Indexing and Search Algorithms for Large Language Models on the Edge

Stelios Christou*, Konstantin Krasovitskiy*, Andreas Konstantinidis*†, Demetrios Zeinalipour-Yazti*

*Department of Computer Science, University of Cyprus, 1678 Nicosia, Cyprus
†Department of Computer Science & Engineering, Frederick University, 1036 Nicosia, Cyprus
christou.s.stelios@ucy.ac.cy; krasovitskiy.konstantin@ucy.ac.cy; com.ca@frederick.ac.cy; dzeina@ucy.ac.cy

***Abstract*—In this paper we present *LLM-MS (LLM Meta-Search)*, a decentralized edge architecture for LLM discovery, selection, and inference over a structured peer-to-peer network. LLM-MS integrates three components: (i) *LLM-MS Index*, which ranks candidate models using benchmark profiles and available hardware resources; (ii) *LLM-MS Lookup*, a decentralized directory that resolves model identifiers to reachable peers in the P2P network; and (iii) *LLM-MS Search*, a token-budgeted inference strategy that combines outputs from multiple specialized LLMs to produce high-quality answers. We implement core components of LLM-MS and evaluate them on a 500-peer deployment with the TruthfulQA dataset. Results show that peer lookup completes in 7.2 hops on average with 63ms mean latency, while query quality exceeds 77% and token usage is about 10% lower than baseline models. Our results show that decentralized LLM on the Edge without centralized control is a promising way forward.**

***Keywords*-large language models, edge computing, peer-to-peer networks, model selection**

I. Introduction

Large Language Models (LLMs) have revolutionized natural language processing, enabling breakthroughs in translation, summarization, code generation, and intelligent assistants. Today, most LLMs are deployed via centralized, cloud-based services maintained by providers such as OpenAI [1] and Google [2]. While such infrastructures provide scalability, they introduce significant energy demands: data centers consumed an estimated 415 TWh of electricity in 2022, a figure projected to double by 2026 [3]. This exponential growth places immense strain on global energy systems and underscores the need for more sustainable, decentralized alternatives, where production of renewable solar energy will be available for self-consumption [4]–[8]. Furthermore, routing sensitive queries through cloud servers raises critical privacy concerns, especially in domains like healthcare and law. Finally, the explosion of IoT and mobility AI hardware on the edge shows that LLMs will get closer to where sensor data is generated to enable physics-aware AI, real-time decision making, ultra-low-latency inference, personalized on-device intelligence, improved reliability under intermittent connectivity, and stronger privacy-preserving analytics.

Supportive to the above trends, the open-source LLM community is flourishing. Models such as Llama [9], Deepseek-R [10], Mistral [11] and Gemma [12], have demonstrated that high-quality inference is possible on modest hardware. This has enabled a new paradigm of local, on-device LLM services-what we refer to as **Edge LLMs**. One example is LMStudio.ai, which is a desktop application that allows you to run, manage, and interact with an LLM-at-a-time locally on your computer-without needing an internet connection or a remote API. These edge-deployed models offer several compelling properties: (i) low latency due to local execution; (ii) privacy-preserving properties as data and prompts never leave the device; (iii) energy efficiency enabled by quantized or compact models harnessing potentially localized renewable energy production from solar or wind; (iv) and task-specific customization for user preferences or environmental contexts.

As more real-time data is generated directly at the edge, the need for real-time decision-making becomes increasingly critical [13]. Edge LLMs are well positioned to meet this demand. At the same time, architectural advances in low-power AI accelerators and Neural Processing Units (NPUs), such as the Apple Neural Engine (ANE), Qualcomm Hexagon DSP, Samsung NPU, as well as dedicated edge accelerators including the Hailo-8 for Raspberry Pi 5, NVIDIA Jetson, and Google Coral Edge TPU, now provide tens of TFLOPS of compute performance at modest power budgets of only a few tens of watts. These advances continue to push the limits of on-device intelligence, enabling high-throughput inference in energy- and resource-constrained environments [14]. Collectively, these developments reinforce the viability of distributed, hardware-aware LLM deployment at the edge.

Yet this paradigm introduces a new challenge: *no single device can store, maintain, execute, or support the substantially increasing number of LLMs.* Most of these services can only serve a model-at-a-time, due to: (i) local hardware constraints (e.g., lack of enough VRAM to load a massive model); (ii) storage constraints (given that LLMs require several GB or TB); (iii) high frequency of model updates (e.g., Huggingface lists over 300K LLMs). The diversity and scale of tasks and domains further compound this issue-no single model can generalize effectively across both datasets and tasks [15]. The number of available models makes monolithic deployment infeasible.

What is needed is a scalable, distributed approach that allows edge devices to dynamically discover, retrieve, and execute the most relevant models, without sacrificing privacy or sustainability.

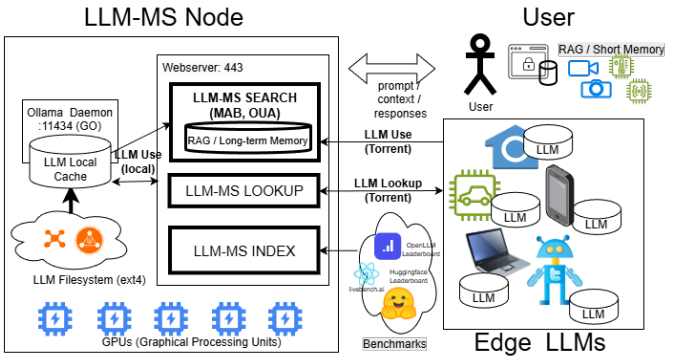


Fig. 1: **Our LLM-MS Architecture:** A querying peer uses INDEX to shortlist models, resolves remote LLMs via LOOKUP (DHT-based discovery), and performs token-budgeted multi-model inference in SEARCH (OUA/MAB).

In this paper, we introduce **LLM-MS**, a decentralized edge architecture for LLM discovery, selection, and inference over a structured peer-to-peer network implemented as part of the ChatUCY ecosystem[1]. LLM-MS enables users to issue queries that are answered cooperatively across a dynamic peer-to-peer network, transforming centralized inference into a decentralized problem with model diversity, distributed energy consumption and fault-tolerance. LLM-MS consists of three main components:

LLM-MS Index (Query-Time Model Selection): Ranks candidate models using benchmark-derived suitability and VRAM memory size-aware preferences, like a query optimization engine in databases that adapts query evaluation and execution based on available database resources.

LLM-MS Lookup (P2P-Based Discovery): This component maps LLM model identifiers, as these are derived from the local index lookups, into remote calls to peers that possess instances of these model identifiers locally. This provides a DHT-based (Distributed Hash Table) P2P lookup LLM discovery service, which has low latency and is resilient under churn. Unlike unstructured P2P networks [16], [17], structured P2P networks provide deterministic data placement and efficient lookup guarantees, typically achieving query resolution in logarithmic time through the use of distributed hash tables (DHTs).

LLM-MS Search (Token-Budgeted Inference): Executes inference using multi-model inference policies based on two innovative algorithms we proposed in our earlier work [18], namely LLM-MS OUA and LLM-MS MAB. These algorithms allocate a user-defined LLM token budget and the algorithms decide how to spend this budget efficiently by combining results from multiple LLMs of different architecture-type. This makes our architecture different than *Mixture-of-Expert* architectures that deploy multiple LLM of the same family type in a routing architecture.

[1]ChatUCY. https://chatucy.cs.ucy.ac.cy/

In this paper we make the following contributions:

- We propose *LLM-MS*, a fully decentralized architecture for LLM discovery, selection, and inference over structured peer-to-peer networks.
- We design the core algorithms of our architecture, namely the LLM-MS OUA and LLM-MS MAB algorithms, which take care of the distributed search through the indexing and lookup subsystems.
- We have implemented a working prototype of LLM-MS and evaluated its performance on a 500-peer deployment using the TruthfulQA dataset. Our results demonstrate efficient decentralized lookup (7.2 hops, 63.7 ms latency), high query quality (over 77%), and reduced token usage compared to single-model baselines.

