

LLM-MS: A Multi-Model LLM Search Engine

Konstantin Krasovitskiy
Department of Computer Science
University of Cyprus
Nicosia, Cyprus
krasovitskiy.konstantin@ucy.ac.cy
0009-0000-6971-0463

Stelios Christou
Department of Computer Science
University of Cyprus
Nicosia, Cyprus
christou.s.stelios@ucy.ac.cy

Demetrios Zeinalipour-Yazti
Department of Computer Science
University of Cyprus
Nicosia, Cyprus
dzeina@ucy.ac.cy
0000-0002-7239-2387

Abstract—Large Language Models (LLMs) are evolving at an unprecedented pace, with new models emerging almost daily. This has created a significant challenge for users trying to identify the most suitable model for a given task. With thousands of models available, selecting the right LLM has become increasingly complex. Retraining or fine-tuning a single model for general-purpose use is computationally expensive, making specialized models a practical necessity. However, the lack of tools to dynamically evaluate and combine these models limits their accessibility and effectiveness. In this paper, we introduce LLM Meta Search (LLM-MS), a web-based multi-model system that dynamically selects and allocates resources to LLMs based on query characteristics. To achieve this, we propose novel LLM selection algorithms that optimize token allocation, balancing response accuracy and computational cost. Our evaluation demonstrates that LLM-MS significantly reduces token usage and response time compared to traditional single-model approaches while maintaining high-quality outputs.

Keywords—LLM Meta Search, Multi Modal System.

I. INTRODUCTION

The rapid growth of Large Language Models (LLMs) [1], [2] has transformed Natural Language Processing (NLP), enabling new opportunities in text generation and summarization. Open-source LLMs in particular have played a crucial role in democratizing AI, encouraging researchers and developers worldwide to drive technological progress and create specialized models like GPT2 [3], GPT-Neo [4], Llama [5], Mistral [6], Gemini 1.0 [7] and Falcon [8]. Techniques like distillation, which replicate specific functionalities of existing models, have further contributed to the expansion of LLM variants (e.g., Deepseek-R1 [9]/V3 [10]) with the latest developments promoting lower latencies, lower computation costs and enhanced performance built to power agentic experiences e.g., Gemini 2.0 Flash [11].

As a result, the number of available LLMs has exploded, with Hugging Face [12] alone hosting over 1,563,509 LLM models, and Meta’s Llama ecosystem having more than 150 variants given that many models pass frequently through revisions (e.g., Llama 3, 3.1, 3.2) but also given that these models are available in different sizes and configurations (e.g., Llama 8B, 70B, 405B). While the above promotes accessibility and flexibility, it also introduces a major challenge: **how can users efficiently select the right LLM for a specific task?** For example, *LLM1 might be better in coding, while LLM2 might be better in aiding with writing and LLM3 in reasoning.*

A seemingly logical solution is to train or fine-tune a massive monolithic (*foundation model* [13]) that is highly capable to succeed across multiple fields and tasks. Yet this is an extremely resource intensive process, requiring substantial computational power and time, with scalability limitations mostly at training time, but also at inference time. It also doesn’t solve the challenge of efficiently deploying the large ecosystem of open pre-trained models, contributing to the diversification of open knowledge from different sources to yield truthful answers. This also doesn’t reduce the sustainability and environmental footprint of LLMs [14]. Another approach is to decide a priori on which pre-trained LLMs to use for specific tasks. However, it is error-prone to decide which LLM to use for specific and over-specialized problems.

Recent development of *Mixture of Experts (MoE)* LLMs (e.g., Mixtral [15], Deepseek-R1/V3 [9], [10], Switch Transformer [16] or rumors about GPT-4 MoE [17]), which is a technique that deploys multiple specialized sub-models (experts) from the same architecture/family with a gating network [18] to select the best expert for each token of the prompt. MoEs are focused on *computational efficiency* (i.e., less GPU VRAM necessary to load LLMs to GPU VRAM) and an improved *inference strategy* (i.e., better inference through dynamically routing individual prompt parts through different experts of the same LLM.)

As an example of the *computational efficiency* advantage MoE pose, consider that the traditional Llama-405B LLM with 16bit quantization won’t fit on a NVIDIA v100 card that has 32GB of GPU VRAM. As an example of a custom inference strategy MoE pose, consider the Mixtral MoE that has 8 experts and deploys at any given time point only 2 of them [15]. MoE LLMs, like traditional LLMs, are focused on a **single-LLM-at-a-time** approach and do not diversify their responses through the deluge of new LLM models that are released on a daily basis. This leads to a degraded user experience as responses will have emerged from particular LLMs that have applied very specific ethical curation, bias mitigation, stylistic preferences (e.g., Shakespearean style) and preference models (formal/informal, professional/casual) [13]. Finally, current approaches, like Google’s Vertex AI Studio [19], are simply tools for rapidly prototyping and testing generative AI models and this is merely used for developers to test individually models as they emerge.

advantage of the expertise of each distinct model, making the system capable of handling more complex queries.

