts query complexity from linguistic features including sentence length, syntactic depth, named entity density, and semantic ambiguity.

Experimental results demonstrate 50–98% cost reduction compared to GPT-4-only deployment while maintaining 99% of output quality on standard benchmarks. FrugalGPT achieves this through three strategies: prompt adaptation (reducing token counts), LLM approximation (caching similar queries), and model selection (routing to appropriate model tiers). On the HELM benchmark suite, FrugalGPT reduces costs by 87% for classification tasks, 63% for summarization, and 54% for open-ended question answering while incurring quality degradation below 2% across categories. Ding et al. [6] present Hybrid LLM, which frames cost-efficient deployment as a qualityaware routing problem between small and large language models. The system employs a learned router that predicts query difficulty and expected quality gaps, dynamically trading off cost and quality at test time by directing straightforward queries to cheaper small models while reserving large models for complex reasoning tasks. Experimental evaluation demonstrates up to 40% reduction in large-model API calls with negligible quality degradation (less than 1% drop) compared to exclusive large-model deployment. The router uses features including query length, syntactic complexity, semantic ambiguity, and domain-specific indicators to make routing decisions, continuously adapting through reinforcement learning from quality feedback.

3 System Architecture and Performance Optimization

3.1 System Overview

The HybridLM architecture follows a layered approach where each layer serves a specific purpose in the query processing pipeline. When a user submits a query, it flows through multiple stages before reaching the user as a response. This routing



(a) Cache-aware query routing framework.

(b) Dual-chain response comparison framework.

Fig. 1: High-level architectures used in the proposed query processing and response selection pipeline.

decision considers query complexity, required reasoning depth, and whether the query contains specific content like code or mathematical notations. Queries routed to the SLM enters the SLM orchestration layer, which manages how the SLM's process requests so as to maintain accuracy. Queries routed to cloud LLMs are sent to the external API's of the respective model provider. Regardless of which path is taken, the response is stored in the cache for potential future reuse before being returned to the user. fig. 1a

Hybrid Series-Parallel SLM Configuration The hybrid configuration combines the strengths of both the parallel and series chaining methods to try to achieve good accuracy thereby allowing for better results from small language models. In this setup, multiple SLM chains run in parallel, and each chain performs a step-by-step refinement of the response given to it by the previous model. Every chain begins with generating an initial answer, then refines and validates it through several stages i.e models. Once each chain produces its final output, the system compares all results across the parallel chains to find the best response by comparing the cosine similarity of the query embedding and the response

embedding which allows the system to benefit from both the error detection of parallel processing and the stepwise reasoning of series refinement.

3.2 Response Caching

Caching is used to prevent redundant computation. When a user submits a query, the system first normalizes the query and then generates a unique key for it. Normalization removes minor differences in formatting, spacing, and capitalization to ensure that queries are treated identically. The system then checks Redis for any cached response associated with this key.

If a cache hit occurs then the system immediately returns the stored response to the user without calling the router to make its choice. This cached response includes not only the text but also metadata about the original routing decision, model used, and generation timestamp. The entire operation reduces the inference time and saves money by preventing inference costs.The system has a 24-hour expiration window for queries after which the cache entry is automatically removed, ensuring that responses to time-sensitive queries are catered to as well. Beyond exact query matching, HybridLM implements semantic similarity caching for improved hit rates. When a cache miss occurs, the system computes the corresponding embedding vectors for the incoming query and compares them against recently processed queries which is stored in a vector index. If a semantically similar query is found with high confidence (cosine similarity above 0.95), the system may reuse the cached response.

4 Query Routing Strategy

The routing approach uses supervised machine learning combined with reinforcement learning to achieve an dynamic routing service that also learns from user interactions and continuously improves performance over time.