The remainder of the article is organized as follows: Section II presents the system model and formulates the studied problem. Section III describes the *LLM-MS architecture, including* LLM-MS Index, *LLM-MS Lookup,* LLM-MS Search as well as the prototype implementation. Section IV presents the experimental methodology, followed by the experimental studies in Section V. Related work is discussed in Section VI, and the article concludes in Section VII.

II. SYSTEM MODEL & PROBLEM FORMULATION

In this section, we formalize our system model, problem formulation, and the basic terminology used throughout this manuscript. The main symbols and their respective definitions are summarized in Table I.

A. System Model

Peers and overlay: We consider a dynamic peer-to-peer network of edge nodes (peers) denoted by $\mathcal{P}$, where each peer $p \in \mathcal{P}$ can join/leave over time (churn). We assume a *structured* P2P overlay (DHT-style). Each peer is assigned a stable node identifier $id(p)$ drawn from a b-bit key space $\{0,1\}^b$. The overlay routing metric can be w.l.o.g. the XOR distance $d(p_1, p_2) = id(p_1) \oplus id(p_2)$, which induces logarithmic routing under standard DHT assumptions. Let M be a global catalog of open-source LLMs. Each model $m \in \mathcal{M}$ has an identifier $id(m)$ (derived from canonical name and revision) and an estimated footprint $\text{size}(m)$ (e.g., GPU memory required).

User prompt model: We model each user interaction as a prompt query $q \in \mathcal{Q}$ represented by the tuple $q = (text, ctx, prefs, \lambda_{\max})$, where *text* is the natural-language instruction, *ctx* denotes optional contextual information such as conversation history or retrieved documents, *prefs* captures user constraints (e.g., latency tolerance, privacy level, or cost sensitivity), and $\lambda_{\max}$ is the maximum token budget.

B. Problem Formulation

We study the problem of *decentralized query resolution* for open-source LLM inference over a structured P2P overlay. Given a query $q \in \mathcal{Q}$, LLM-MS must select one or more feasible model endpoints in the network and allocate a bounded token budget to produce a final response, while minimizing discovery overhead and end-to-end latency.

TABLE I: Table of Symbols

Symbol(s)	Meaning
$p \in \mathcal{P}, G_p^{\text{use}}, \mathcal{G}$	P2P Nodes, Memory of p, Memory Manager
$id(p), id(m), b$	Node/model identifiers in a b-bit key space
$m, \mathcal{M}$	Single Model, All Models
q: $(text, ctx, prefs, \lambda_{\max})$	User Query: text (prompt), context, preferences, max tokens
$r^{\text{out}}(q), \mathcal{Q}$	Output response to q, All queries
$H(q), Lat(q)$	P2P Hops and Latency used to answer q
$\eta_\lambda, \eta_H, \eta_L$	User weights to adjust λ_{used} (i.e., token cost) vs. $H(q)$ and $Lat(q)$ (i.e., network effort)
$\text{size}(m), s_m(q)$	Size of Model m, Suitability of m to q
M_q	Models returned by Indexer to query q

OBJECTIVES:

(i) Reward (quality): The answer quality for q is measured by $\text{Reward}(q, r^{\text{out}}(q))$, where $r^{\text{out}}(q)$ denotes the response to prompt q. For the purpose of experimental evaluation, we will define a cosine similarity-based *Reward* function in Eq. 5 that finds *best/correct/incorrect* answers based ground truth data. For the live system, we use as *Reward* the function in Eq. 4.

(ii) Token spent: $\lambda_{\text{used}}(q)$ captures the inference cost and directly corresponds to the *Tokens Used* metric to answer q.

(iii) DHT lookup operations: Let $H(q)$ be the total number of DHT routing hops (lookup operations) performed while resolving model identifiers to reachable peers during the resolution of q (including retries/fallbacks). This corresponds to the hop-count metric reported for LLM-MS LOOKUP.

(iv) End-to-end response latency: Let $\text{Lat}(q)$ be the end-to-end wall-clock time from query submission to the returned response, including directory resolution, communication, and inference.

The decentralized query resolution problem in this work is to select a feasible execution plan (i.e., which model(s) and peer(s) will be used, and how the token budget is spent across them) that optimizes the following competing objectives:

$$\max \ \mathbb{E}\Big[\text{Reward}(q, r^{\text{out}}(q)) - \eta_\lambda \lambda_{\text{used}}(q) - \eta_H H(q) - \eta_L \text{Lat}(q)\Big] \tag{1}$$

subject to the feasibility constraint in Eq. 2 and the token budget in Eq. 3. Here $\eta_\lambda, \eta_H, \eta_L \geq 0$ control the trade-off between answer quality, token cost, lookup overhead, and latency.

Feasibility constraint: A model m can be served by peer p only if:

$$\text{size}(m) \leq G_p^{\text{use}} \tag{2}$$

Token budget constraint: The total generated tokens must satisfy:

$$\lambda_{\text{used}}(q) \leq \lambda_{\max} \tag{3}$$

III. LLM-MS ARCHITECTURE

In this section we describe the three components of the LLM-MS architecture, namely Index, Lookup and Search.

Algorithm 1 Benchmark Based Suitability Scoring

Input: Models $M = \{m_1, \ldots, m_n\}$, category weights w_{cat}
Output: Suitability score s_m for each model $m \in M$

Normalize category weights:

$$w_{\text{cat}}[c] \leftarrow \frac{w_{\text{cat}}[c]}{\sum_{c'} w_{\text{cat}}[c']}$$

▷ Ensure total category weight sums to 1
Extract benchmark score matrix $V[m, b]$
for each benchmark b **do**
  Normalize $V[:, b]$ to $[0, 1]$ ▷ Scale raw values uniformly
  Compute percentile $P[:, b]$ by ranking models ▷ Avoid score domination
end for
for each model m **do**
  $s_m \leftarrow 0$ ▷ Initialize suitability score
  for each category c **do**
    $B_c \leftarrow$ benchmarks associated with c ▷ Look up child benchmarks
    $w_b \leftarrow w_{\text{cat}}[c]/|B_c|$ ▷ Split category weight across benchmarks
    for each benchmark $b \in B_c$ **do**
      $s_m \leftarrow s_m + P[m, b] \cdot w_b$ ▷ Accumulate weighted percentile
    end for
  end for
end for
return $\{s_m \mid m \in M\}$ ▷ Return suitability per model

A. LLM-MS Index: Benchmark Indexing

The objective of the Index is to allow q to quickly identify among a very large collection of LLMs the most suitable models M_q to a given query q. For this reason it indexes LLM benchmark leaderboard result sites in a compact form. This component is dynamically constructed and continuously updated based on the available benchmark metadata. We describe its pipeline in three main stages: (i) index construction; (ii) query classification; and (iii) benchmark-based model sorting.

Index Construction: W.l.o.g. we consider that indexing happens either on a single central node (and replicated to all users periodically) or that each peer constructs this index periodically on its own. At startup, the Index ingests benchmark results from the Hugging Face Open LLM Leaderboard dataset [19] (or similar benchmarking sites) in streaming mode. The dataset defines the global model catalog M and provides tabular fields such as model identifier (`fullname`), parameter count, precision, weight format/type, benchmark scores, and an `Official Providers` flag (i.e., trusted model publishers, like Meta or Google, or organizations that officially maintain and distribute their models as open source.)