Prior research on the usage of multiple LLMs has already been conducted [21][22]. However, most approaches lean heavily on the use of ensembles or fusion models rather than the use of multiple individual LLMs on their own in a single system. LLM-Blender [21] for example, describes a framework which first ranks multiple models based on their response for a given query and then merges the top-ranked models to generate a response, allowing it to grasp on the strengths of each model. In a similar manner, FuseLLM [22] proposes a fusion-based approach, in which the capabilities of existing LLMs are combined into a new LLM that incorporates the strengths of each LLM.

MetaLLM [23] has introduced a cost-based optimization framework for dynamically selecting and combining LLMs, but their approach is primarily focused on minimizing computational costs. In contrast, our work emphasizes content quality as the primary optimization criterion. Research has also been conducted on the ranking of LLMs based on given queries; however, it is not directly tied to multi-model systems as we do in this work. ZOOTER [24] proposes a method for finding an optimal routing function to direct queries to the best fitting LLM, and similarly Shnitzer et al. [25] differentiate models based on their suitability for each scenario. PandaLM [26] along with Zheng et al. [27] describes the use of LLMs as a way to evaluate and rank other LLMs.

While the primary focus of this work is the usage of multiple LLMs within a single system, related research has provided complementary techniques that aim to enhance LLM performance. Studies on vector databases demonstrate their role in improving response accuracy, as seen in RAG [28] and LLM for Data Management [29], and optimizing response time, as described in VELO [30].

III. THE LLM-MS SEARCH ALGORITHMS

In this section, we describe two of the search algorithms we have developed for the LLM-MS architecture. The first algorithm, named *LLM-MS-OUA (Overperformers-Underperformers Algorithm)*, carries out a best-search LLM traversal method that identifies the over-performers and the under-performers at each step of the search process. The second algorithm, named *LLM-MS-MAB (Multi-Armed Bandit Algorithm)*, is a reinforcement learning algorithm that carries out the exploration-exploitation trade-off dilemma using a UCB1 strategy. LLM-MS-OUA focuses on dynamically pruning weaker models to concentrate computational resources on the strongest candidates, and LLM-MS-MAB balances exploration and exploitation by adaptively selecting models based on their historical performance and uncertainty estimates.

A. LLM-MS-OUA

This algorithm (shown in Algorithms 1) applies a heuristic-based search, similar to A^* , which dynamically selects the most relevant response from multiple LLMs during text generation. It operates in a token-by-token manner, allocating a fixed number of tokens (λ) to each model and iteratively evaluating

Algorithm 1 LLM-MS-OUA

(Overperformers-Underperformers Algorithm)

Input: Set of LLMs $LLM = \{LLM_1, LLM_2, \dots, LLM_N\}$, User Prompt p , Maximum Tokens λ_{max}

Output: Best Response (r_p)

```

1:  $\alpha \leftarrow 0.7, \beta \leftarrow 0.3$   $\triangleright$  Tuning parameters for response scoring
2:  $\lambda \leftarrow \frac{\lambda_{max}}{N}$   $\triangleright$  Allocate tokens equally among  $N$  LLMs
3:  $responses \leftarrow \{\}$   $\triangleright$  Store model responses
4:  $embeddings \leftarrow \{\}$   $\triangleright$  Store response embeddings
5:  $scores \leftarrow \{\}$   $\triangleright$  Store similarity scores
6:  $doneReasons \leftarrow \{\}$   $\triangleright$  Store termination reasons
7:  $bestResponse \leftarrow \emptyset$   $\triangleright$  Initialize best response
8:  $bestScore \leftarrow 0$   $\triangleright$  Initialize best similarity score
9:  $activeModels \leftarrow LLM$   $\triangleright$  Set of active models
10: while  $|activeModels| > 0$  do
11:   for all  $LLM_i \in activeModels$  do
12:      $response_i, doneReason_i \leftarrow getChunk(LLM_i, p, \lambda)$ 
13:      $responses[LLM_i] \leftarrow response_i$ 
14:      $doneReasons[LLM_i] \leftarrow doneReason_i$ 
15:   end for
16:   for all  $LLM_i \in activeModels$  do
17:      $embedding_i \leftarrow vectorize(responses[LLM_i])$ 
18:      $embeddings[LLM_i] \leftarrow embedding_i$ 
19:      $qScore \leftarrow cosineSimilarity(embedding_i,$ 
20:        $vectorize(p))$ 
21:      $interScore \leftarrow InterModelSimilarity(embedding_i,$ 
22:        $embeddings)$ 
23:      $scores[LLM_i] \leftarrow (\alpha \times qScore) + (\beta \times interScore)$ 
24:   end for
25:    $bestModel \leftarrow \max(scores)$ 
26:    $secondBestScore \leftarrow secondMax(scores)$ 
27:   if  $scores[bestModel] > secondBestScore + 0.5$  and
28:      $doneReasons[bestModel] = \text{"stop"}$  then
29:     return  $(bestModel, responses[bestModel])$   $\triangleright$  Early
30:     stopping
31:   end if
32:    $worstModel \leftarrow \min(scores)$ 
33:    $secondWorstScore \leftarrow secondMin(scores)$ 
34:   if  $secondWorstScore - scores[worstModel] > 0.5$  then
35:      $activeModels \leftarrow activeModels \setminus \{worstModel\}$   $\triangleright$ 
36:     Prune low-scoring models
37:   end if
38:   for all  $LLM_i \in activeModels$  do
39:     if  $doneReasons[LLM_i] = \text{"finished"}$  then
40:        $activeModels \leftarrow activeModels \setminus \{LLM_i\}$ 
41:     end if
42:   end for
43:    $bestPerformer \leftarrow \max(scores)$ 
44: return  $(responses[bestPerformer])$   $\triangleright$  Return best response