4.1 Query Feature Extraction

The routing decision process begins with feature extraction from each incoming query to construct a multi-dimensional vector of the incoming query. This feature vector serves as input to a machine learning classifier (logistic regressor). The system then analyses the basic structure of each query by observing three metrics that have been extracted: the number of tokens (individual words or punctuation marks), the total character count, and the actual word count. These metrics help serve as the initial indicators of query length and structural complexity. Quite a lot of the time longer queries with more tokens often indicate a complex request has been sent even though this relationship is not absolute. For example, a query like ”What is Python?” has 3 tokens and 15 characters is considered simple as it does not require reasoning or thinking before acting whereas a complex query such as ”Explain the differences between supervised and unsupervised learning

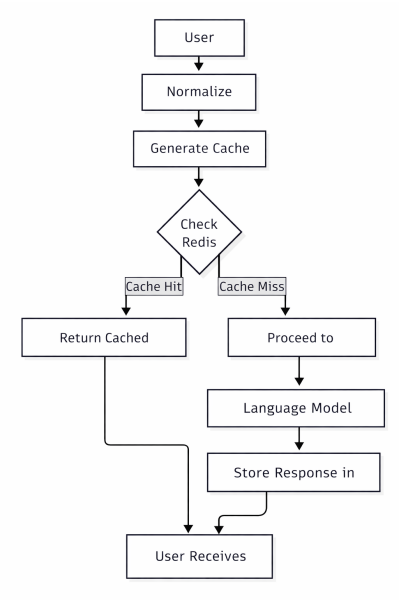


Fig. 2: Cache-based query processing workflow showing query normalization, cache key generation, Redis cache lookup, cached response return, and fallback to the language model when a cache miss occurs.

algorithms like a 5 year old" which requires the LLM to think about how to present the content contains 10 tokens and 76 characters.

The presence of words that indicate a relationship between the word and thinking before doing like "Explain", "Debug" and so on, indicates that the query requires deeper reasoning capabilities beyond simple fact retrieval. Each detected keyword contributes to an aggregate score. Queries containing multiple such keywords receive higher complexity ratings as they get concatenated, increasing the likelihood of an LLM routing decision.

4.2 Machine Learning Classification Model

The core routing decision mechanism uses a Logistic Regression classifier which has been chosen for its computational efficiency and and effectiveness in binary classification tasks as this is a binary problem, either LLM or SLM. The model learns the decision boundaries from the routing data that is received when the more and more queries are processed. The classifier receives the computed feature vector of the query which contains all extracted features as input. The model then learns the optimal weight for each feature during training. It then combines these weighted features to compute a probability score indicating the likelihood that a query requires LLM capabilities. The computed probability ranges from 0 (definitely suitable for SLM) to 1(definitely requires LLM). The model then

applies the learned weights to each feature, sums the weighted contributions, and converts the result to a probability using a sigmoid function which is then used to route the query to the respective model.

Decision Criteria and Confidence Metrics The routing decision uses a threshold approach: 0.5-0.5 split i.e queries with probability scores that are 0.5 or higher are routed to the LLM, while those below 0.5 are sent to the SLM. This threshold ensures that queries are routed to the more capable model only when there is reasonable certainty of the query being complex or not. A confidence score is also calculated for each routing decision such as for LLM-bound queries, the confidence equals the computed probability (e.g., 0.72 indicates 72% confidence) and for SLM-bound queries, the confidence is calculated as one minus the probability (e.g., a probability of 0.3 yields 70% confidence in SLM routing). Higher confidence scores indicate greater certainty in the routing decision.

4.3 Reinforcement Learning using Epsilon Greedy

To support the classifier, the system uses reinforcement learning principles that allow for continuous learning and performance improvement through user interaction and feedback loops. The RL system helps discover new routing strategies that can improve performance as with more inputs and feedback the classifier can be retrained to perform better. For the RL strategy, epsilon greedy is used that addresses this need by introducing controlled randomness into the routing process. Specifically, the system follows the classifier's prediction for 90% of queries, ensuring stable and consistent performance based on learned knowledge. However, for the remaining 10% of queries, the system randomly chooses between LLM and SLM routing regardless of the classifier's recommendation. This exploration rate is denoted as epsilon = 0.1. This exploration serves multiple purposes such as:

1. It allows the system to discover cases where the classifier may be consistently wrong.
2. It generates diverse training data, preventing the model from becoming too rigid in its predictions.
3. It adapts to changing query distributions over time by continuously testing alternative routing strategies.