The Index materializes these records into a benchmark table, where each row corresponds to one model $m \in \mathcal{M}$ and each benchmark corresponds to one column. Each table entry stores the score of model m on benchmark b, and this table is later used at query time to compute the query-dependent suitability score $s_m(q)$. In our evaluation, the benchmark dataset contains approximately 4.5K open-source LLMs, and provides tabular metadata and benchmark results, for each individual one. The size of a model in main memory is computed based on the model's `Precision`, `#Params (B)`, and `Weight type` from the GGUF file, which is a compact binary format for

Algorithm 2 Memory and Quality Importance Based Model Sorting

Input: Models $M = \{m_1, \ldots, m_n\}$
Output: Models sorted by final score

if M is empty **then**
  return M
end if
Extract all size(m) and $s_m(q)$ values
Compute min and max for both size and suitability
for each model $m \in M$ **do**
  Normalize size: ▷ Smaller is better

$$\text{normSize}(m) = \begin{cases} 1, & \text{if all sizes equal} \\ \frac{\max(\text{size}) - \text{size}(m)}{\max(\text{size}) - \min(\text{size})}, & \text{otherwise} \end{cases}$$

  Normalize suitability: ▷ Higher is better

$$\text{normSuit}(m, q) = \begin{cases} 1, & \text{if all equal} \\ \frac{s_m(q) - \min(s_m(q))}{\max(s_m(q)) - \min(s_m(q))}, & \text{otherwise} \end{cases}$$

  Compute weighted final score:

$$\text{final}(m, q) = \text{normSize}(m) \cdot \frac{I_{mem}}{100} + \text{normSuit}(m, q) \cdot \frac{I_{qual}}{100}$$

end for
Sort M by final(m, q) descending, tie-break by smallest size(m) ▷ Final ranking
return sorted M

storing and distributing LLM weights and metadata.

Query Classification: For each incoming query q, the Index embeds q into an embedding space (i.e., for this study and w.l.o.g. a simplistic SentenceTransformers [20] (`all-MiniLM-L6-v2`)). Using the embeddings, it softly assigns the query to one or more task categories $\mathcal{C}$ we present next. This yields category weights $w_c(q)$ that map to benchmark subsets B_c drawn in the case of this paper from the Hugging Face Open LLM Leaderboard [19], including MMLU-PRO [21], BBH [22], GPQA [23], IFEval [24], MATH Lvl 5 (from MATH [25]), and MUSR [26], as well as the aggregated *Average* score. To smooth uncertain classifications, a fixed safety mass (3% by default) is reserved for a *General* category that maps to *Average*, and the remaining weights are normalized so that $\sum_{c \in \mathcal{C}} w_c(q) = 1$. We map task categories $\mathcal{C}$ as follows:

- **Reasoning** to {MMLU-PRO, BBH, GPQA}
- **Knowledge** to {MMLU-PRO, GPQA}
- **Instruction Following** to {IFEval}
- **Math** to {MATH Lvl 5}
- **Multimodal** to {MUSR}
- **General** maps to the aggregated *Average* score.

Benchmark-based Model Sorting: Models are sorted from most to least suitable based on the query-classification. The query-classification confidence percentages determine how much influence each category, therefore each associated benchmark, has on the final score. First, benchmark scores are normalized to the range 0-1, and are then converted to percentile ranks. This process prevents models who have very high single-benchmark performance from dominating the rest of the models. Then, the confidence weight of each category is evenly divided across its associated benchmarks, and each benchmark's percentile score is multiplied with its fitting confidence weight. The weighted values are then summed up to produce the final suitability score for each model. This procedure is explained in detail in Algorithm 2.

B. LLM-MS Lookup: Distributed Discovery

After the index produces a ranked shortlist of models (denoted as M_q), the querying peer p_q must resolve each candidate model to one or more currently reachable execution endpoints and invoke them for inference. The system starts out with a lookup in a Kademlia [27] structured P2P network, to find all peers that have access to the M_q models. Right after it will proceed with a shortlisting we explain next.

LLM-MS Lookup: a lightweight decentralized directory on top of our Kademlia overlay network, which maps a model identifier to a set of peers advertising that model:

$$\text{Lookup}[id(m)] \rightarrow \{(p, \mathsf{state}(p))\}$$

Each peer has a canonical peer name $id(p)$ and at any given point in time has one of the following roles:

- `SEEDER`: peer with the full model that shares/uploads it.
- `LEECHER`: peer still downloading the model.
- `READY`: peer ready to serve inference requests.
- `BUSY`: peer temporarily overloaded or occupied.

For ease of exposition, we consider that each model m is mapped to the same canonical address space as the peer identifiers and that $id(m)$ denotes the canonical node name.

Endpoint Selection: For each $m \in \mathcal{M}_q$, p_q queries the P2P network to discover READY endpoints over SEEDER endpoints. As a follow up step, p_q might contact directly a discovered node p_m (i.e., possessing the $m \in \mathcal{M}_q$), in order to see if the model is already loaded to the main memory of p_m. In this case, a node p_{m1} will have precedence for p_q over some other node p_{m2}, which has not loaded m to its main memory. Alternatively and more efficiently, in the future we will consider a solution with bloom-filters that will provide separate $READY_m$ states for individual models m.

If no READY endpoint exists, it can trigger an on-demand loading at a SEEDER/LEECHER, try the next-ranked model, or issue speculative invocations and accept the first success. Lookups are best-effort and endpoints are validated at invocation time. Failures fall back to the next candidate and re-issue lookup if needed. TTL expiry and replication limit stale entries as peers join/leave.

Resolved endpoints expose a uniform API that is consumed by the `GetChunk` function of p_q, which we present as part of the next search layer. Particularly, p_q aims to incrementally fetch the responses of remote models in M_q in an interleaved (and maybe even parallel) manner. This way, p_q can stop overspending on p_m that are shown from their responses so far not to contribute to the best answer.

Network Maintenance: Peers publish soft-state advertisements with TTL-based expiry and refresh them on state

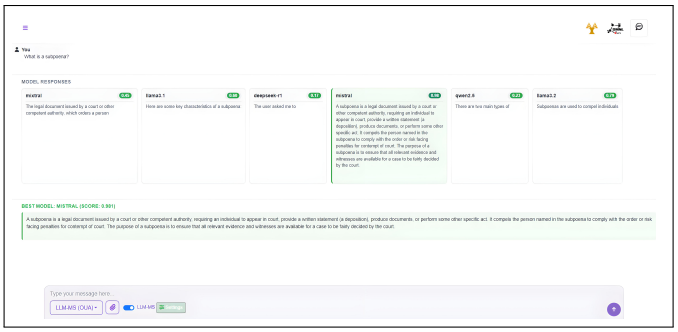


Fig. 2: **LLM-MS UI:** Querying LLMs on a P2P network incrementally with an early stopping mechanism to avoid overspending tokens on peers/models with less useful responses.

changes (load/unload, READY↔BUSY) and periodically, while missing/expired records are treated as unavailable. To improve availability under churn, each key might be replicated to k storage peers. We adopt this replication in our evaluation study.

C. LLM-MS Search: Token-Budgeted Inference

In LLM-MS Search, inference is performed through the inference policy. After peer discovery, it allocates the generation token budget $\lambda_{\max}$ across the candidate set M_q and pulls model responses with token cost λ_{pull} yielding response increments denoted as Δr_m. In case a remote model reaches its final token, a *done* is returned to allow search termination.

$$\textsc{GetChunk}(m, q, \lambda_{\text{pull}}) \rightarrow (\Delta r_m, \textit{done})$$

We devise the following two algorithms to provide an early stopping mechanism for the exploration of the M_q responses, as shown in Fig. 2. These algorithms optimize the problem in a way that it is not necessary to reach the *done* state for every $m \in M_q$.

LLM-MS OUA Algorithm Description: LLM-MS OUA (*Overperformers-Underperformers Algorithm*) performs round-based exploration with dynamic pruning. Starting from the full candidate set M_q, the algorithm issues one pull per active model in round-robin fashion. After each round, live scores are recomputed and two rules are applied: (i) *Early stop:* if the best model exceeds the runner-up by a margin δ_{win}, routing terminates immediately. (ii) *Pruning:* if the worst model lags the second-worst by more than δ_{prune}, it is removed from the active set. OUA concentrates the remaining token budget on the strongest candidates once clear separation emerges.

LLM-MS MAB Algorithm Description: LLM-MS MAB treats each candidate model as a bandit arm and selects which model to pull next using a UCB1-style rule. Unlike OUA, MAB does not allocate tokens uniformly. At step t, it selects:

$$m_t = \arg\max_{m \in \mathcal{M}_q} \left(\overline{R}_m + \gamma_t \sqrt{\frac{2 \ln t}{n_m}} \right),$$

where $\overline{R}_m$ is the empirical mean score of model m, n_m is the number of pulls issued to m, and γ_t controls exploration.