```

responses in real-time, using vector embeddings and semantic similarity. This approach ensures that the system identifies the most accurate response while minimizing unnecessary computational overhead. In this subsection, we discuss our LLM-MS-OUA algorithm in depth and its related components and parameters.

Token Allocation: Given a maximum token budget of $\lambda_{max} = 2048$, the algorithm begins by evenly distributing the available tokens among N models such that each model receives $\lambda_m = \frac{\lambda_{max}}{N}$. For example, if $N = 4$, each model is allocated 512 tokens. This ensures that all models operate with the same

initial token budget (although more careful analysis of this design choice remain a topic of future research).

Round-Robin Response Generation: Each model generates responses one token at a time. The algorithm employs a round-robin strategy to collect responses from all models in a balanced manner. This ensures that no single model dominates the response generation process. The responses are continuously evaluated for accuracy, similarity, and conciseness.

Vectorization and Similarity Scoring: Each token-level response is vectorized into a 1024-dimensional embedding using a dedicated AI model. These embeddings are then compared to the query embedding using cosine similarity to compute a similarity score. This score quantifies how closely the response aligns with the original query.

Outlier Detection and Model Pruning: The algorithm continuously compares the response of each model against:

- The original user prompt (to ensure relevance).
- Other model responses (to detect inconsistencies and outliers).

After comparing the responses, if a model’s response is identified as an outlier (i.e., its similarity score is significantly lower than the others), it is stopped and is not taking into further consideration, thus reducing the number of models being used.

Termination Criteria: The algorithm terminates under one of the following conditions:

- A model produces a response with a similarity score that is 0.5 or higher than the others, indicating a clear best response.
- The maximum token limit is reached, in which case the model whose response is complete and has the highest similarity score is considered the best response.
- All the models finished gracefully below the token limit, in which case the model with the best response is picked

Iterative Refinement: The process continues token-by-token until an optimal response is determined based on the defined stopping criteria.

Example: Consider a scenario with four LLMs ($N = 4$) and a maximum token limit of 2048 tokens. Each model is initially allocated 512 tokens. The algorithm begins by collecting token-level responses from each model in a round-robin fashion. After vectorizing and scoring the responses, it identifies that Model 2 produces a significantly lower similarity score. As a result, Model 2 is terminated for further token allocations, while Models 1, 3 and 4 continue generating their responses. After some iterations, Model 3 appears to produce a low similarity score, while Model 4 produces a high similarity score, exceeding the 0.6 threshold. Regardless of the outcome of models 1 and 3, as Model 4 has surpassed the set threshold, the algorithm terminates, and the response of Model 4 is returned as the best output.

B. LLM-MS-MAB

The LLM-MS-MAB (Multi-Armed Bandit) leverages the UCB1[31] algorithm to dynamically allocate computational resources across multiple LLMs. Our approach balances exploration and exploitation to efficiently determine the best-performing model while sticking to a strict token budget. In this subsection, we discuss our LLM-MS-MAB algorithm and its related components and parameters.

Token Allocation Strategy: Given a total token budget of $\lambda_{\max} = 2048$, the algorithm does not evenly distribute tokens upfront. Instead, it dynamically assigns tokens on demand to the most promising model at each step. The selected model can request additional tokens in small increments, allowing for fine-grained control over generation.

UCB1 Bandit Selection for Model Prioritization: At each step, the algorithm must decide which model to query next. To achieve this, we employ the UCB1 multi-armed bandit algorithm, which maintains an adaptive reward system based on past performance.

- Each model is treated as a bandit arm and starts with zero knowledge of response quality.
- The bandit algorithm explores all models *at least once* before prioritizing models that have demonstrated higher response quality.
- The UCB1 selection criterion is defined as:

$$UCB_i = \frac{\text{total reward}_i}{\text{number of pulls}_i} + \gamma \cdot \sqrt{\frac{2 \ln(\text{total pulls})}{\text{number of pulls}_i}}$$

where γ is a dynamic exploration coefficient that controls the exploration-exploitation trade-off. We adapt γ over time using the formula:

$$\gamma = 0.3 \cdot \left(1 - \frac{\text{usedTokens}}{\lambda_{\max}}\right)$$

This ensures that exploration is prioritized in the early stages of the algorithm but gradually decreases as more tokens are consumed, forcing the algorithm to stabilize and converge to the best-performing model.

- Models that have produced higher similarity scores compared to both the original question and other models are selected more frequently for additional token allocations.

