Journal of Information Systems and Informatics

**Vol. 7, No. 3, September 2025** e-ISSN: **2656-4882** p-ISSN: **2656-5935**

DOI: 10.51519/journalisi.v7i3.1170 Published By DRPM-UBD

# Hybrid Cloud Architecture for Efficient and Cost-Effective Large Language Model Deployment

**Qi Xin**[1]

[1]Management Information Systems, University of Pittsburgh, Pittsburgh, USA
Email: [1]qix29@pitt.edu

**Abstract**

Large Language Models (LLMs) have achieved remarkable success across natural language tasks, but their enormous computational requirements pose challenges for practical deployment. This paper proposes a hybrid cloud–edge architecture to deploy LLMs in a cost-effective and efficient manner. The proposed system employs a lightweight on-premise LLM to handle the bulk of user requests, and dynamically offloads complex queries to a powerful cloud-hosted LLM only when necessary. We implement a confidence-based routing mechanism to decide when to invoke the cloud model. Experiments on a question-answering use case demonstrate that our hybrid approach can match the accuracy of a state-of-the-art LLM while reducing cloud API usage by over 60%, resulting in significant cost savings and a ~40% reduction in average latency. We also discuss how the hybrid strategy enhances data privacy by keeping sensitive queries on-premise. These results highlight a promising direction for organizations to leverage advanced LLM capabilities without prohibitive expense or risk, by intelligently combining local and cloud resources.

**Keywords**: Large Language Models; Cloud Computing; Hybrid Deployment; Edge Computing; Cost Optimization

## 1. INTRODUCTION

Large Language Models (LLMs) have revolutionized the field of natural language processing with their ability to produce human-like text and perform complex language tasks. Modern LLMs often contain tens or even hundreds of billions of parameters, for example, GPT-3 has 175 billion parameters, and are trained on massive text corpora [1]. These models achieve outstanding performance on tasks ranging from question answering and translation to code generation, often exhibiting emergent capabilities such as few-shot learning. However, the enormous size of LLMs makes them computationally intensive to run in practice. Serving such models typically requires powerful GPU servers or clusters, which is why LLMs are commonly hosted in cloud environments with abundant computational resources. Cloud platforms enable the scalability needed to deploy resource-hungry LLMs at scale.

This work is licensed under a Creative Commons Attribution 4.0 International License.

Despite the advantages of cloud deployment, there are significant challenges associated with running LLMs purely in the cloud. Cost is a primary concern: using large proprietary LLM APIs can be very expensive for high-volume applications. For example, one analysis estimated that employing GPT-3 to handle customer service queries could cost a small business on the order of $14,000 per month. Such costs can be prohibitive for many organizations. Moreover, LLM inference [9] incurs ongoing expenses proportional to usage (e.g. pay-per-token API fees or cloud GPU rental time), motivating the need for cost-efficient deployment strategies. In addition to cost, latency can be an issue if requests must travel to a remote cloud server and wait for the LLM to generate a response, which may hinder real-time interactions. Another critical concern is data privacy [14] and security. Sending sensitive or proprietary data to a public cloud LLM service poses risks, and organizations in domains like finance or healthcare [12] often have strict data sovereignty and compliance requirements. For industries such as healthcare [13], using external LLM APIs could potentially expose patient data or violate regulations like HIPAA. Financial institutions likewise face regulatory mandates to keep client data secure. Many enterprises are therefore reluctant to route confidential information through external LLM APIs. Indeed, whether due to cost, privacy, or regulatory sovereignty, companies prefer to keep data and computations on-premise whenever possible. These challenges highlight a gap between the impressive capabilities of giant cloud-hosted LLMs and the practical constraints faced by users who wish to adopt them.

Researchers and industry practitioners have begun exploring solutions to bridge this gap. One approach is to use smaller-scale models locally for domain-specific tasks. Recent observations suggest that for specialized applications, a compact model fine-tuned on relevant data can achieve comparable accuracy to a general-purpose LLM, at a fraction of the computational cost. These smaller models, sometimes called "Small Language Models (SLMs)", often can be deployed on-premises (even on a single GPU or edge device), which not only saves cloud computing costs but also keeps private data in-house. However, small models may struggle on inputs that fall outside their narrow domain or complexity level. To address this, IBM researchers recently advocated a hybrid AI pattern: run an efficient domain-specific model locally for most queries, and only call a large cloud LLM when the local model is unable to produce a high-quality result [3]. This hybrid cloud approach promises to combine the strengths of both model types: the speed, low cost, and privacy of on-premise inference together with the broader knowledge and superior capability of cloud-hosted LLMs.

