Bachelor Thesis

A Platform for Querying Multiple LLM Models Simultaneously

Konstantin Krasovitskiy

University of Cyprus

Department of Computer Science

May 2025

University of Cyprus

Department of Computer Science

A platform for querying multiple LLM models simultaneously

Konstantin Krasovitskiy

Supervisor

Prof. Demetris Zeinalipour

The Individual Thesis was submitted in partial fulfillment of the requirements for the Degree of Computer Science of the Department of Computer Science of the University of Cyprus.

May 2025

Acknowledgments

I would like to express my deepest gratitude to my thesis supervisor, Prof. Demetris Zeinalipour, for his unwavering support, invaluable guidance, and insightful feedback throughout the completion of this work. His expertise, patience, and dedication were instrumental in shaping this thesis, and I am truly grateful for his mentorship. I would also like to thank Demetra Paphitou from the DMSL lab for her generous assistance and thoughtful insights, and Stelios Christou for his valuable help and support throughout this project. I also extend my sincere appreciation to my family and friends for their constant encouragement, understanding, and support during this journey. Their belief in me kept me motivated even during the most challenging moments.

Credits

This thesis is based on knowledge gained in the context of the following scientific work, which is currently being published, co-authored with Stelios Christou and Prof. Demetris Zeinalipour:

- ”LLM-MS: A Multi-Model LLM Search Engine”, Konstantin Krasovitskiy, Stelios Christou, and Demetrios Zeinalipour-Yazti, First Intl. Workshop on Coupling of Large Language Models with Vector Data Management (LLM+Vector Data) (LLMVDB’25), collocated with the 40th IEEE International Conference on Data Engineering (IEEE ICDE’25), IEEE Computer Society, pp. 8 pages, Hong Kong SAR, China | May 19 – 23, 2025, DOI: , 2025. (accepted)

Abstract

Large Language Models (LLMs) are evolving at an unprecedented pace, with new variants and capabilities released regularly. While this proliferation has expanded the possibilities for deploying specialized models in diverse tasks such as open-domain question answering, code synthesis, mathematics, summarization, translation, and beyond, it has also introduced substantial complexity for users and developers. In particular, choosing the most suitable LLM for a given query has become more difficult due to differences in model size, training data, and alignment characteristics. Traditional single-model deployments often result in suboptimal trade-offs between accuracy, cost. In this work a platform is presented for orchestration of multiple open-source LLMs. The platform invokes candidate models in parallel to produce partial outputs, continuously evaluates these outputs for semantic relevance and inter-model agreement, and reallocates token budgets dynamically by pruning low-performing models and concentrating resources on the most promising ones. The architecture supports plug-and-play integration of heterogeneous models, which is demonstrated by combining LLaMA, Mistral, and Qwen under a unified interface. To guide model selection under strict token budgets two algorithms were explored: i) the Overperformers Underperformers Algorithm (OUA), which assigns each model a combined reward score equal to α times the semantic similarity to the user prompt plus β times the agreement score with other models, ii) Multi-Armed Bandit (MAB), which treats LLMs as an arm with an unknown reward distribution, where each round the algorithm allocates the next token chunk to the model with the highest upper confidence bound, and adjusts an exploration parameter γ that decays as the token budget is consumed. The platform was implemented end to end, from distributed token dispatching through centralized scoring and final aggregation. Evaluated performance on the TruthfulQA benchmarks using two metrics: F1-score and accuracy per token usage. Our results show that intelligent multi-model orchestration can yield improvement in F1-score and accuracy compared to static single-model baselines. These findings demonstrate the practical viability of our approach for delivering accuracy/cost effective LLM services.