Embedding and Similarity Scoring: After generating each token chunk, the semantic quality of the response is evaluated using cosine similarity scoring.

- **Query Embedding:** The user’s input is converted into a high-dimensional vector representation.
- **Response Embedding:** Each token chunk generated by a model is transformed into an embedding vector.
- **Scoring Mechanism:**
 - The similarity between the response embedding and the query embedding is computed using cosine similarity.
 - The response is compared against the responses generated by the rest of the models.

- A weighted reward function is applied:

$$\text{reward} = \alpha \cdot \text{cosine similarity with query} \\ + \beta \cdot \text{average similarity with other models}$$

- **Early Termination:** A model produces a response with a similarity score that is 0.5 or higher than the others, indicating a clear best response.

Dynamic Model Pruning: As the algorithm progresses, models that consistently underperform are naturally de-prioritized. Explicit pruning is not required, as the bandit strategy automatically minimizes allocations to poorly performing models. If a model’s response is consistently behind in terms of similarity score, the model will eventually be eliminated from further pulls.

Termination Criteria: The generation process stops when one of the following conditions is met:

- **Token budget exhausted:** The algorithm has allocated all 2048 tokens.
- **High-confidence response identified:** A model achieves a significantly higher reward than others.

At the end of execution, the model with the highest total reward is identified, and its latest response is returned as the best response.

IV. LLM-MS PROTOTYPE ARCHITECTURE

The LLM-MS system is a multi-model, multi-modal platform designed to efficiently allocate computational resources and dynamically select LLMs based on query characteristics. The architecture consists of four main components:

- **Hardware Layer:** GPU and CPU resource management.
- **Storage Layer:** Filesystem, vector database and client-side local storage.
- **Computation Layer:** Model execution and API interaction.
- **Application Layer:** Web server, and user interface.

Figure 2 illustrates the LLM-MS system architecture.

A. Hardware Layer

The foundation of LLM-MS relies on a heterogeneous hardware architecture that uses NVIDIA GPUs to execute models efficiently. Multiple GPUs are utilized for parallel processing and if no GPU resources are available, computations are handled by the CPU.

B. Storage Layer

The *LLM Filesystem* is a dedicated place in which the supported LLMs are stored in an ext4 format. For these models to function properly a shared vector database among them is mandatory. Unlike traditional databases that organize data in tables and rows, vector databases store data as multi-dimensional vectors. Data, of any type, is first transformed into an embedding, a series of numbers representing the data, which is then stored in the database as a vector. These embeddings are generated based on the context of the data, meaning that similar data have similar embeddings, which

Algorithm 2 LLM-MS-MAB

Input: Set of LLMs $LLM = \{LLM_1, LLM_2, \dots, LLM_N\}$, user prompt p , Maximum tokens λ_{\max}

Output: Best response r_p

```

1:  $\alpha \leftarrow 0.7, \beta \leftarrow 0.3$   $\triangleright$  Tuning parameters for response scoring
2: Initialize UCB1 bandit with  $N$  arms
3:  $explore\_coefficient \leftarrow 0.3$   $\triangleright$  Base exploration coefficient for UCB
4:  $embeddings \leftarrow \{\}$   $\triangleright$  Store response embeddings
5:  $responses \leftarrow \{\}$   $\triangleright$  Store partial responses per LLM
6:  $scores \leftarrow \{\}$   $\triangleright$  Store similarity scores per LLM
7:  $usedTokens \leftarrow 0$   $\triangleright$  Track cumulative tokens used
8:  $totalPulls \leftarrow 0$ 
9:  $bestResponse \leftarrow \emptyset$ 
10:  $bestScore \leftarrow 0$ 
11: while ( $usedTokens + \lambda_{pull} \leq \lambda_{\max}$ ) do
12:    $i \leftarrow \text{selectArm}()$   $\triangleright$  Choose LLM with UCB1
13:    $LLM_i \leftarrow LLM[i]$ 
14:   ( $chunk, doneReason$ )  $\leftarrow \text{pullChunk}(LLM_i, p, \lambda_{pull})$ 
15:   Append chunk to  $responses[LLM_i]$ 
16:    $usedTokens \leftarrow usedTokens + \lambda_{pull}$ 
17:    $totalPulls \leftarrow totalPulls + 1$ 
18:    $\gamma \leftarrow explore\_coefficient \cdot (1 - usedTokens/\lambda_{\max})$ 
19:    $embedding_i \leftarrow \text{vectorize}(responses[LLM_i])$ 
20:    $embeddings[LLM_i] \leftarrow embedding_i$ 
21:    $qScore \leftarrow \text{cosineSimilarity}(embedding_i, \text{vectorize}(p))$ 
22:    $interScore \leftarrow \text{InterModelSimilarity}(embedding_i, embeddings)$ 
23:    $scores[LLM_i] \leftarrow (\alpha \times qScore) + (\beta \times interScore)$ 
24:    $reward \leftarrow scores[LLM_i]$ 
25:    $\text{updateBandit}(i, reward, \gamma)$   $\triangleright$  Update UCB1 formula
26:    $bestModel \leftarrow \max(scores)$ 
27:    $secondBestScore \leftarrow \text{secondMax}(scores)$ 
28:   if  $scores[bestModel] > secondBestScore + 0.5$  then
29:     return  $\text{completeAnswer}(bestModel, responses[bestModel])$   $\triangleright$  Returning full answer
30:   end if
31: end while
32: for all  $LLM_i \in LLM$  do  $\triangleright$  Compute final scores for each LLM
33:   if  $responses[LLM_i]$  is non-empty then
34:      $embedding_i \leftarrow \text{vectorize}(responses[LLM_i])$ 
35:      $embeddings[LLM_i] \leftarrow embedding_i$ 
36:      $qScore \leftarrow \text{cosineSimilarity}(embedding_i, \text{vectorize}(p))$ 
37:      $interScore \leftarrow \text{InterModelSimilarity}(embedding_i, embeddings)$ 
38:      $scores[LLM_i] \leftarrow (\alpha \times qScore) + (\beta \times interScore)$ 
39:   else
40:      $scores[LLM_i] \leftarrow 0$ 
41:   end if
42: end for
43:  $bestModel \leftarrow \max(scores)$ 
44:  $bestResponse \leftarrow responses[bestModel]$ 
45: return  $bestResponse$ 