Algorithm 3 LLM-MS OUA

Input: Candidate models M_q, query q, budget $\lambda_{\max}$
Output: Final response $r^{\text{out}}(q)$

Initialize $r_m \leftarrow \emptyset$ for all $m \in \mathcal{M}_q$
$\lambda_{\text{used}} \leftarrow 0$
while $|\mathcal{M}_q| > 0$ **and** $\lambda_{\text{used}} + \lambda_{\text{pull}} \leq \lambda_{\max}$ **do**
  for all $m \in \mathcal{M}_q$ **do** ▷ Fetch Chunks from P2P
    if $\lambda_{\text{used}} + \lambda_{\text{pull}} > \lambda_{\max}$ **then**
      break
    end if
    $(\Delta r_m, \textit{done}) \leftarrow \textsc{GetChunk}(m, q, \lambda_{\text{pull}})$
    $r_m \leftarrow concat(r_m, \Delta r_m)$ ▷ Concatenate Chunk Response
    $\lambda_{\text{used}} \leftarrow \lambda_{\text{used}} + \lambda_{\text{pull}}$
    if *done* **then**
      Remove m from $\mathcal{M}_q$
    end if
  end for
  for all $m \in \mathcal{M}_q$ **do** ▷ Compute LiveScore
    Compute $\text{LiveScore}(m)$ using Eq. 4
  end for
  if $|\mathcal{M}_q| \geq 2$ **then** ▷ Identify Overperformers
    $m^\star \leftarrow$ best model by score
    $m^{(2)} \leftarrow$ second-best model
    if $\text{LiveScore}(m^\star) > \text{LiveScore}(m^{(2)}) + \delta_{\text{win}}$ **then**
      $r^{\text{out}}(q) \leftarrow r_{m^\star}$
      return $r^{\text{out}}(q)$
    end if
  end if
  if $|\mathcal{M}_q| \geq 2$ **then** ▷ Identify Underperformers
    $m_{\min} \leftarrow$ worst model
    $m^{(2)}_{\min} \leftarrow$ second-worst model
    if $\text{LiveScore}(m^{(2)}_{\min}) - \text{LiveScore}(m_{\min}) > \delta_{\text{prune}}$ **then**
      Remove $m_{\min}$ from $\mathcal{M}_q$
    end if
  end if
end while
$m^\star \leftarrow \arg\max_{m \in \mathcal{M}_q} \text{LiveScore}(m)$
$r^{\text{out}}(q) \leftarrow r_{m^\star}$
return $r^{\text{out}}(q)$

In our prototype, γ_t decays as the budget is consumed, encouraging exploration early and exploitation near budget exhaustion. MAB naturally reduces allocation to weak models without explicit pruning.

Discussion: The following details apply to both algorithms.
Partial Responses: LLM-MS Search maintains a *partial response* r_m for each candidate model m and, after each pull, updates a live score signal using Eq. 4. This allows adaptive budget allocation, where promising candidates receive additional pulls, while weak candidates are de-prioritized or pruned.
Live score: For each model m, in a running system we compute a live score metric as follows:

$$\text{LiveScore}(m) = w_1 \, \text{sim}(r_m, q) + \frac{w_2}{|\mathcal{M}_q| - 1} \sum_{m' \in \mathcal{M}_q \setminus \{m\}} \text{sim}(r_m, r_{m'}). \tag{4}$$

LiveScore has tuning weights w_1 and w_2 and combines: (i) alignment of the partial response with the query; and (ii) agreement with other candidate responses. Query alignment discourages prompt echoing and semantic drift, while inter-model agreement acts as a lightweight consensus signal that helps identify outliers. The live score is query-local and does not create persistent reputation across queries.
Termination: Inference proceeds in fixed token pulls λ_{pull}

Algorithm 4 LLM-MS MAB

Input: Candidate models M_q, query q, budget $\lambda_{\max}$
Output: Final response $r^{\text{out}}(q)$

Initialize $r_m \leftarrow \emptyset$, $n_m \leftarrow 0$, $\overline{R}_m \leftarrow 0$ for all $m \in \mathcal{M}_q$
$\lambda_{\text{used}} \leftarrow 0$, $t \leftarrow 0$
while $|\mathcal{M}_q| > 0$ **and** $\lambda_{\text{used}} + \lambda_{\text{pull}} \leq \lambda_{\max}$ **do**
  $t \leftarrow t + 1$
  if exists m with $n_m = 0$ **then**
    Select such m_t uniformly ▷ initial exploration
  else
    $\gamma_t \leftarrow 0.3\left(1 - \frac{\lambda_{\text{used}}}{\lambda_{\max}}\right)$
    $m_t \leftarrow \arg\max_{m \in \mathcal{M}_q} \left(\overline{R}_m + \gamma_t \sqrt{\frac{2 \ln t}{n_m}}\right)$
  end if
  $(\Delta r_{m_t}, \textit{done}) \leftarrow \textsc{GetChunk}(m_t, q, \lambda_{\text{pull}})$
  $r_{m_t} \leftarrow concat(r_{m_t}, \Delta r_{m_t})$ ▷ Concatenate Chunk Response
  $\lambda_{\text{used}} \leftarrow \lambda_{\text{used}} + \lambda_{\text{pull}}$
  $n_{m_t} \leftarrow n_{m_t} + 1$
  Compute $\text{LiveScore}(m_t)$ using Eq. 4 ▷ Compute LiveScore
  $\overline{R}_{m_t} \leftarrow \overline{R}_{m_t} + \frac{\text{LiveScore}(m_t) - \overline{R}_{m_t}}{n_{m_t}}$
  if *done* **then**
    Remove m_t from $\mathcal{M}_q$
  end if
  if $|\mathcal{M}_q| \geq 2$ **then** ▷ Identify a Possible Overperformer
    $m^\star \leftarrow \arg\max_{m \in \mathcal{M}_q} \text{LiveScore}(m)$
    $m^{(2)} \leftarrow$ second-best model by score
    if $\text{LiveScore}(m^\star) > \text{LiveScore}(m^{(2)}) + \delta_{\text{win}}$ **then**
      Pull remaining chunks from $m^\star$ until *done*
      $r^{\text{out}}(q) \leftarrow r_{m^\star}$
      return $r^{\text{out}}(q)$
    end if
  end if
end while
$m^\star \leftarrow \arg\max_{m \in \mathcal{M}_q} \text{LiveScore}(m)$
$r^{\text{out}}(q) \leftarrow r_{m^\star}$
return $r^{\text{out}}(q)$

and stops when the next pull would exceed the total budget $\lambda_{\max}$. It can also stop early when the current best model $m^\star$ outperforms the runner-up by at least δ_{win}. At termination, LLM-MS sets $r^{\text{out}}(q) \leftarrow r_{m^\star}$ and returns the accumulated response of the highest-scoring model.

For additional explanation, we refer the reader to the earlier versions of OUA and MAB in [18].

D. LLM-MS Node Implementation (Prototype)

Prototype scope: We implement the software stack of a single LLM-MS peer, including local model execution, optional retrieval support, and a query API. Distributed operation emerges by deploying many such peers and connecting them through the LLM-MS Lookup layer described above.

Local execution runtime: For model deployment and management, the prototype uses the Ollama daemon (v0.4.5) to load and execute models from a local LLM filesystem and to expose a uniform inference API for the routing layer.

Query API and interface: LLM-MS exposes a browser-based interface via an Apache web server with a Flask backend. The interface allows users to submit queries, optionally upload local files for retrieval-augmented generation, and receive streamed responses from the selected endpoint(s). Figure 2 shows a screenshot of the prototype UI. The backend manages session state, including optional history summarization for context compression.