Reward Function Design After each query is processed and a response is generated, the system then calculates whether the routing decision was correct and optimal. This evaluation is performed through by a reward function that assesses three key dimensions of the query i.e response quality, latency, and cost

Quality Assessment via LLM-as-Judge Response quality is scored by using a LLM-as-judge, whereby another SLM analyzes each response across the four

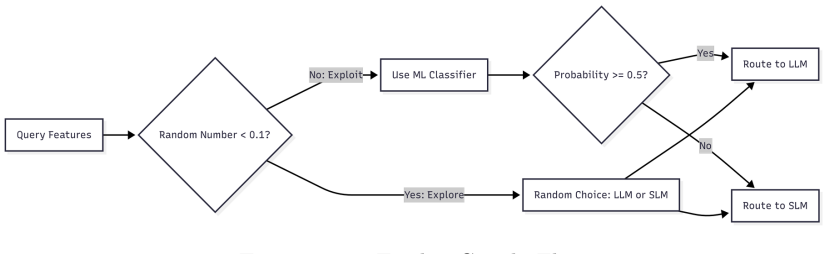


Fig. 3: Cache-based query processing workflow showing query normalization, cache key generation, cache lookup in Redis, return of the cached response, and fallback to the language model on a cache miss.

dimensions. The evaluation considers whether the response for the query is well-structured, factually accurate, addresses the user's query, and if it offers sufficient detail and completeness. Each dimension receives a score between 0 and 1, and these four scores are then averaged to calculate an overall quality score. It also measures response latency(ms). Faster responses receive higher scores and vice-versa that also has a normalization factor that penalizes the responses that take longer than 5 seconds. For example, a response that is delivered in 2.3 seconds may receive a lower score, while a faster 1-second response would score higher in comparison.

The cost scoring also differ based on the model used. SLM responses, being locally hosted and free, will always receive a perfect cost score of 1.0 whereas an LLM comes with usages and are scored based on the actual cost incurred, with lower costs receiving higher scores. The final reward combines these three components with weights that reflect system priorities. Quality receives the highest weight at 70%, ensuring that response accuracy and usefulness remain the primary objective. Latency and cost each receive 15% weight. This weighted combination produces a single reward score between 0 and 1 that summarizes the overall success of the routing decision. Higher rewards indicate optimal routing while the lower rewards signal that there were better decisions to be made. These reward scores help the continuous learning process, and ensuring future model updates.

4.4 Online Learning Framework

The application also incorporates an online ML system wherein the logistic regressor undergoes a periodic retraining as and when additional data is available from production usage. This then makes sure that the system adapts to more query patterns and also improves its decision-making over time and more usage. To allow this to function, a background monitoring process is used that checks for new training data every 5 minutes. During checks it counts how many new routing decisions have been made since the last model training session and when

the number of new decisions is 100 or more, the system automatically triggers a retraining cycle. The retraining process is not only incremental.

Instead of just training on the new samples, the system retrains using all the data that is stored in a database, which includes all previous decisions plus the newly accumulated data. This approach ensures that the updated model benefits from all the data and maintains consistency which solves the problem of catastrophic forgetting. Once retraining completes successfully and the new model passes validation checks, it is then automatically deployed to production. The entire process occurs in the background.

Bootstrap Initialization One of the biggest issues is that any ML model starts of untrained. To mitigate this issue, the system includes a curated bootstrap dataset comprising of 147 pre-labeled synthetic query-routing pairs. This dataset spans a various distribution of query types:

- **Simple queries (SLM-appropriate):** factual questions, basic definitions, and short greetings.
- **Complex queries (LLM-required):** analytical tasks, code debugging, comparative analysis, and reasoning-intensive problems.

The bootstrap dataset provides initial training data that allows the model to train before the system accumulates production data, so that the classifier can make good decisions right from the start.