```

naturally forms relationships among the data. For the embeddings to be accessible to all LLMs in the system, an embedding normalizer is used to ensure consistency across all embeddings, which allows for seamless mapping of the queries to the appropriate LLMs. A handful of databases have been developed to facilitate the new needs that come with

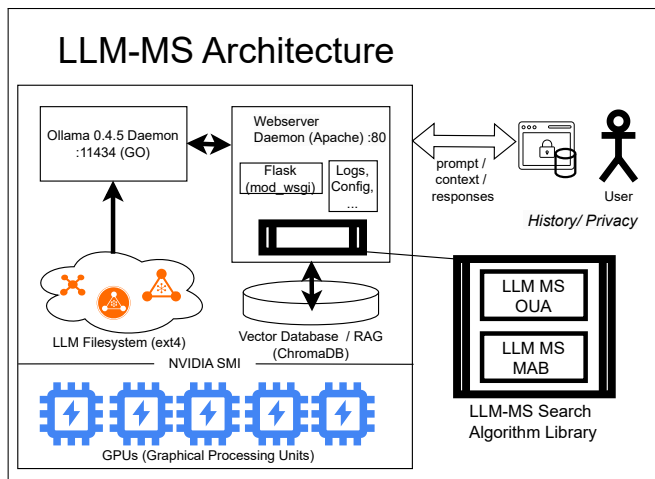


Fig. 2. The LLM-MS Architecture

LLMs, with ChromaDB and Milvus being among the most popular ones. Pan et al. [32] provide an extensive survey on vector database management systems, further analyzing their internal processes and their role in AI-driven applications.

LLM-MS uses ChromaDB for its Retrieval-Augmented Generation (RAG) pipeline and local storage for client-side session persistence. It supports hierarchical context management by summarizing large chat histories while keeping essential information. Files and queries uploaded by users are processed, vectorized and stored in ChromaDB. By using cosine similarity, a core feature of ChromaDB, we fetch the most relevant text segments that align with the query. An enhanced prompt is constructed and fed to the large language models. Thus, the large language model returns an informed answer. Additionally, our system supports hierarchical context management by summarizing extensive chat histories while preserving essential details, and it ensures session continuity through local storage.

Algorithm 3 RAG using Flask, ChromaDB, and Ollama

- 1: **Input:** Receive file F and query Q via Flask.
- 2: **Extract:** Extract text T from file F using Python libraries (supports PDF, TXT, image).
- 3: **Store:** Compute embeddings E for text T and store the pair (T, E) in ChromaDB.
- 4: **Embed Query:** Compute embeddings E_Q for query Q .
- 5: **Retrieve:** Use cosine similarity in ChromaDB to fetch the most relevant text segments S matching E_Q .
- 6: **Prompt:** Construct prompt P by combining Q with retrieved segments S .
- 7: **Infer:** Send P to the Ollama client and receive response R .
- 8: **return** R

C. Computation Layer

For model deployment and management, LLM-MS uses *Ollama 0.4.5 Daemon*, which is responsible for:

- Loading and executing models from the LLM Filesystem.
- Handling API requests for model inference.
- Distributing workload among system resources.

D. Application Layer

This layer accommodates the web-based interface and user interaction logic of LLM-MS. The Apache *Webservice* runs using a Flask back-end and is responsible for:

- Processing user queries and feeding them to the Ollama daemon.
- Retrieving and displaying responses in real-time.
- Managing session data while ensuring history summarization for long conversations.
- Hosting the algorithms described in Section III, LLM-MS-OUA and LLM-MS-MAB.

LLM-MS functions on a browser-based User Interface that allows users to interact with the system, and supports:

- Dynamic model selection.
- Real-time response streaming.
- Session history with privacy controls that is wiped once the session expires.