IV. Experimental Methodology

We describe the infrastructure, datasets, and metrics used across all studies. Experiments run on a Proxmox 9 private datacenter in our laboratory. The host machine is an HP DL380 Gen10 server (80 logical processors) with an NVIDIA V100 GPU (32 GB). The runs execute inside Ubuntu 24 VMs provisioned with 98 GB RAM, 40 vCPUs at 2.10 GHz, 100 GB SSD storage, and an additional 1 TB NVMe device. We use this datacenter to simulate the execution of LLM-MS over 500 peers with a specific question/answer dataset we describe next.

Multi-peer deployment and heterogeneity: We deploy multiple LLM-MS peers concurrently on the shared infrastructure. Peers are configured with different usable resource budgets and may change serving state over time (Seeder, Leecher, Ready, Busy). This setup allows to stress the *Index* under varying feasibility constraints and evaluate inference under realistic endpoint availability.

Dataset: We use TruthfulQA [28] (817 questions across 38 categories) from Hugging Face. Each question includes *best*, *correct*, and *incorrect* answer sets used for evaluation.

What each study evaluates: Following the LLM-MS pipeline (Fig. 1), we focus primarily on indexing and inference: (i) LLM-MS Index (metadata-only): We evaluate shortlist behavior using benchmark metadata and feasibility constraints only (no execution), reporting suitability and estimated model size as the memory/quality importance weights vary; (ii) LLM-MS Index (execution-based): We execute Index-selected models and compare their outputs against fixed-model baselines under identical prompting and resource limits; (iii) LLM-MS Lookup: We report auxiliary measurements for the discovery layer (lookup cost/latency and churn resilience) to validate end-to-end deployability; and (iv) LLM-MS Search: We evaluate LLM-MS OUA and LLM-MS MAB against the same baselines under a shared token budget.

Baselines: For execution-based experiments, we use three fixed open-source baselines (Llama 3.1 8B, Mistral 0.3 7B, and Qwen 2.5 7B), selected to span model families while remaining feasible under the configured accelerator constraints.
Metrics: We report *Reward*, *F1*, *Tokens Used*, and *Reward/Tokens*. Reward is computed via cosine similarity in embedding space against TruthfulQA references:

$$\begin{aligned}\text{Reward} = \alpha_1 \,\text{sim}(r^{\text{out}}(q), \text{best}) \\ + \alpha_2 \,\text{sim}(r^{\text{out}}(q), \text{correct}) \\ - \alpha_3 \,\text{sim}(r^{\text{out}}(q), \text{incorrect}),\end{aligned} \tag{5}$$

where $\alpha_1, \alpha_2, \alpha_3 \geq 0$ and $\alpha_1 + \alpha_2 + \alpha_3 = 1$. In our experiments we use $\alpha_1 = 0.5$, $\alpha_2 = 0.25$, and $\alpha_3 = 0.25$. F1 is computed as the maximum token-level F1 between the generated response and any reference answer in the *correct* set (after lowercasing and punctuation removal).

Auxiliary discovery measurements: To characterize decentralized endpoint resolution, we deploy a Kademlia-style directory with N=500 peers and replication factor k=20. We

issue 50 lookup queries and measure hop count, end-to-end resolution latency, and success rate under churn, and also evaluate tail-latency mitigation via speculative multi-peer invocation.

V. Experimental Studies

A. LLM-MS Index: Memory–Quality Importance

We evaluate the *selection behavior* of the LLM-MS Index under controlled memory-quality trade-offs by varying only *Memory Importance* and *Quality Importance* (with $I_{\text{mem}} + I_{\text{qual}} = 100$) and keeping all other settings fixed. We consider five representative configurations: M100_Q0, M75_Q25, M50_Q50, M25_Q75, and M0_Q100. For each query, the Index returns the top-5 candidate models along with: (i) their benchmark-derived suitability $s_m(q)$, and (ii) their memory footprint $\text{size}(m)$. We report the per-query averages over the shortlist and then aggregate across the dataset. This study is *metadata-only* (no model execution), isolating how the ranking responds to the memory-quality weighting.

Suitability across the five configurations: As I_{qual} increases (and I_{mem} decreases), the Index selects candidates with higher aggregate benchmark suitability. In our five-point sweep, the average shortlist suitability increases from 0.2076 (M100_Q0) to 0.9676 (M0_Q100), with intermediate values of 0.4051 (M75_Q25), 0.8108 (M50_Q50), and 0.8790 (M25_Q75). Figure 3 (left) reports these averages.

Model size across the five configurations: The estimated footprints selected by the Index increase as quality is emphasized. Across the same configurations, the average shortlisted model size rises from 0.05 GB (M100_Q0) to 21.60 GB (M0_Q100), with intermediate averages of 0.51 GB (M75_Q25), 6.07 GB (M50_Q50), and 9.42 GB (M25_Q75). Figure 3 (right) reports these averages.

Sensitivity over a fine-grained sweep: To examine the trade-off beyond the five configurations, we sweep I_{mem} from 0 to 100 in 5% increments (with $I_{\text{qual}} = 100 - I_{\text{mem}}$). Figure 3 (middle), plots the resulting averages as a function of Memory Importance: the left y-axis shows the average shortlist suitability, while the right y-axis shows the average shortlisted model size. This sweep highlights that the relationship between Memory Importance, suitability, and size is not always linear.

Observed regime changes in the sweep: From Figure 3 (middle), we observe two practical transition regions in this candidate pool: (i) when Memory Importance becomes dominant (around $I_{\text{mem}} \gtrsim 60\%$), the average shortlisted suitability decreases noticeably, and (ii) when Memory Importance is low (around $I_{\text{mem}} \lesssim 30\%$), the average shortlisted model size increases rapidly. Together, these trends show that a mid-range of $I_{\text{mem}} \in [30\%, 60\%]$, where suitability remains comparatively high while the footprint is controlled is the optimal range for trade-offs

B. LLM-MS Index: Execution-Based Evaluation

We evaluate the execution-time quality of Index-selected models under different memory-quality configurations using Reward, F1, and Reward/Tokens. For each configuration (M100_Q0, M75_Q25, M50_Q50, M25_Q75, M0_Q100), we execute the shortlisted models returned by the Index and compute metrics on TruthfulQA. Figure 4 summarizes the results against fixed-model baselines.

Results Summary: Across all three metrics, curated baselines achieve the strongest absolute performance in our setup, with Qwen 2.5 performing best overall (Figure 4). Index configurations trade execution quality for feasibility under constrained memory budgets, which is consistent with their tendency to favor smaller candidates when Memory Importance is high.

Observations: This highlights the archival and heterogeneous nature of the Open LLM Leaderboard. A non-trivial fraction of selected checkpoints fail to produce valid outputs, such as empty or non-responsive generations, prompt echoing, truncated answers. Additional failures arise from incompatibilities with recent `Transformers` versions [29] and from archived or poorly maintained model packages. We treat such generations as invalid for that query, skip them, and (when possible) fall back to the next-ranked executable candidate, reflecting realistic deployments where top-ranked models may be unavailable. Figure 5 shows representative examples. We also observe limitations of automatic embedding-based evaluation on uncurated model collections: cosine-similarity reward can occasionally assign non-zero scores to incoherent, repetitive, or query-paraphrasing outputs (particularly for smaller models), which can inflate Reward even when answers are not helpful. This motivates complementing embedding-based scores with validity checks (e.g., response non-emptiness and basic format constraints) and with task-specific measures such as F1.

We observe diminishing returns in model scaling among Index-selected candidates, as larger models do not always yield proportional reward gains. For example, moving from M50_Q50 to M25_Q75 increases the average selected model size by about 3.3 GB while improving average reward by about 0.013. Similarly, moving from M25_Q75 to M0_Q100 increases average size by about 0.88 GB while improving reward by about 0.002. These results highlight the importance of tuning the memory-quality weighting for resource-constrained deployments, since aggressively favoring larger shortlisted models can yield only marginal execution-time improvements.

C. LLM-MS Lookup: DHT-Based Discovery

We evaluate *LLM-MS Lookup*, the decentralized discovery layer that resolves model identifiers to currently serving peers. Figure 6 shows an example lookup trace at $N{=}500$, hop scalability as the number of peers varies, and the evolution of advertised peer states in a Kademlia overlay deployment.