Contents

1 Introduction **1**

1.1 The Growing Ecosystem of Large Language Models 1

1.2 Challenges in Managing and Accessing Multiple LLMs 2

1.3 Motivation for a Unified Multi-LLM Querying Platform 3

1.4 Overview of the Multi-LLM Querying Platform 4

1.5 Contributions 5

2 Background and Related Work **7**

2.1 Overview of Large Language Models 7

2.1.1 Transformer Foundations 8

2.1.2 Pretraining Objectives and Fine-Tuning Paradigms 10

2.1.3 Scaling Laws and Emergent Behaviors 11

2.1.4 Taxonomy of Model Variants 12

2.1.5 Training Strategies and Optimization 13

2.1.6 Capabilities, Limitations, and Ethical Considerations 14

2.1.7 Emerging Trends and Future Directions 15

2.1.8 Summary 16

2.2 Diversity in Models: Capabilities and Limitations 17

2.3 Multi-Model and Ensemble Approaches 17

2.4 Vector Databases and Retrieval-Augmented Generation (RAG) 18

2.5 System-Level Considerations for LLM Platforms 19

3 Platform Architecture **20**

3.1 Design Principles 20

3.2 Hardware Layer 21

3.3 Storage Layer 21
3.4 Computation Layer 22
3.5 Application Layer 22
3.6 Extensibility and Integration 22

4 Computation Layer **23**
4.1 Overview 23
4.2 LLM-MS OUA (Overperformers–Underperformers Algorithm) 24
4.2.1 Algorithm Description 24
4.2.2 Pseudocode 25
4.3 LLM-MS MAB (Multi-Armed Bandit Algorithm) 25
4.3.1 Algorithm Description 26
4.3.2 Pseudocode 26

5 Application Layer **27**
5.1 Landing Page and Main Query Interface 27
5.2 Session Management 27
5.3 Algorithm Selection and Settings 28
5.4 Model Selection and Chat Interface 29
5.5 Contextual Memory and Mobile Layout 29

6 Platform Querying Process **33**
6.1 Query Lifecycle 33
6.2 Preprocessing and Embedding Generation 34
6.3 Dynamic Model Selection and Token Allocation 34
6.4 Response Aggregation and Delivery 35
6.5 Context and Session Management 36
6.6 User Interface Overview 36

7 Implementation Details **38**
7.1 Technologies Used 38
7.2 Data Flow and Interaction Between Components 39
7.3 User Interface Features 40

8 Experimental Evaluation **41**

8.1 Experimental Setup 41

8.2 Metrics 44

8.3 Results 46

8.3.1 Average Reward per Model 46

8.3.2 Average F1 Score per Model 46

8.3.3 Average Reward to Tokens Ratio per Model 46

8.4 Analysis 47

9 Conclusion **51**

9.1 Summary 51

9.2 Experimental Insights 52

9.3 Benefits of a Platform Approach 52

9.4 Challenges and Limitations 53

9.5 Future Extensions 53

9.6 Final Remarks 55

Chapter 1

Introduction

1.1 The Growing Ecosystem of Large Language Models 1
1.2 Challenges in Managing and Accessing Multiple LLMs 2
1.3 Motivation for a Unified Multi-LLM Querying Platform 3
1.4 Overview of the Multi-LLM Querying Platform 4
1.5 Contributions 5

1.1 The Growing Ecosystem of Large Language Models

Large Language Models (LLMs) have become foundational to modern natural language processing (NLP), demonstrating strong capabilities in understanding and generating human-like text [1]. In recent years, a wide range of LLMs have been released by organizations such as OpenAI (e.g., GPT-2, GPT-3 [2, 3]), Anthropic (Claude [4]), and Meta (LLaMA series [5]), among others. These models vary significantly in architecture, scale and training methodology, creating a rapidly expanding ecosystem of tools suitable for diverse applications.

While some models are trained to perform well across general-purpose tasks, many are fine-tuned for specific domains such as code generation, legal reasoning, or biomedical information extraction. Architecturally, they include both decoder-only models optimized for auto regressive tasks and encoder-decoder models better suited for sequence-to-sequence applications. Some recent efforts also explore sparse expert models and hybrid routing techniques, introducing even more architectural variation [6].

This growing diversity, while promising, poses a significant challenge: different models often behave inconsistently across tasks and input types due to differences in training data, alignment objectives, and inference dynamics. For practitioners, selecting the most appropriate model for a given query often requires trial-and-error or specialized knowledge. This highlights the need for more systematic approaches to LLM selection and orchestration, particularly in settings where accuracy, efficiency, and domain alignment are critical.

1.2 Challenges in Managing and Accessing Multiple LLMs

As the ecosystem of Large Language Models (LLMs) continues to grow, integrating and managing multiple models within a single system introduces both architectural and operational challenges. Each model is typically developed with a different training dataset, tokenization strategy, and alignment objective, making them non-interoperable by default. This heterogeneity, while enabling specialization, increases complexity when attempting to unify their usage in real-world applications.