V. EXPERIMENTAL METHODOLOGY & EVALUATION

This section presents an experimental evaluation of our proposed approach. We start out with the experimental methodology and setup, followed by various experiments conducted that expose the core benefits of our *LLM-MS* family of algorithms.

A. Methodology

This subsection provides details on the algorithms, metrics, and datasets used for performance evaluation.

Testbed: The evaluation was conducted on a VMware private datacenter within our laboratory, particularly on an HP DL380 Gen10 with 80 logical processors and a powerful NVIDIA V100 card. The virtualized computing node consists of a Ubuntu O.S. 20.4 image equipped with 98GB of RAM, 40 virtual CPUs (@ 2.10GHz), 100GB SSD, and 1TB of NVMe memory.

Truthful QA Query Dataset [33]: Our experimental evaluation uses the TruthfulQA benchmark dataset found on Huggingface [12], which measures whether a language model is truthful in generating answers to questions. The benchmark comprises of 817 questions that span 38 categories, including health, law, finance and politics. Questions are crafted so that some humans would answer falsely due to a false belief or misconception. The datasets contains ground truth, i.e., (i) the best answer; (ii) the correct answers and; (iii) the wrong answers, to each of the 817 questions in the dataset.

LLM Datasets: We utilize the following LLM models in our evaluation:

- **Llama 3.1 8B** [34], which is a light-weight model (compared to its siblings 70B and 405B), used for higher quality inference and distillation. This dataset is 4.9 GB.
- **Mistral 0.3 7B** [6], which is a model from the Mistral family, the smallest of the family used for fine-tuning at any task.
- **Qwen2.5 7B** [35], which is the latest series of Qwen large language models, pretrained on Alibaba latest large-scale dataset.

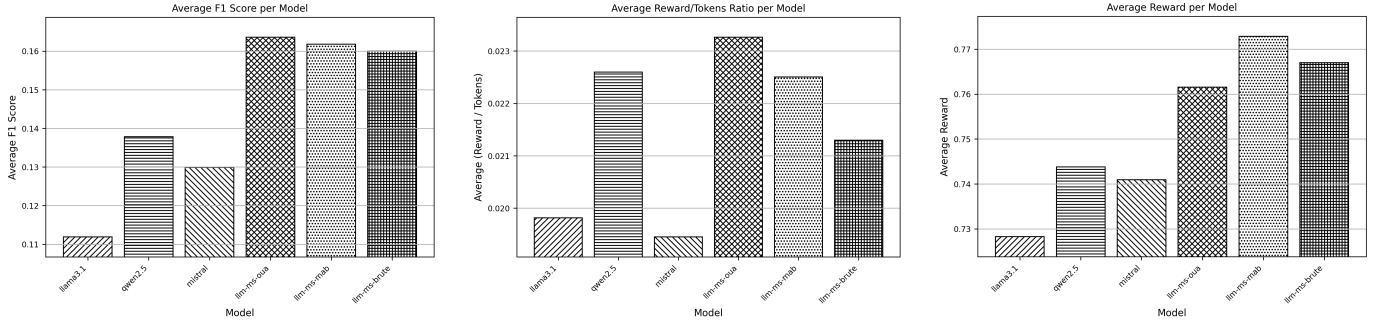


Fig. 3. **Performance Evaluation:** The left graph shows the average F1 scores per model, the middle graph depicts the average reward/tokens ratio, and the right graph shows the average reward per model.

TABLE I
SUMMARY OF RESULTS PER MODEL

Model	Best Ratio	Avg. Ratio	Avg. Reward	Best F1	Avg. F1
llama3.1	0.1260	0.0198	0.7283	0.6556	0.1119
qwen2.5	0.1329	0.0226	0.7438	0.8049	0.1378
mistral	0.1329	0.0195	0.7409	0.8049	0.1299
LLM-MS-OUA	0.1309	0.0233	0.7616	0.7230	0.1636
LLM-MS-MAB	0.0944	0.0225	0.7728	0.7582	0.1618
LLM-MS-BRUTE	0.1270	0.0210	0.7672	0.7900	0.1590

Metrics: The efficiency of the proposed technique is evaluated using two primary metrics: *Reward* and *Tokens Used*, as described in Section IV. The *Reward* is computed via a weighted scoring mechanism based on the cosine similarity between embeddings. Specifically, the reward is defined as:

$$\begin{aligned} \text{Reward} = & w_1 \cdot \text{sim}(\text{response}, \text{golden answer}) \\ & + w_2 \cdot \text{sim}(\text{response}, \text{correct answers}) \\ & - w_3 \cdot \text{sim}(\text{response}, \text{incorrect answers}) \end{aligned} \quad (1)$$

where $w_1 = 1$, $w_2 = 0.5$, and $w_3 = 0.5$, and $\text{sim}(\cdot, \cdot)$ is the cosine similarity between two embedding vectors. Additionally, the F1 score is used to assess the quality of the generated responses, while token efficiency is the total number of tokens utilized during the inference.

We have developed a set of algorithms that identify the best response, each based on a different approach.