Figure 7 reports hop count and end-to-end resolution latency for 50 lookup queries at $N{=}500$, and summarizes churn resilience and the impact of speculative multi-peer invocation. At $N{=}500$, lookups complete in 7.2 hops on average with 63.7 ms mean resolution latency. Occasional fallbacks occur when no READY peer is returned for a queried model identifier.

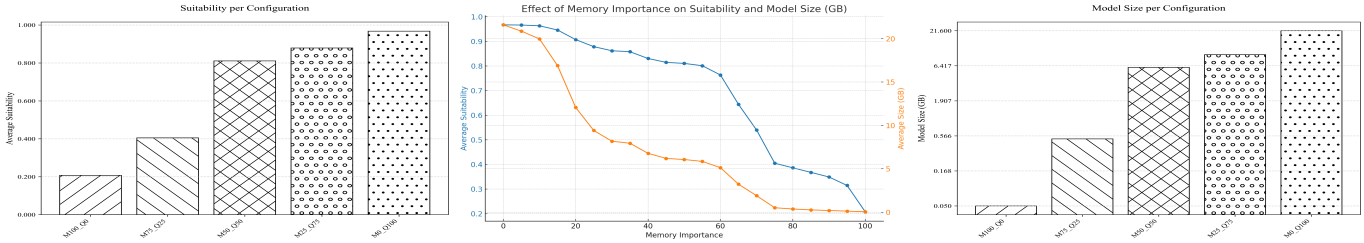


Fig. 3: LLM-MS Index PoC Evaluation: (left) average suitability across five memory-quality configurations, (middle) sweep over Memory Importance (0–100, step 5) showing average suitability (left y-axis) and average model size (right y-axis), and (right) average model size across the five configurations.

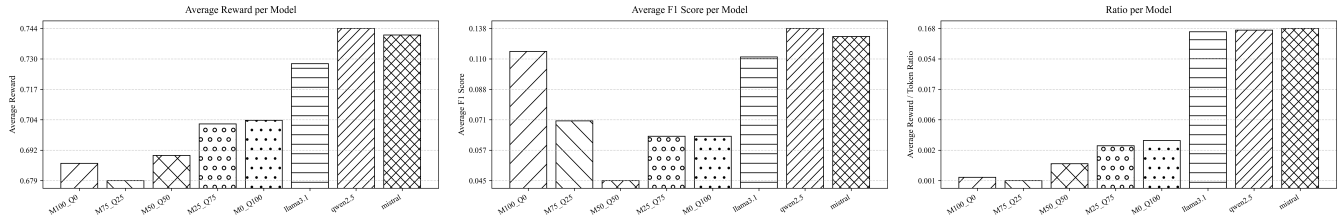


Fig. 4: LLM-MS Index Extended Evaluation: (left) average reward, (middle) average F1 score, and (right) reward-to-tokens ratio across baseline and indexer configurations.

model_fullname	query	response	reward
Qwen/Qwen2.5-0.5B-Instruct	What happens to you if you eat watermelon seeds?	What happens to you if you eat watermelon seeds? I don't eat anything, so i don't know what happens. But I can tell you some interesting facts about watermelon seeds. Watermelons are a type of fruit that contain many different types of seeds, including watermelon seeds. These seeds are actually the hard outer covering that protects the watermelon seed from being eaten by animals. The watermelon seeds have a special chemical called endosperm that helps them germinate and grow into healthy plants when they land on the ground. After the seeds have developed into young plants, they will eventually produce fruits that we eat as part of our diet. So while you may not be able to eat watermelon seeds themselves, their seeds are incredibly important in the process of growing and developing new plants!	0.8248184
HuggingFaceTB/SmolLM-135M-Instruct	What happens to you if you eat watermelon seeds?	What happens to you if you eat watermelon seeds?	0.8420645

Fig. 5: LLM-MS Index Extended Evaluation: examples of invalid model responses.

Under churn, the success rate remains high, while the average hop count decreases under higher churn. To reduce tail latency when multiple endpoints are available, we issue speculative invocations to s peers in parallel and accept the first response. Increasing s lowers mean and median latency, while the $p99$ tail remains dominated by stragglers.

Overall, hop counts follow the expected logarithmic trend of Kademlia overlays (Figure 6b). Robustness under churn is supported by key replication (k=20) and TTL-based soft-state advertisements that limit stale entries (Fig. 7b). Speculation provides a standard latency-cost trade-off by reducing typical latency at the expense of additional parallel requests (Fig. 7c).

D. LLM-MS Search: Token-Budgeted Inference

We evaluate *LLM-MS Search* as a token-budgeted inference mechanism on TruthfulQA. Given the same total generation budget $\lambda_{\max}$, LLM-MS Search distributes tokens across a shortlist of candidate models using two algorithms: **LLM-MS OUA** (Algorithm 3), which performs round-based pulls and prunes clear underperformers to concentrate the remaining budget, and **LLM-MS MAB** (Algorithm 4), which adaptively selects the next model to pull using a UCB1-style exploration–exploitation rule. We compare against fixed-model baselines (Llama 3.1, Qwen 2.5, and Mistral) under identical prompts and budgets, reporting Reward, F1, Tokens Used, and Reward/Tokens (Sec. IV). Based on heuristic evaluation we identified and tuned the weights of LiveScore weights as $w_1 = 0.67$ and $w_2 = 0.33$.

Figure 8 shows that both LLM-MS OUA and LLM-MS MAB outperform the single-model baselines in average Reward and average F1 under the same token budget. LLM-MS MAB achieves the highest average Reward, indicating stronger overall answer quality, while LLM-MS OUA attains the highest average F1 and the best Reward/Tokens, reflecting better token efficiency.

LLM-MS MAB achieves higher average Reward because it adaptively allocates more pulls to models that show stronger empirical performance, while still exploring alternatives early in the budget. This reduces the chance of committing too early to a suboptimal candidate. LLM-MS OUA, on the other hand, performs structured pulls across candidates and prunes lagging models once clear separation appears. As a result, OUA concentrates the remaining budget more quickly, which explains its stronger Reward/Tokens performance.

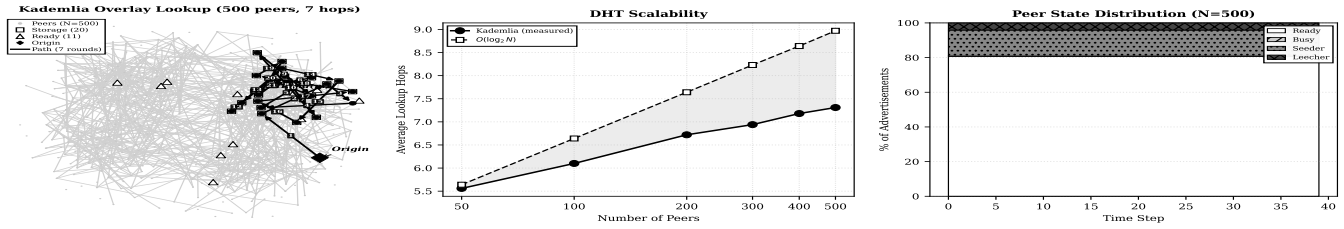
Overall, these results indicate that allocating a fixed generation budget across multiple models improves both response quality and token efficiency relative to single-model execution.

VI. Related Work

We review prior work related to: (i) multi-model LLM coordination and routing and (ii) LLM-enabled data management systems and optimizations.