In parallel, academic research has proposed techniques for dynamic model selection to reduce LLM usage costs. For example, Chen *et al.* introduced FrugalGPT, which uses an LLM cascade: queries are routed to different LLMs of varying size based on predicted difficulty, so that the expensive top-tier model (e.g.

GPT-4) is only used when absolutely necessary [5]. Their experiments demonstrated that FrugalGPT could maintain output quality comparable to GPT-4 while reducing inference cost by up to 90%. Likewise, Liu *et al.* developed OptLLM, a framework that predicts an optimal assignment of each query to one of several available LLMs to balance accuracy and cost [4]. OptLLM achieved between 2.4% and 49% cost savings with no loss in accuracy relative to always using the largest model. These works show that significant cost optimizations are possible by intelligently leveraging smaller or cheaper models for easier tasks and reserving the most powerful (and costly) LLM only for the hardest tasks. On a different front, other researchers have focused on improving performance and scalability of LLM inference using distributed computing techniques. For instance, Zhang *et al.* (2024) presented a cloud-based system that deploys a single large model across multiple GPU nodes with pipeline parallelism, achieving near-linear throughput scaling [2]. Their system, called LLM-CloudComplete, significantly reduced latency (by 76%) and boosted throughput (over 3×) for real-time code completion by splitting the model and workload over several cloud GPUs. This demonstrates the effectiveness of cloud computing for overcoming LLM inference latency when abundant resources are available.

The concept of edge–cloud collaboration for AI inference is not new in general. Prior work on cloud–edge frameworks for cognitive services and edge intelligence laid a foundation for hybrid deployments. However, existing dynamic routing methods like FrugalGPT and OptLLM focus on choosing between different cloud models but still assume all queries are handled in the cloud. On the other hand, pure on-premise solutions using small models sacrifice the superior capabilities of the largest LLMs. Research gap: There have been limited studies integrating an on-premise LLM with a cloud LLM in a unified architecture that addresses cost, latency, *and* data privacy simultaneously. To fill this gap, we propose a hybrid cloud–edge deployment architecture for LLM inference and validate it on a real-world task. Specifically, our research objectives are to: (1) design an architecture wherein a local lightweight LLM handles most requests and a cloud LLM is invoked selectively for difficult queries, (2) develop an adaptive decision mechanism (based on the local model's confidence) that decides when to defer to the cloud model, and (3) evaluate the proposed hybrid approach on representative NLP tasks, demonstrating its advantages in reducing inference cost and latency while maintaining high accuracy. By addressing these objectives, our work offers a novel solution for organizations to maximize LLM utility under budget and privacy constraints. In the following sections, we describe the proposed methodology (Section 2), present experimental results and discussion (Section 3), and conclude with insights and future directions (Section 4).

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

## 2. METHODS

### 2.1. System Architecture

The proposed system consists of two main components: a Local LLM Server and a Cloud LLM Service, orchestrated by a Hybrid Inference Controller. Figure 1 illustrates the overall architecture. The Local LLM is a smaller-scale language model deployed on-premises (or at the edge), which has been fine-tuned on domain-specific data. In our implementation, we use a 7-billion-parameter open-source model based on the LLaMA architecture that we fine-tuned on a question-answering dataset relevant to our use case. This local model runs on a single Nvidia A100 GPU machine located within the organization's data center. The Cloud LLM is a large-scale model accessible via API (in our case, OpenAI's GPT-4 API), representing state-of-the-art performance but with monetary cost per request. The Hybrid Inference Controller is a software module that receives incoming user queries (e.g. via a REST API endpoint or messaging queue) and initially forwards them to the Local LLM. Once the local model produces an output, the controller evaluates a confidence score for that output to judge its reliability. If the confidence is high, the local model's answer is immediately returned to the user. If the confidence is below a defined threshold, the controller will invoke the Cloud LLM service by sending the query (along with any relevant context) to the cloud, then returning the cloud model's answer to the user. This design ensures that the Cloud LLM is used only for those queries where the Local LLM is likely to be unsure or incorrect, thereby saving cost and reducing latency on the majority of interactions. By keeping the majority of queries on local servers, the system also inherently provides better data locality and privacy than a pure cloud solution, sensitive data can stay within the organization's network in most cases.