- **LLM-MS-OUA:** With this approach, the best response is generated by progressively increasing the value of λ allocated to models, based on previous responses. The models first generate responses using a small λ , and based on the outcomes, a subset of the used models is called again with a bigger λ .
- **LLM-MS-MAB:** This approach formulates model selection as a reinforcement learning problem, leveraging the UCB1 multi-armed bandit algorithm. The models are treated as bandit arms, and the algorithm dynamically selects which model to allocate tokens to based on exploration-exploitation trade-offs. This strategy prioritizing models that demonstrate higher response quality while ensuring sufficient exploration of all candidates.

B. Performance Evaluation

We evaluated our *LLM-MS-OUA* and *LLM-MS-MAB* algorithms against three baseline models, namely **llama3.1**,

qwen2.5, and **mistral**, on the TruthfulQA benchmark dataset. We measured both *response quality* and *token efficiency*. Table I shows the main results, and the following paragraphs discuss key observations.

a) *Average Reward:* *LLM-MS-MAB* has the highest average reward (0.7728), which shows that it consistently produces responses closer to correct or best answers while avoiding incorrect ones.

b) *Average F1:* *LLM-MS-OUA* shows the highest average F1 (0.1636). Although qwen2.5 and mistral achieve higher single data-point F1 scores (up to 0.8049), *LLM-MS-OUA* maintains better overall consistency.

c) *Reward/Tokens Ratio:* *LLM-MS-OUA* has the highest average reward-to-tokens ratio (0.0233), suggesting it uses tokens efficiently. Meanwhile, *LLM-MS-MAB* has a ratio of 0.0225, which is competitive.

d) *Takeaways:*

- **LLM-MS-MAB** has the highest average reward (0.7728).
- **LLM-MS-OUA** has the highest average F1 (0.1636) and the highest average reward-to-tokens ratio (0.0233).

Overall, *LLM-MS* balances reward maximization with token usage, showing stable performance across the dataset.

VI. CONCLUSIONS AND FUTURE WORK

The rapid evolution of LLMs has revolutionized NLP, enabling new opportunities in text generation, summarization, and multi-modal understanding. However, this has introduced significant challenges in model selection, resource allocation, and computational efficiency. In this paper, we introduced *LLM-MS*, a web-based multi-model system designed to address these challenges by dynamically selecting and allocating resources to LLMs based on query characteristics.

LLM-MS leverages a three-layer architecture, with a user-friendly front-end, a robust back-end, and a vector database, to

provide real-time interaction, efficient data management, and hierarchical context management. Using advanced search and reinforcement learning algorithms, the system optimizes token allocation and response quality while maintaining a balance between computational cost and accuracy. We introduced two algorithms to guide model selection, namely: (i) **LLM-MS-OUA**, which is an iterative approach progressively allocates more tokens to high-performing models while filtering out underperformers, ensuring efficient resource distribution; and (ii) **LLM-MS-MAB**, which leverages the UCB1 algorithm to dynamically balance exploration and exploitation, prioritizing models that generate the most relevant responses while maintaining diversity in selection.

Our experimental evaluation demonstrates that both algorithms minimize token usage and response quality compared to standard single-model approaches, while maintaining or improving response quality. These results highlight LLM-MS's potential for efficient and adaptive model selection in real-time applications.

Future work includes extending LLM-MS to support additional models and refining response accuracy metrics. We also aim to investigate how LLM-MS can be combined with MoE architectures and deploying open architectures that realize our proposed ideas. By continuously improving model integration and optimization strategies, LLM-MS represents a step forward in making LLM-based systems more accessible, efficient, and adaptable for a wide range of applications.