4.5 System Architecture and Data Flow

The complete pipeline flows through several stages as introduced before that are, 1.feature extraction, 2.classification, 3.exploration, 4.response generation, 5.reward calculation, and 6.continuous learning into a streamlined workflow. When a user sends in a query, the system extracts all relevant features from the query and context. These features are then passed to the RL module which then determines whether to follow the ML classifier's prediction (90% probability) or explore randomly (10% probability) by using the epsilon greedy method.

If exploiting, the ML classifier receives a feature vector and computes the probability score. Queries scoring 0.5 or higher are routed to the LLM, while those below 0.5 are directed to the SLM as mentioned earlier. If exploring, the system randomly selects either LLM or SLM regardless of the features.

Once the routing decision is made, the selected model then generates a response to the user's query, and then the system records the response time in a database and also calculates the API cost based on token usage to get an estimate of how much API has been consumed. After the response is generated, it undergoes quality evaluation, and a score is generated, which then decides whether the route chosen was the correct one or not. It then stores all the routing information in the database that includes the extracted features, routing choice, confidence score, latency, cost, and calculated reward. This data becomes part of the training data for future model updates. When 100 new samples have been collected,

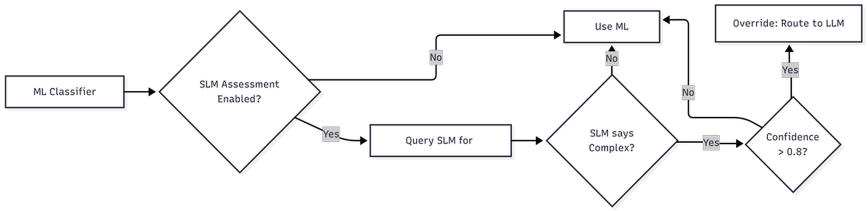


Fig. 4: Fallback and override routing mechanism.

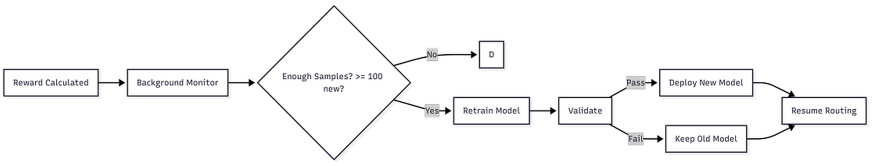


Fig. 5: Online model update workflow.

the system triggers automatic retraining on the complete dataset. Upon successful validation, the new model is deployed and begins handling routing decisions.

Safety Mechanisms and Fallback Strategies The routing system also implements multiple safeguards to ensure reliable operation and try to prevent any routing errors that could reduce the user experience. An optional validation layer has an SLM-based complexity assessor that provides an independent evaluation of query complexity. Before the final routing decision is executed, the system can query the local SLM to assess whether it believes that the query is easy to solve or requires the need for LLM inference. This SLM operates independently of the ML classifier, as it analyzes the query and returns both a complexity classification (i.e. simple or tough) and a confidence score. When the SLM strongly indicates that a query is complex with confidence exceeding 0.8—the system overrides the primary classifier's decision and routes to the LLM regardless of the original prediction. This safeguard mechanism addresses a critical failure mode: false negatives where the primary classifier incorrectly routes complex queries to the SLM. In such cases, the SLM can trigger an override to the more capable model. However, if the SLM's confidence is below 0.8 or it assesses the query as simple, the original ML classifier decision stands.

Fig. 6: Reward calculation pipeline.

5 Experimentation and Results

The evaluation focuses on four metrics i.e routing accuracy, which measures the correctness of model selection decisions, computational cost, measured in API expenses for cloud LLM queries, response latency, representing the time from query submission to response delivery, and F1-score, which captures the balanced precision-recall tradeoff in routing decisions.

5.1 Experimental Setup

Experiment 1: MMLU-Lite Benchmark Evaluation The first experiment utilises the MMLU-Lite benchmark, a subset of the Massive Multitask Language Understanding dataset designed to evaluate language model capabilities across diverse knowledge domains. This experiment employs 400 questions spanning multiple subject areas, including STEM fields, humanities, social sciences, and professional domains. The MMLU-Lite benchmark provides a standardised evaluation framework that enables direct comparison between HybridLM and single-model baselines. Each question in the dataset represents a distinct query that the system must process and route appropriately. The benchmark measures both the accuracy of the final responses and the efficiency of the routing decisions that led to those responses.