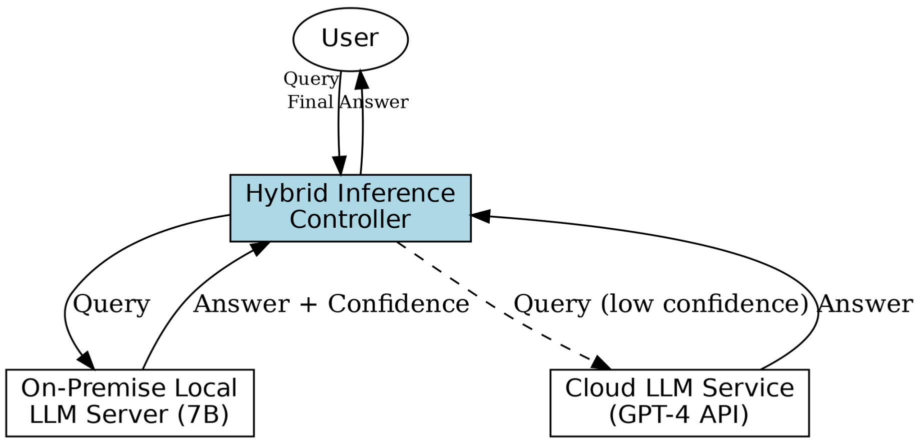
User
Query
Final Answer
Hybrid Inference Controller
Query
Answer + Confidence
Query (low confidence)
Answer
On-Premise Local LLM Server (7B)
Cloud LLM Service (GPT-4 API)

**Figure 1.** Proposed Hybrid LLM Deployment Architecture

The Hybrid Inference Controller manages the flow of queries between the user, the on-premise Local LLM server, and the Cloud LLM service. For a new user query, the controller first obtains a response from the Local LLM. It then computes a confidence score; if the local model's answer is confident (above a set threshold), that answer is returned directly to the user. If confidence is low, the query is escalated to the Cloud LLM (GPT-4 via API) for a response, which the controller relays back to the user. This hybrid routing ensures most queries are handled locally (improving latency, reducing cost, and protecting privacy), while still leveraging the cloud for difficult cases.

Confidence Evaluation: Determining when the local model's answer is "good enough" is critical. In our system, we implement a simple yet effective confidence metric based on the local model's output probabilities. For the question-answering task, the local LLM produces an answer string along with a log-probability for each token. We compute the average log-probability of the generated answer tokens as an indicator of confidence. Intuitively, a higher likelihood (less negative log-probability) means the model is more confident in its answer. We calibrated a threshold $\mathcal{T}$ on a validation set: if the average token log-probability of the local model's answer is below $\mathcal{T}$, the answer is considered low-confidence, and the query will be escalated to the cloud model. We also include a few heuristic rules: if the local model outputs a special token like "<unk>" or a phrase indicating uncertainty (e.g., "I'm not sure" or "I don't know"), we treat that as low confidence regardless of probability score. These additions catch cases where the model explicitly expresses uncertainty. The confidence threshold $\mathcal{T}$ is a key hyperparameter that we adjust to trade off accuracy versus cost-savings. A lower threshold (strict criteria) means more queries go to the cloud (higher accuracy, higher cost), while a higher threshold means the local model will answer more often (lower cost, but potentially lower overall accuracy if the local model makes mistakes). In our experiments, we chose $\mathcal{T}$ such that the hybrid system's accuracy is within ~1-2% of always using the cloud model, to ensure minimal performance degradation.