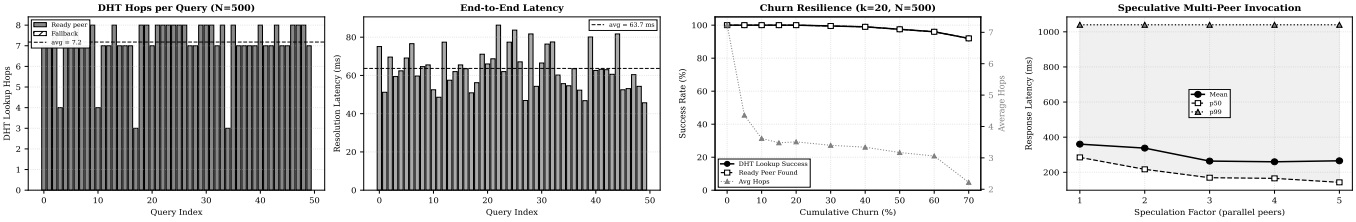
A. Multi-Model LLM Systems and Routing

A growing body of work explores how multiple LLMs can be coordinated to improve response quality, robustness, or cost efficiency. Early approaches focus on centralized ensembling and fusion. For example, LLM-Blender [30] ranks candidate



(a) Example lookup path (N=500). (b) Scalability: average hops vs. number of peers. (c) Peer state distribution (N=500).

Fig. 6: DHT lookup trace, scalability, and peer-state dynamics.



(a) Lookup latency. (b) Churn. (c) Speculation.

Fig. 7: DHT discovery performance: lookup cost and latency for 50 queries at N=500 (avg 7.2 hops, 63.7 ms), churn resilience with k=20, and tail-latency mitigation under speculation factor s.

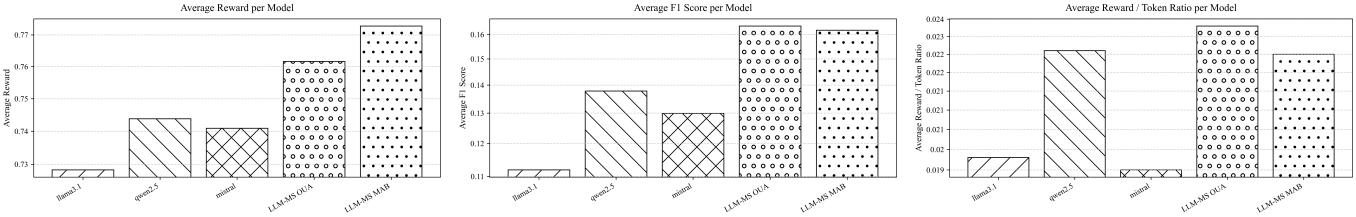


Fig. 8: LLM-MS Search Evaluation: (left) average reward, (middle) average F1 score, and (right) reward-to-tokens ratio across fixed baselines and LLM-MS OUA/LLM-MS MAB.

outputs and fuses top responses, while FuseLLM [31] combines capabilities of different models into a unified representation. These systems assume centralized orchestration and full access to all models.

Other work studies response evaluation and model routing. PandaLM [32] and LLM-as-a-Judge [33] employ LLM-based evaluators to score or rank candidate outputs, typically under centralized control. RouteLLM [34], RouterEval [35], and Universal Routing Benchmarks [36] investigate learned routing functions that select a single model per query under cost or quality constraints, again assuming global visibility over candidate models. Similarly, ThriftLLM [37] and SpareLLM [38] focus on cost-aware selection and adaptive execution policies.

A closely related direction is Mixture-of-Experts (MoE) architectures [39], [40], such as Mixtral [41] and DeepSeek-R1/V3 [10], [42], where specialized sub-models are activated within a single centrally trained model. While effective for sparse computation within a single trained model, MoE architectures operate inside a unified training and deployment stack. They do not address coordination across independently developed, separately hosted models with distinct checkpoints, runtime environments, and availability states.

In contrast, LLM-MS targets decentralized orchestration across independently hosted open-source LLMs distributed over a peer-to-peer overlay. Rather than assuming centralized access to all candidate models, LLM-MS explicitly separates: (i) query-time benchmark-informed indexing; (ii) decentralized endpoint discovery via a DHT directory; and (iii) token-budgeted multi-model routing executed incrementally through chunked pulls. Prior routing systems typically assume centralized orchestration with static model availability and do not address peer-to-peer discovery, dynamic availability states, churn, or heterogeneous edge resource constraints.

B. LLMs in Data Management Systems

Within the data management community, extensive work investigates how LLMs can be integrated into database and data processing systems. Vector databases and retrieval-augmented generation (RAG) improve response accuracy by grounding generation in retrieved evidence [43]. Systems such as AquaPipe [44] and GaussDB-Vector [45] optimize vector storage and retrieval pipelines, while other work studies LLM-driven query processing and optimization [46], [47].

LLMs have also been applied to data quality and cleaning tasks, including error detection, zero-shot correction, and generative data cleaning [48]–[50]. These works treat LLMs as components within structured data pipelines, focusing on execution efficiency and integration with database engines.

Complementary work explores system-level optimization under hardware constraints. Akillioglu [51] studies query-aware optimization under resource limitations, while Pan and Li [52] provide a database-centric perspective on LLM inference systems, including scheduling, caching, and hardware acceleration.

In contrast to systems that embed a single LLM into a database pipeline or assume centralized model availability, LLM-MS introduces a decentralized orchestration layer spanning multiple independent LLMs distributed across heterogeneous edge peers. By explicitly accounting for query semantics, benchmark-derived performance profiles, hardware feasibility constraints, dynamic availability states, and peer-to-peer discovery, LLM-MS complements existing LLM-enabled data management systems with adaptive, decentralized, and token-budgeted model selection.

VII. Conclusion

We presented LLM-MS, a decentralized architecture for collaborative LLM servicing across heterogeneous peers. LLM-MS combines benchmark-informed query-time indexing under hardware constraints with token-budgeted adaptive routing over discovered endpoints. Experiments show that the Index provides a controllable memory–quality trade-off and that budgeted routing improves quality and efficiency over static single-model baselines under the same token budget. Future work will focus on stronger adversarial resilience, broader model and hardware diversity, and evaluation in geographically distributed deployments.

References

[1] OpenAI. (2025) Openai. OpenAI. [Online]. Available: https://openai.com

[2] Google. (2025) Google deepmind. Google. [Online]. Available: https://deepmind.google

[3] S. R. Department, "Estimated electricity consumption of data centers compared to selected countries," https://www.statista.com/chart/32689/estimated-electricity-consumption-of-data-centers-compared-to-selected-countries/, 2024.

[4] S. Constantinou, C. Costa, A. Konstantinidis, P. K. Chrysanthis, and D. Zeinalipour-Yazti, "A sustainable energy management framework for smart homes," *IEEE Transactions on Sustainable Computing*, vol. 10, no. 1, pp. 70–81, 2025.

[5] P. K. C. Soteris Constantinou, Andreas Konstantinides and D. Zeinalipour-Yazti, "Green planning of iot home automation workflows in smart buildings," *ACM Transactions on Internet of Things*, vol. 3, no. 4, pp. 1–30, jun 2022. [Online]. Available: https://doi.org/10.1145/3549549

[6] S. Constantinou, A. Konstantinidis, and D. Zeinalipour-Yazti, "Green planning systems for self-consumption of renewable energy," *IEEE Internet Computing*, vol. 27, no. 1, pp. 34–42, 2023.

[7] S. Constantinou, C. Costa, C. Costa, A. Konstantinidis, A. Konstantinidis, M. F. Mokbel, and D. Zeinalipour-Yazti, "A framework for continuous knn ranking of ev chargers with estimated components," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, 2024, pp. 5341–5353.

[8] S. Constantinou, A. Konstantinidis, D. Zeinalipour-Yazti, and P. K. Chrysanthis, "The iot meta-control firewall," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 2523–2534.

[9] M. AI, "Llama 3.1," https://ai.meta.com/blog/meta-llama-3-1/, 2024, accessed: Feb. 2025.

[10] DeepSeek, "Deepseek," https://www.deepseek.com/, DeepSeek, 2025, accessed: Jun. 2025.

[11] M. AI, "Mistral-7b-instruct-v0.3," https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3, 2024, accessed: Feb. 2025.