REFERENCES

- [1] OpenAI. (2023) Chatgpt. [Online]. Available: <https://chat.openai.com>
- [2] Anthropic. (2023) Claude. [Online]. Available: <https://claude.ai>
- [3] OpenAI. (2019) Gpt-2. [Online]. Available: <https://openai.com/research/gpt-2>
- [4] EleutherAI. (2021) Gpt-neo. [Online]. Available: <https://www.eleuther.ai/artifacts/gpt-neo>
- [5] M. AI. (2023) Llama. [Online]. Available: <https://www.llama.com/>
- [6] ——. (2023) Mistral ai. [Online]. Available: <https://mistral.ai/>
- [7] G. AI. (2023) Gemini. [Online]. Available: <https://ai.google.dev/gemini-api/docs/models/gemini>
- [8] F. AI. (2023) Falcon. [Online]. Available: <https://www.falcon.ai>
- [9] DeepSeek. (2025) Deepseek. [Online]. Available: <https://www.deepseek.com/>
- [10] DeepSeek-AI, “Deepseek-v3 technical report,” 2024. [Online]. Available: <https://arxiv.org/abs/2412.19437>
- [11] Google. (2025) Gemini 2.0 flash. [Online]. Available: <https://deepmind.google/technologies/gemini/flash/>
- [12] H. Face. (2023) Hugging face. [Online]. Available: <https://huggingface.co>
- [13] D. Myers, R. Mohawesh, V. I. Chellaboina, A. L. Sathvik, P. Venkatesh, Y.-H. Ho, H. Henshaw, M. Alhawawreh, D. Berdik, and Y. Jararweh, “Foundation and large language models: fundamentals, challenges, opportunities, and social impacts,” *Cluster Computing*, vol. 27, no. 1, p. 1–26, Nov. 2023. [Online]. Available: <https://doi.org/10.1007/s10586-023-04203-7>
- [14] S. Samsi, D. Zhao, J. McDonald, B. Li, A. Michaleas, M. Jones, W. Bergeron, J. Kepner, D. Tiwari, and V. Gadepally, “From words to watts: Benchmarking the energy costs of large language model inference,” in *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, 2023, pp. 1–9.
- [15] M. AI. (2023) Mistral ai. [Online]. Available: <https://mistral.ai/news/mixtral-of-experts>
- [16] W. Fedus, B. Zoph, and N. Shazeer, “Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity,” 2022. [Online]. Available: <https://arxiv.org/abs/2101.03961>
- [17] S. Betts, “Peering inside gpt-4: Understanding its mixture of experts (moe) architecture,” 2023. [Online]. Available: <https://medium.com/@seanbetts/peering-inside-gpt-4-%E2%80%90-understanding-its-mixture-of-experts-moe-architecture-2a42eb8dbcb3>
- [18] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer,” 2017. [Online]. Available: <https://arxiv.org/abs/1701.06538>
- [19] Google. (2025) Vertex-ai. [Online]. Available: <https://openai.com/research/gpt-2>
- [20] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, “A survey of large language models,” 2024. [Online]. Available: <https://arxiv.org/abs/2303.18223>
- [21] D. Jiang, X. Ren, and B. Y. Lin, “LLM-blender: Ensembling large language models with pairwise ranking and generative fusion,” in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 14 165–14 178. [Online]. Available: <https://aclanthology.org/2023.acl-long.792/>
- [22] F. Wan, X. Huang, D. Cai, X. Quan, W. Bi, and S. Shi, “Knowledge fusion of large language models,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=jiDsk12qcz>
- [23] Q. H. Nguyen, D. C. Hoang, J. Decugis, S. Manchanda, N. V. Chawla, and K. D. Doan, “Metallm: A high-performant and cost-efficient dynamic framework for wrapping llms,” 2024. [Online]. Available: <https://arxiv.org/abs/2407.10834>
- [24] K. Lu, H. Yuan, R. Lin, J. Lin, Z. Yuan, C. Zhou, and J. Zhou, “Routing to the expert: Efficient reward-guided ensemble of large language models,” in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, K. Duh, H. Gomez, and S. Bethard, Eds. Mexico City, Mexico: Association for Computational Linguistics, Jun. 2024, pp. 1964–1974. [Online]. Available: <https://aclanthology.org/2024.naacl-long.109/>
- [25] T. Shnitzer, A. Ou, M. Silva, K. Soule, Y. Sun, J. Solomon, N. Thompson, and M. Yurochkin, “Large language model routing with benchmark datasets,” 2023. [Online]. Available: <https://arxiv.org/abs/2309.15789>
- [26] Y. Wang, Z. Yu, Z. Zeng, L. Yang, C. Wang, H. Chen, C. Jiang, R. Xie, J. Wang, X. Xie, W. Ye, S. Zhang, and Y. Zhang, “Pandalm: An automatic evaluation benchmark for llm instruction tuning optimization,” 2024. [Online]. Available: <https://arxiv.org/abs/2306.05087>
- [27] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, “Judging LLM-as-a-judge with MT-bench and chatbot arena,” in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. [Online]. Available: <https://openreview.net/forum?id=uccHPGDlao>
- [28] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, “Retrieval-augmented generation for large language models: A survey,” 2024. [Online]. Available: <https://arxiv.org/abs/2312.10997>
- [29] G. Li, X. Zhou, and X. Zhao, “Llm for data management,” *Proc. VLDB Endow.*, vol. 17, no. 12, p. 4213–4216, Aug. 2024. [Online]. Available: <https://doi.org/10.14778/3685800.3685838>
- [30] Z. Yao, Z. Tang, J. Lou, P. Shen, and W. Jia, “Velo: A vector database-assisted cloud-edge collaborative llm qos optimization framework,” in *2024 IEEE International Conference on Web Services (ICWS)*, 2024, pp. 865–876.
- [31] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.
- [32] J. J. Pan, J. Wang, and G. Li, “Survey of vector database management systems,” *The VLDB Journal*, vol. 33, no. 5, p. 1591–1615, Jul. 2024. [Online]. Available: <https://doi.org/10.1007/s00778-024-00864-x>
- [33] T. Authors, “Truthfulqa: A dataset for evaluating ai’s truthfulness,” 2025. [Online]. Available: https://huggingface.co/datasets/truthfulqa/truthful_qa/tree/main
- [34] M. AI. (2025) Llama 3.1 8b. [Online]. Available: <https://ai.meta.com/llama>
- [35] A. D. Academy. (2025) Qwen 2.5 7b. [Online]. Available: <https://qwenlm.github.io/blog/qwen2.5/>