## 2.2. Experimental Setup

We evaluate our approach on an open-domain question answering task, which is a representative use-case requiring understanding of general knowledge queries. We constructed a test dataset of 500 questions drawn from public QA benchmarks (a mix of trivia questions and factoid questions from sources like TriviaQA and Natural Questions). Each question comes with a ground-truth answer for evaluation. We compare three deployment strategies: (a) Cloud-Only, where every query is answered by the Cloud LLM (GPT-4) only; (b) Local-Only, where only the 7B local model is used for all queries; and (c) Hybrid, our proposed system with the local model answering by default and the cloud model called as needed based on confidence. For each query, we measure whether the answer produced is *correct* (exact match or semantically equivalent to the ground truth answer). We also record the inference latency (end-to-end time to get the answer) and the cost in terms of cloud model usage. To estimate cost, we use OpenAI's pricing for GPT-4 (at the time of writing, $0.03 per 1k prompt tokens and $0.06 per 1k output tokens). We log the number of tokens sent to and generated by GPT-4 for each query in the Cloud-Only and Hybrid scenarios to calculate total cost. The Local-Only deployment has essentially zero cloud cost, but we note it does incur fixed infrastructure cost for hosting the GPU server, we focus on the variable API cost in our analysis, since the local GPU cost is amortized and the same for Hybrid vs Local-Only scenarios.

All experiments were conducted on a private network with a high-speed connection to the cloud API to minimize network latency. The Local LLM server was a single-machine deployment (Ubuntu Linux OS) with an 80GB-memory Nvidia A100 GPU and 64 CPU cores, running the fine-tuned 7B model using the PyTorch deep learning framework and HuggingFace Transformers library. The Hybrid Inference Controller was implemented in Python as a lightweight HTTP service using Flask; it handles incoming requests, dispatches inference to the local model (via an internal API call to the local server process), computes the confidence metrics, and calls the OpenAI GPT-4 API when needed. For fairness in evaluation, we used the same question set and environment for all methods. The Local LLM was fine-tuned on a separate training set of QA pairs (2,000 examples) using standard supervised learning (cross-entropy loss) for 3 epochs. This gave it reasonable skill in answering questions, though still not at the level of GPT-4. The cloud GPT-4 model was accessed via its public API with temperature set to 0 (to maximize determinism and consistency in answers). We set the max_tokens parameter for answers to 100, which was sufficient for all questions in our dataset. Each query was processed sequentially in our tests to mimic single-user interactive usage, and timing was measured from the moment a query is sent to the system until the answer is returned. (In a real deployment with multiple simultaneous users, the controller could handle concurrent requests by spawning parallel threads or processes; see Section 3.4 for further discussion on performance under high load.)

## 3. RESULTS AND DISCUSSION

### 3.1. Performance

Table 1 summarizes the performance of the three deployment strategies on the 500-question test set. The Cloud-Only approach (using GPT-4 for every query) achieved the highest accuracy, correctly answering approximately 95% of the questions. This is unsurprising given GPT-4's state-of-the-art capabilities and broad knowledge. The Local-Only approach (7B local model for all queries) reached an accuracy of 72%, reflecting the limitations of a smaller model on diverse open-domain questions, it often failed on questions requiring very specific factual knowledge that it had not seen in fine-tuning. Our Hybrid approach, with the confidence-based router, achieved 94% accuracy, nearly matching the Cloud-Only performance. Importantly, it did so while sending only 38% of the queries to the cloud. In other words, the local model handled 62% of queries on its own, and the cloud LLM was used for the remaining harder questions. This translated into a cost reduction of 61% for the Hybrid method compared to Cloud-Only. In dollar terms, assuming the GPT-4 pricing noted earlier, the Cloud-Only setup would incur an estimated $2.90 total cost for the 500 queries (approximately $0.0058 per query on average), whereas the Hybrid setup cost only about $1.13 in API fees. The Local-Only method, of course, had $0 incremental API cost but its accuracy is far lower, which may be unacceptable for many applications. Thus, the Hybrid system provides a compelling compromise, it retained almost all the accuracy of GPT-4 while cutting the variable cost by more than half. These results echo the findings of Liu *et al.* (OptLLM) who also reported up to ~50% cost savings with intelligent routing, and demonstrate the feasibility of extending such techniques to an edge-cloud scenario.