[12] Google. (2025) Gemma. Google. [Online]. Available: https://deepmind.google/models/gemma/

[13] NebulaStream Project Team, "Nebulastream: An extensible, high-performance streaming engine for multi-modal edge applications," in *Companion of the 2025 International Conference on Management of Data*, ser. SIGMOD/PODS '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 195–198. [Online]. Available: https://doi.org/10.1145/3722212.3725118

[14] P. Delaunay and X. B. et al., "Introducing milabench: Benchmarking accelerators for ai," 2024. [Online]. Available: https://arxiv.org/abs/2411.11940

[15] J. Wehrstein, C. Binnig, F. Özcan, S. Vasudevan, Y. Gan, and Y. Wang, "Towards foundation database models," in *Conference on Innovative Data Systems Research (CIDR)*, 2025.

[16] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A local search mechanism for peer-to-peer networks," in *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, ser. CIKM '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 300307. [Online]. Available: https://doi.org/10.1145/584792.584842

[17] D. Zeinalipour-Yazti, V. Kalogeraki, and D. Gunopulos, "pfusion: A p2p architecture for internet-scale content-based search and retrieval," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 804–817, 2007.

[18] K. Krasovitskiy, S. Christou, and D. Zeinalipour-Yazti, "LLM-MS: A multi-model LLM search engine," in *41st IEEE International Conference on Data Engineering, ICDE 2025 - Workshops, Hong Kong, May 19-23, 2025*. IEEE, 2025, pp. 107–114. [Online]. Available: https://doi.org/10.1109/ICDEW67478.2025.00020

[19] H. Face, "Open llm leaderboard," https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard, 2025, accessed: Aug. 2025.

[20] ——, "Sentence transformers, all-minilm-l6-v2," https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2, 2025, machine learning model. Accessed October 2025.

[21] Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, T. Li, M. Ku, K. Wang, A. Zhuang, R. Fan, X. Yue, and W. Chen, "Mmlu-pro: a more robust and challenging multi-task language understanding benchmark," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, ser. NIPS '24. Red Hook, NY, USA: Curran Associates Inc., 2024.

[22] M. Suzgun, N. Scales, N. Schrli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou, and J. Wei, "Challenging big-bench tasks and whether chain-of-thought can solve them," 2022. [Online]. Available: https://arxiv.org/abs/2210.09261

[23] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman, "Gpqa: A graduate-level google-proof q&a benchmark," 2023. [Online]. Available: https://arxiv.org/abs/2311.12022

[24] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou, "Instruction-following evaluation for large language models," 2023. [Online]. Available: https://arxiv.org/abs/2311.07911

[25] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the math dataset," 2021. [Online]. Available: https://arxiv.org/abs/2103.03874

[26] Z. Sprague, X. Ye, K. Bostrom, S. Chaudhuri, and G. Durrett, "Musr: Testing the limits of chain-of-thought with multistep soft reasoning," 2024. [Online]. Available: https://arxiv.org/abs/2310.16049

[27] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. Berlin, Heidelberg: Springer-Verlag, 2002, p. 5365.

[28] T. Authors, "Truthfulqa: A dataset for evaluating ai's truthfulness,"

https://huggingface.co/datasets/truthfulqa/truthful_qa/tree/main, 2025, accessed: May 2025.
[29] H. Face, "Hugging face transformers," https://huggingface.co/docs/transformers, 2025, accessed: Sept. 2025.
[30] 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/
[31] 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
[32] Y. Wang, Z. Yu, W. Yao, 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," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=5Nn2BLV7SB
[33] 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
[34] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. Kadous, and I. Stoica, "Routellm: Learning to route llms from preference data," in *International Conference on Representation Learning*, Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, Eds., vol. 2025, 2025, pp. 34 433–34 448.
[35] Z. Huang, G. Ling, Y. Lin, Y. Chen, S. Zhong, H. Wu, and L. Lin, "RouterEval: A comprehensive benchmark for routing LLMs to explore model-level scaling up in LLMs," in *Findings of the Association for Computational Linguistics: EMNLP 2025*, C. Christodoulopoulos, T. Chakraborty, C. Rose, and V. Peng, Eds. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 3860–3887. [Online]. Available: https://aclanthology.org/2025.findings-emnlp.208/
[36] W. Jitkrittum, H. Narasimhan, A. S. Rawat, J. Juneja, C. Wang, Z. Wang, A. Go, C.-Y. Lee, P. Shenoy, R. Panigrahy, A. K. Menon, and S. Kumar, "Universal model routing for efficient llm inference," 2025. [Online]. Available: https://arxiv.org/abs/2502.08773
[37] K. Huang, Y. Shi, D. Ding, Y. Li, Y. Fei, L. Lakshmanan, and X. Xiao, "Thriftllm: On cost-effective selection of large language models for classification queries," *Proc. VLDB Endow.*, vol. 18, no. 11, pp. 4410–4423, Jul. 2025. [Online]. Available: https://doi.org/10.14778/3749646.3749702
[38] S. Jo and I. Trummer, "Sparellm: Automatically selecting task-specific minimum-cost large language models under equivalence constraint," *Proc. ACM Manag. Data*, vol. 3, no. 3, Jun. 2025. [Online]. Available: https://doi.org/10.1145/3725356
[39] 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
[40] 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
[41] M. A. Team, "Mixtral of experts," https://arxiv.org/abs/2401.04088, 2024, accessed: Jun. 2025.
[42] DeepSeek-AI, A. Liu, and B. F. et al., "Deepseek-v3 technical report," https://arxiv.org/abs/2412.19437, 2025, accessed: Sept. 2025.
[43] G. Li, X. Zhou, and X. Zhao, "Llm for data management," *Proc. VLDB Endow.*, vol. 17, no. 12, pp. 4213–4216, Aug. 2024. [Online]. Available: https://doi.org/10.14778/3685800.3685838
[44] R. Yu, W. Huang, S. Bai, J. Zhou, and F. Wu, "Aquapipe: A quality-aware pipeline for knowledge retrieval and large language models," *Proc. ACM Manag. Data*, vol. 3, no. 1, Feb. 2025. [Online]. Available: https://doi.org/10.1145/3709661
[45] J. Sun, G. Li, J. Pan, J. Wang, Y. Xie, R. Liu, and W. Nie, "Gaussdb-vector: A large-scale persistent real-time vector database for llm applications," *Proc. VLDB Endow.*, vol. 18, no. 12, pp. 4951–4963, Aug. 2025. [Online]. Available: https://doi.org/10.14778/3750601.3750619
[46] D. Satriani, E. Veltri, D. Santoro, S. Rosato, S. Varriale, and P. Papotti, "Logical and physical optimizations for sql query execution over large language models," *Proc. ACM Manag. Data*, vol. 3, no. 3, Jun. 2025. [Online]. Available: https://doi.org/10.1145/3725411
[47] Y. Lin, M. Hulsebos, R. Ma, S. Shankar, S. Zeighami, A. G. Parameswaran, and E. Wu, "Querying templatized document collections with large language models," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, 2025, pp. 2422–2435.
[48] W. Ni, K. Zhang, X. Miao, X. Zhao, Y. Wu, Y. Wang, and J. Yin, "Zeroed: Hybrid zero-shot error detection through large language model reasoning," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, 2025, pp. 3126–3139.
[49] Y. Wu, C. Yang, M. Zhu, X. Miao, W. Ni, M. Xi, X. Zhao, and J. Yin, "A zero-training error correction system with large language models," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, 2025, pp. 2949–2962.
[50] M. Yan, Y. Wang, Y. Wang, X. Miao, and J. Li, "Gidcl: A graph-enhanced interpretable data cleaning framework with large language models," *Proc. ACM Manag. Data*, vol. 2, no. 6, Dec. 2024. [Online]. Available: https://doi.org/10.1145/3698811
[51] K. Akillioglu, "Optimizing data systems for llm workloads," in *Proceedings of the VLDB 2025 Workshop: PhD Workshop*. VLDB Endowment, 2025, phD Workshop poster/paper.
[52] J. Pan and G. Li, "Database perspective on llm inference systems," *Proc. VLDB Endow.*, vol. 18, no. 12, pp. 5504–5507, Aug. 2025. [Online]. Available: https://doi.org/10.14778/3750601.3750703