Experiment 2: Custom Labeled Query Dataset The second experiment employs a custom dataset of 100 labeled queries. This dataset follows a 60-40 distribution where 60% of queries are classified as simple and 40% are classified as hard that might require reasoning. This controlled distribution enables targeted evaluation of routing accuracy across different query complexity levels. Simple queries include factual questions, basic definitions, and straightforward information retrieval tasks. Hard queries involve analytical reasoning, code debugging, comparative analysis, and multi-step problem solving that typically exceeds SLM capabilities.

ML Optimising Function The classifier is trained using the binary cross-entropy loss function, which is defined as

$$L = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(\hat{y}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-\hat{y}^{(i)}\right)\right] \tag{1}$$

where m is the total number of training samples, $y^{(i)}$ is the true class label for sample i, and $\hat{y}^{(i)}$ is the corresponding predicted probability. In this routing framework, $y^{(i)} = 1$ indicates that the query should be routed to the LLM, while $y^{(i)} = 0$ indicates routing to the SLM. Model parameters are updated through gradient descent using a learning rate of $\alpha = 0.01$.

Table 1: Global-MMLU-Lite benchmark results for 400 questions.

Model/System	Accuracy	Total Cost	Avg Time
GPT-4.1 (LLM-Only)[5]	87.5%	$0.17	1.03 s
Granite 4.0 H-Small (SLM-Only)[5]	75.03%	$0.00	0.52 s
HybridLM (ML+RL)	**86.25%**	**$0.11**	**0.96 s**

5.2 Experiment 1: Global-MMLU-Lite Benchmark Results

The Global-MMLU-Lite evaluation with 400 questions reveals significant performance differences across model configurations. Table 1 compares HybridLM against its two constituent single-model baselines: GPT-4.1, the cloud LLM used for complex query routing, and Granite 4.0 H-Small, representative of the local SLM tier. Accuracy figures for the single-model baselines are sourced from the Cohere Labs Global-MMLU-Lite leaderboard [5]. Costs for GPT-4.1 are derived from measured per-query token usage in our evaluation logs at published API rates ($2.00/$8.00 per million input/output tokens); Granite 4.0 H-Small is run via local inference at negligible cost.

HybridLM achieves 86.25% accuracy on Global-MMLU-Lite, within 1.25 percentage points of the GPT-4.1 LLM-only baseline while operating at a lower total cost of $0.11 versus $0.17. Of the 400 questions, 126 (31.5%) were handled locally by the SLM and 274 (68.5%) were escalated to GPT-4.1. Crucially, the router selectively assigns to the SLM only those queries whose complexity falls within its capability envelope — factual recall, definitions, and straightforward reasoning — meaning Granite 4.0 H-Small operates on a curated subset of queries on which it performs substantially above its global benchmark average of 75.03%. This selective routing is the mechanism by which HybridLM achieves accuracy well above what a naive weighted combination of the two baselines would predict, and directly demonstrates that the routing strategy is working as intended. The SLM-only baseline achieves only 75.03% accuracy across all query types, confirming that indiscriminate use of the SLM would degrade overall performance. Latency performance further validates the hybrid approach, with HybridLM completing queries in an average of 0.96 seconds, nearly 7% faster than the GPT-4.1-only baseline at 1.03 seconds, due to avoiding network overhead for locally processed queries.

5.3 Experiment 2: Custom Query Dataset Results

The custom 100-query dataset provides targeted insight into routing strategy performance and cost-quality tradeoffs across controlled complexity distributions.

The custom dataset results confirm the cost dynamics observed in the MMLU-Lite evaluation. GPT-4.1 LLM-only processes 100 queries for $0.04, while the SLM-only baseline incurs negligible cost at the expense of routing accuracy. HybridLM's total cost of $0.21 reflects its ensemble of Groq-hosted SLMs used for query assessment and local inference, which carries a per-call overhead absent

Table 2: Routing strategy performance comparison (100 queries).