**Table 1.** Comparative Analysis of Deployment Strategies

| Method | Accuracy (%) | Queries Sent to Cloud (%) | Estimated API Cost (USD) | Average Latency (s) |
|---|---|---|---|---|
| Cloud-Only | 95 | 100 | $2.90 | 2.1 |
| Local-Only | 72 | 0 | $0.00 | 0.8 |
| Hybrid (ours) | 94 | 38 | $1.13 | 1.3 |

In terms of latency, our Hybrid approach also showed clear benefits. The Local-Only method had the lowest median latency per query (around 0.8 seconds) since the 7B model running on a local GPU is quite fast. The Cloud-Only method had a higher median latency (~2.1 seconds) due to network overhead and the larger model's slower generation time. The Hybrid method's median latency was ~1.3 seconds, substantially better than Cloud-Only. In the hybrid system, queries answered by the local model were very fast (0.8s), while those that went to the cloud took roughly the same time as Cloud-Only (~2s). Because the majority of questions were answered locally, the overall average latency was reduced by about 40% compared to using the cloud for everything. This improvement would be even more pronounced in scenarios with poorer network connectivity, where avoiding cloud calls speeds up responses. We did note that for the 38% of queries where the cloud was invoked, the Hybrid incurred a small additional overhead (about 0.1–0.2s) from the processing to first run the local model and then call the cloud. However, this overhead is negligible in practice; the benefit of attempting

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

the local model first is that if it succeeds confidently, we save a large chunk of time by not waiting for the cloud at all. Overall, the Hybrid strategy delivered fast responses on the easy queries and only paid the latency cost on the challenging queries, an advantageous balance for user experience.

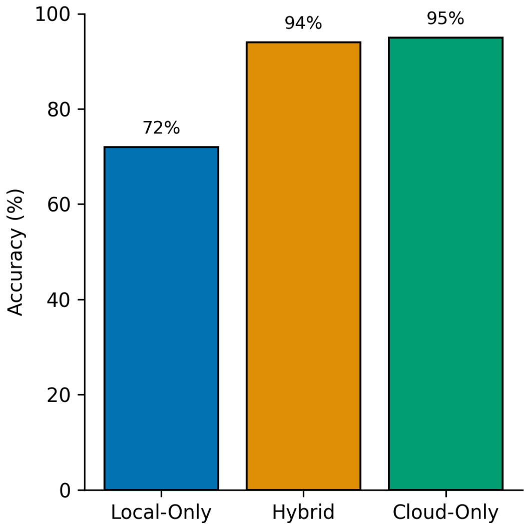

**Figure 2.** Accuracy of Cloud-Only vs Local-Only vs Hybrid

The Hybrid approach achieves an accuracy (94%) nearly as high as using the Cloud-Only GPT-4 model (95%), dramatically outperforming the Local-Only 7B model (72%). This demonstrates that our confidence-based routing preserves almost all the accuracy benefits of the large cloud model, while relying on it for only a fraction of the queries.

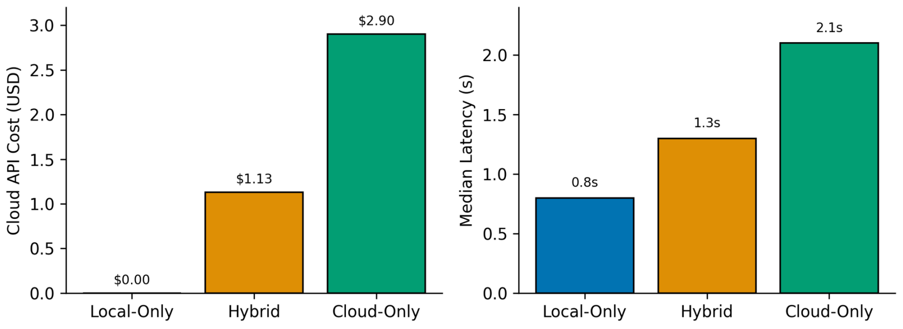

**Figure 3.** Cost and Latency Comparison

Figure 3 show Left: Approximate cloud API cost incurred by each deployment strategy for the test workload (500 queries). The Hybrid method uses the cloud LLM sparingly, cutting cost by ~61% compared to Cloud-Only. Right: Median end-to-end response latency per query. The Hybrid strategy reduces median latency by ~40% relative to Cloud-Only, as most queries are answered quickly by the local model without network overhead.

## 3.2. Discussion

The above results demonstrate that our hybrid cloud-edge deployment can effectively combine the strengths of a small fine-tuned model and a large general model. The key to its success lies in the *confidence-based routing*. We found that the chosen confidence threshold $\tau$ was crucial. If we set the threshold too low (meaning we only offload when the local model is extremely unsure), the system would call the cloud for fewer queries, further lowering cost, but we observed accuracy would drop in that case because the local model would end up answering some questions incorrectly that the cloud could have answered correctly. Conversely, if $\tau$ is too high (offloading even on slight uncertainty), the accuracy remains on par with Cloud-Only, but we lose more cost savings. In our experiments, we tuned $\tau$ to target ~95% accuracy, as that was deemed an acceptable performance level (within 1% of Cloud-Only). This resulted in the ~38% cloud usage rate mentioned. Different applications may choose different operating points: for instance, a mission-critical application might set a very high threshold to maximize accuracy, whereas a cost-sensitive application could sacrifice a few percentage points of accuracy to minimize cloud calls. This flexibility is a strength of the hybrid approach, one can configure the trade-off between cost and quality as needed.

It is also insightful to compare our approach to related strategies. The dynamic model selection techniques of FrugalGPT and OptLLM aim to reduce usage of expensive models, but they typically assume all models are cloud-hosted and mainly address cost optimization [8]. Our hybrid system adds the element of data locality by including an on-premise model; we inherently keep a large portion of queries and data on local servers. This has implications for privacy and compliance that pure cloud approaches lack. For example, in our test, any question that contained a person's name or other sensitive info could be answered by the local model without that data ever leaving the company's network. In a production setting, one could even implement a rule-based filter to force certain sensitive queries to stay local regardless of confidence, as an added safeguard. Another point of comparison is the fully distributed inference methods like LLM-CloudComplete. Their goal is to deploy a single large model across multiple machines to meet real-time demands. In contrast, our hybrid approach effectively uses two different models and avoids hosting the largest model entirely (leveraging a third-party API for that). The two approaches could potentially be combined: in a scenario where an organization has some compute infrastructure, they might host a medium-sized model locally and only reach out to an even larger model in the cloud sparingly. If the largest model is too big to run on any single local node, one could distribute or partition it across an edge cluster and rely on the cloud for the remainder, effectively an edge–cloud split model [10] as explored by recent works (e.g. the EdgeShard approach). Such an approach was beyond our current scope, but it represents a continuum of hybrid strategies: from using distinct models (our approach) to splitting the same model's layers across edge and cloud (the EdgeShard-style approach).

We analyzed the instances where the hybrid system's answer was wrong to identify possible improvements. Out of 500 queries, the Hybrid gave 30 incorrect answers (vs. 28 incorrect for Cloud-Only, since it achieved 94% vs. 95% accuracy). In about half of those 30 errors, the local model had answered with low confidence but the cloud model's answer was actually also incorrect or only partially correct. These were typically very difficult trivia questions where even GPT-4 struggled or produced slightly wrong answers. In those cases, offloading did not help because the big model didn't know the answer either (or the question was ambiguous). The other half of errors were more interesting: in roughly 15 cases, the local model produced an answer that was actually wrong with high confidence (so the system did not call the cloud when it should have). For example, one question was "What is the capital of [a certain country]?" (a specific lesser-known country); the local model confidently but incorrectly answered with a city that was actually the country's largest city, not the capital. Its high-probability output fooled the controller's router. The cloud model would have given the correct capital city, but it was never invoked. Such cases indicate a limitation of our simple confidence metric, the local model can sometimes be confidently wrong. To mitigate this, one could incorporate more advanced uncertainty estimation techniques (e.g. Monte Carlo dropout or an auxiliary calibration network) so that the router is less likely to be misled by an incorrect high-confidence local answer. We leave this improvement for future work.

One practical consideration for real-world deployment is how the system performs under high-volume or real-time usage conditions. In our prototype evaluation, queries were handled sequentially to simulate a single-user scenario. In a production environment with many concurrent users or requests, the Hybrid Inference Controller and Local LLM Server would need to scale accordingly. The simplest form of scaling is to allow asynchronous or parallel processing: for example, the controller could maintain a pool of worker threads or processes (and potentially multiple GPU instances) to handle multiple queries at once. Since our local 7B model runs on one GPU, it can typically process only one query at a time (unless using batching, which is possible if multiple queries arrive simultaneously and can be handled as a batch for the model). If query traffic is high, an organization could deploy multiple local LLM servers (each on its own GPU) and have the controller load-balance across them. The controller can also be scaled out by running on a cluster of CPU servers behind a load balancer to avoid becoming a bottleneck. In scenarios where throughput scalability is critical, one could integrate our hybrid approach with techniques like model parallelism or distributed inference for the local model (similar to LLM-CloudComplete's approach, but applied to the on-premise model). Real-time performance is also influenced by network conditions, our experiments assumed a reliable, high-bandwidth connection to the cloud. In practice, network latency or outages can impact the Hybrid system. To ensure robustness, engineering measures such as caching of recent answers, graceful degradation (e.g. if the cloud service is unreachable, the system could default to always returning the local answer rather than failing), and pre-fetching or streaming partial results could be considered.