Strategy	Routing Accuracy	Avg Latency	F1-Score
Rule-Based (Fallback)	59.0%	1.48 s	0.09
ML-Only (Logistic Reg.)	64.0%	1.68 s	0.28
ML+RL (Hybrid)	**87.0%**	**2.26 s**	**0.83**

in single-model deployments. Despite this, the hybrid approach delivers substantially better routing accuracy than either single-model baseline, as shown in Table 2.

Table 2 reveals the critical importance of routing strategy sophistication. Rule-based heuristics achieve only 59.0% routing accuracy, demonstrating that simple complexity thresholds are insufficient for effective hybrid routing. Machine learning improves this to 64.0%, while the hybrid ML+RL approach reaches 87.0% routing accuracy through epsilon-greedy exploration and reward-based online learning. This 28-point improvement over the ML-only baseline directly translates to better query outcomes: hard queries that would have been misrouted to the SLM are correctly escalated to GPT-4.1, and simple queries avoid unnecessary cloud calls.

5.4 Confusion Matrix Analysis

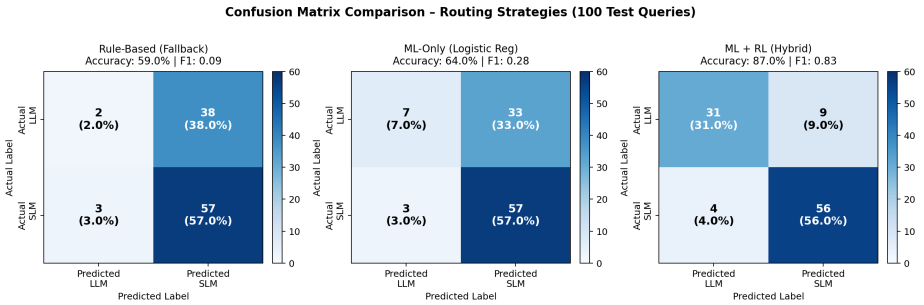


Fig. 7: Confusion matrix comparison of three routing strategies on 100 test queries: rule-based fallback, ML-only logistic regression, and the proposed ML+RL hybrid method. The hybrid strategy achieves the highest routing accuracy and F1-score.

The confusion matrices for the three routing strategies on 100 test queries reveal the progressive improvement in classification accuracy. Rule-based routing exhibits poor performance, correctly routing 57 of 60 simple queries to the SLM (95.0%) but only 2 of 40 hard queries to the LLM (5.0%), producing 38 false negatives in which complex queries are incorrectly handled by the local model.

This strong bias toward SLM routing yields a precision of 0.40 and a recall of just 0.05, resulting in an F1-score of 0.09. ML-only routing reduces false negatives to 33, improving LLM routing recall to 17.5% while maintaining 95.0% SLM accuracy. The improvement in precision to 0.70 raises the F1-score to 0.28, though the classifier remains conservative in escalating hard queries. The hybrid ML+RL approach achieves the best balance, correctly routing 56 of 60 simple queries (93.3%) and 31 of 40 hard queries (77.5%), with only 9 false negatives and 4 false positives. This yields a precision of 0.89, recall of 0.78, and an F1-score of 0.83. The progression across strategies demonstrates that reinforcement learning effectively addresses the misclassification bias of simpler approaches, particularly reducing the critical false negative rate where complex queries would receive inadequate responses from the SLM.

5.5 Analysis and Discussion

The experimental analysis shows that HybridLM occupies a distinct position within the cost–quality–latency design space in contrast to conventional single-model systems. Instead of operating along the existing tradeoff curve between high-cost, high-quality LLMs and low-cost, lower-quality SLMs, the combined framework—with its intelligent routing—creates an entirely new efficiency frontier. Results from the routing strategy evaluation indicate that routing accuracy is the main determinant of overall system performance. Improvements from rule-based routing (59.0%) to ML-only (64.0%) and ML+RL (87.0%) routing directly correspond to gains in both cost performance and response quality. The substantial cost reductions result from precisely recognising which queries can be handled by SLMs without sacrificing correctness.

Across both experiments, HybridLM consistently assigns a majority of queries to local models while reserving the cloud LLM for cases that require advanced reasoning. In the Global-MMLU-Lite evaluation, 31.5% of queries were handled locally, and the system achieved 86.25% accuracy — within 1.25 percentage points of GPT-4.1 alone. This near-equivalent accuracy demonstrates that the routing mechanism correctly identifies which queries the SLM can handle without quality loss. The primary cost driver is the Groq-hosted SLM ensemble used for complexity assessment, which introduces overhead per query but enables the routing decisions that bring accuracy close to the full LLM baseline. In deployments where SLM assessment cost is minimised or amortised, the cost advantage of HybridLM over a pure LLM approach would be more pronounced.

6 Conclusion and Future Work

HybridLM shows that deploying language models is not a binary choice between always paying for the very best model or settling for a lower-quality model to save money. This paper demonstrates that routing, orchestration, and caching can work together to make better use of both large and small models. In this way, HybridLM reshapes the usual trade-off between cost, quality, and latency.

Instead of picking a point on an existing tradeoff curve, the system routes most low-complexity questions to efficient local SLMs but escalates only truly difficult queries to powerful cloud LLMs. This results in large reductions in API spending while keeping accuracy close to premium baselines. The series-parallel orchestration layer highlights that how models are combined can matter as much as model size. Validating and refining SLM outputs using structured chains pushes small models closer to big-model quality for many tasks, all without changing their architecture. HybridLM also uses online learning and exploration.

This makes it a learning system that improves over time, responding to new query patterns and continually sharpening its routing. Combined, these parts make HybridLM a practical blueprint for organizations with real-world budget and latency constraints. This suggests that the future of scalable language model deployment will rely as much on intelligent orchestration as on advances in the models themselves.

References

1. Abdin, M., Aneja, J., Awadalla, H., Awadallah, A., et al.: Phi-3 technical report: A highly capable language model locally on your phone. arXiv preprint arXiv:2404.14219 (2024), https://arxiv.org/abs/2404.14219
2. Belcak, P., Heinrich, G., Diao, S., Fu, Y., Dong, X., Muralidharan, S., Lin, Y.C., Molchanov, P.: Small language models are the future of agentic AI. arXiv preprint arXiv:2506.02153 (2025), https://arxiv.org/abs/2506.02153
3. Chen, L., Zaharia, M., Zou, J.: FrugalGPT: How to use large language models while reducing cost and improving performance. In: Proceedings of the 40th International Conference on Machine Learning. pp. 4490–4513 (2023), https://arxiv.org/abs/2305.05176
4. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H.W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., et al.: PaLM: Scaling language modeling with pathways. Journal of Machine Learning Research **24**(240), 1–113 (2023), https://jmlr.org/papers/v24/22-1144.html
5. Cohere Labs: Global MMLU Lite benchmark. Kaggle Benchmarks (2024), https://www.kaggle.com/benchmarks/cohere-labs/global-mmlu-lite
6. Ding, D., Mallick, A., Wang, C., Sim, R., Mukherjee, S., Ruhle, V., Lakshmanan, L.V.S., Awadallah, A.H.: Hybrid LLM: Cost-efficient and quality-aware query routing. arXiv preprint arXiv:2404.14618 (2024), https://arxiv.org/abs/2404.14618
7. Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C.C.T., Del Giorno, A., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., Salim, A., Shah, S., Behl, H.S., Wang, X., Bubeck, S., Eldan, R., Kalai, A.T., Lee, Y.T., Li, Y.: Textbooks are all you need. arXiv preprint arXiv:2306.11644 (2023), https://arxiv.org/abs/2306.11644
8. Wang, F., Zhang, Z., Zhang, X., Wu, Z., Mo, T., Lu, Q., Wang, W., Li, R., Xu, J., Tang, X., He, Q., Ma, Y., Huang, M., Wang, S.: A comprehensive survey of

small language models in the era of large language models: Techniques, enhancements, applications, collaboration with LLMs, and trustworthiness. arXiv preprint arXiv:2411.03350 (2024), https://arxiv.org/abs/2411.03350