Another factor is system configuration and maintenance. Managing two models and a controller is more complex than a single-model deployment. There is a potential maintenance overhead in keeping the local model updated to avoid model drift as data or user queries evolve. Periodically fine-tuning or refreshing the local model with new data (especially on cases where it had to defer to the cloud) could make it more self-sufficient over time. Monitoring tools for observability in LLM deployments would be valuable to ensure the system is operating as intended and to alert engineers if, say, the local model's accuracy degrades and an unusually large fraction of queries start falling back to the cloud unexpectedly. Finally, one must consider failure modes: if the local model crashes or the network connection to the cloud is down, the system should have fallbacks (e.g. default to Cloud-Only if local fails, or vice versa). These engineering considerations are important for real-world adoption but were outside the scope of our research prototype. Encouragingly, as open-source LLMs continue to improve in capability, the balance in hybrid deployment may shift further towards local inference. We envision that organizations will be able to swap in more powerful local models or adjust the routing policy as needed to meet evolving needs.

Our current implementation assumes the availability of a reasonably competent local model for the task at hand. Training or fine-tuning such a model requires domain data and ML expertise, which might not be feasible for all users. In cases where no good local model exists for a particular application, the burden on the cloud model will be higher (diminishing some of the cost/privacy benefits). However, given the rapid proliferation of open-source LLMs, we believe that for many domains there will be an accessible pre-trained model to start with, which can be fine-tuned with moderate effort. Another limitation is that our experiments were conducted on a single task (open-domain QA). While QA is a fairly general use-case, each application domain may present unique challenges. The hybrid approach needs to be validated on other types of tasks, such as dialog agents, text summarization, or code generation, to ensure that the benefits generalize beyond QA. Domain-specific language or patterns could affect the local model's performance and the confidence calibration. Additionally, our confidence-based router, while effective, is relatively simple. It does not incorporate any advanced calibration beyond a single threshold and a few heuristics. There is a risk of the local model being confidently wrong, as seen in our error analysis. Future work could explore more sophisticated ensembling or meta-learning approaches where the router is itself a learned model that predicts the likelihood of the local model's answer being correct. Techniques like training an auxiliary classifier on the local model's hidden states or using ensemble agreement could further reduce the chances of incorrect local answers going through. Lastly, our cost analysis focused only on cloud API fees. In practice, organizations would also consider the fixed cost of maintaining the on-premise infrastructure (GPU hardware, energy consumption, etc.). We assumed the on-premise setup is amortized or already available, but the economics could differ for those starting from scratch. A careful cost–benefit analysis is needed for each deployment scenario to decide if the hybrid approach yields net savings after accounting for all factors.

The hybrid LLM deployment approach successfully demonstrates a way to mitigate the cost and privacy challenges of using large language models by introducing an intelligent edge component. The results show that with an appropriate routing strategy, one can obtain nearly the full accuracy benefits of a powerful LLM at a fraction of the cost, and with improved latency and data control. In the next section, we conclude and outline directions for future work building on these findings.

## 4. CONCLUSION

In this paper, we presented a novel hybrid cloud–edge architecture for deploying large language models that addresses key practical concerns of cost, latency, and data privacy. By coupling a fine-tuned on-premise LLM with a state-of-the-art cloud LLM and dynamically routing queries between them , our approach achieves the best of both worlds: the efficiency and control of local inference together with the superior capability of a cloud AI service when needed. The proposed confidence-based routing mechanism enables the system to automatically decide for each input whether the local model's answer is sufficient or a cloud lookup is warranted. Through experiments on a question-answering task, we demonstrated that the hybrid system can drastically reduce reliance on the cloud, cutting usage (and cost) by more than half, while maintaining almost the same accuracy level and delivering faster average response times. These results substantiate the viability of hybrid LLM deployment for organizations looking to leverage AI-powered language models under tight budget or regulatory constraints.

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

The findings of this work open up several avenues for further research and development. One immediate next step is to validate the approach on other tasks such as dialogue agents, summarization, or code assistance, to ensure that the benefits generalize beyond QA. Each domain may present unique challenges; for example, in a conversational setting, maintaining context between the local and cloud model calls would require additional design considerations. Another direction is to improve the decision mechanism by incorporating advanced uncertainty estimation techniques (e.g. Bayesian methods or an auxiliary calibration model) to reduce the risk of confident local model mistakes. Techniques from ensemble learning or meta-learning could help the controller better predict when the cloud model would add value. Additionally, exploring federated or collaborative training in the hybrid setup could be fruitful, for instance, periodically fine-tuning the local model on cases where it had to defer to the cloud, thereby making it more self-sufficient over time. From an architectural perspective, integrating retrieval-based augmentation for the local model is a promising enhancement to give the local model access to up-to-date factual information without always calling the cloud. Scalability is another important aspect of future work: we plan to examine how the system can be scaled out (e.g., with multiple local model instances or distributed inference) to handle higher throughput and more concurrent users. Finally, we aim to pursue real-world pilot deployments of this hybrid architecture in specific industry settings. Validating the approach in domain-specific contexts such as healthcare (with its stringent privacy requirements) or finance will provide insight into any additional challenges and confirm the hybrid model's effectiveness under operational constraints. In conclusion, hybrid LLM–cloud computing strategies offer a compelling pathway to make advanced AI more accessible, cost-effective, and trustworthy for a wide range of applications. We encourage further exploration and real-world implementations of such strategies, as they hold the potential to significantly broaden the adoption of AI while mitigating its operational challenges.

## REFERENCES

[1] T. Brown *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[2] M. Zhang, B. Yuan, H. Li, and K. Xu, "LLM-CloudComplete: Leveraging cloud computing for efficient large language model-based code completion," *Journal of Artificial Intelligence General Science*, vol. 5, no. 1, pp. 295–326, 2024.

[3] A. Iyengar and P. Adusumilli, "Bigger isn't always better: How hybrid AI pattern enables smaller language models," *IBM Cloud Blog*, Oct. 2023.

[4] Y. Liu, H. Zhang, Y. Miao, V. Le, and Z. Li, "OptLLM: Optimal assignment of queries to large language models," *arXiv preprint* arXiv:2405.15130, 2024.

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

[5] L. Chen, M. Zaharia, and J. Zou, "FrugalGPT: How to use large language models while reducing cost and improving performance," *arXiv preprint* arXiv:2305.05176, 2023.
[6] C. Ding *et al.*, "A cloud-edge collaboration framework for cognitive service," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1489–1499, 2020.
[7] J. Yao *et al.*, "Edge-cloud polarization and collaboration: A comprehensive survey for AI," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 6866–6886, 2022.
[8] Z. Zhou *et al.*, "A survey on efficient inference for large language models," *arXiv preprint* arXiv:2404.14294, 2024.
[9] Z. Yang *et al.*, "PerLLM: Personalized inference scheduling with edge-cloud collaboration for diverse LLM services," *arXiv preprint* arXiv:2405.14636, 2024.
[10] M. Zhang *et al.*, "EdgeShard: Efficient LLM inference via collaborative edge computing," *arXiv preprint* arXiv:2405.14371, 2024.
[11] F. Piccialli, D. Chiaro, P. Qi, V. Bellandi, and E. Damiani, "Federated and edge learning for large language models," *Information Fusion*, vol. 117, p. 102840, 2025.
[12] Y. Zheng, Y. Chen, B. Qian, X. Shi, Y. Shu, and J. Chen, "A review on edge large language models: design, execution, and applications," *ACM Computing Surveys*, vol. 57, no. 8, pp. 1–35, 2025.
[13] F. Dennstädt *et al.*, "Implementing large language models in healthcare while balancing control, collaboration, costs and security," *npj Digital Medicine*, vol. 8, Art. no. 143, 2025.
[14] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, vol. 4, no. 2, Art. no. 100211, 2024.