Most LLMs are accessible via diverse interfaces-ranging from proprietary APIs (e.g., OpenAI, Anthropic) to open-source tool chains-which require model-specific configurations, prompting schemes, and I/O formats. As a result, developers are often forced to implement custom logic to support multiple backends, which leads to duplicated effort, inconsistent behavior, and technical debt. These issues are compounded when attempting to evaluate models side-by-side or compose them in an ensemble.

In addition to software-level complexity, computational resource constraints are a major limiting factor. Running several large-scale models concurrently places a significant burden on available GPU memory and compute capacity. This problem is particularly acute in real-time applications, where token usage directly affect both cost and user experience. For example, invoking multiple models for every query without coordination can lead to substantial inefficiencies, as token limits and rate constraints vary across providers.

Furthermore, without a unifying control mechanism, selecting the most suitable model for a specific query becomes a manual and heuristic-driven process. Users must either rely on subjective comparison across outputs or use rigid fallback logic that fails to exploit the nuanced strengths of different models. This lack of orchestration undermines the benefits of model diversity and impedes scalable adoption.

These observations underscore the need for a platform-level approach to LLM integration-one that abstracts model-specific behavior, supports dynamic selection, and manages resources in a unified, efficient manner.

1.3 Motivation for a Unified Multi-LLM Querying Platform

The rapid diversification of large language models (LLMs) has created a compelling opportunity, and corresponding necessity, for unified systems that can manage and coordinate access to multiple models. While many LLMs demonstrate strong performance on specialized tasks, no single model offers consistent superiority across all domains, query types, or prompt styles. As such, relying exclusively on one model is both computationally inefficient and strategically limiting, particularly in dynamic application settings.

This motivates the development of systems capable of orchestrating multiple LLMs under a single querying framework. Rather than treating LLMs as static tools, such systems can treat them as dynamically interchangeable components, selected based on their task relevance, prior performance, and resource constraints. This reflects a broader shift toward adaptive, modular architectures in software systems, where runtime decisions optimize for context and efficiency.

Moreover, many real-world use cases involve input that extends beyond a standalone natural language query-such as user-provided documents, embedded context, or prior conversational turns. Integrating retrieval-augmented generation (RAG) pipelines enables a more grounded response, leveraging vector similarity to extract relevant information from document stores and embed it into the prompt space. This is particularly valuable for applications in law, education, and enterprise workflows, where context-aware answers are essential.

The motivation of this work, therefore, is to close the gap between the growing availability of task-specific LLMs and the lack of infrastructure for their coordinated use. By introducing intelligent selection strategies, dynamic token allocation, and retrieval-driven context enhancement, the proposed system enables scalable and efficient multi-model querying, while abstracting away the operational burden typically required to work with LLMs individually.

1.4 Overview of the Multi-LLM Querying Platform

The proposed system is a web-based platform designed to facilitate interactive, multi-model generation. It is based a layered architecture that separates concerns across four domains: user interaction, orchestration logic, model execution, and context management. Users submit queries through a browser interface, optionally accompanied by supplemental documents or files. The system processes the input, generates embeddings, and then dynamically determines the most relevant models to invoke. Tokens are allocated among selected models via Overperformers–Underperformers Algorithm (OUA) and Multi-Armed Bandit (MAB) strategies. Responses are generated with real-time feedback.
The architecture is organized into the following layers:

- **Hardware Layer**
  Manages inference hardware resources, abstracting device specific details and optimizing resource utilization for concurrent model execution.

- **Storage Layer**
  Hosts local model files and manages temporary in-memory storage of session embeddings. Uses ChromaDB for embedding indexing and similarity based retrieval of relevant document segments.

- **Computation Layer**
  Executes models through an engine that supports parallel inference and manages the orchestration logic. It streams model outputs and supports partial generation, enabling responsive query processing and dynamic allocation of tokens across multiple LLMs.

- **Application Layer**
  Provides the user interface and handles interaction with the end user. It coordinates query input, displays streaming responses, and offers controls for switching models or tuning parameters via a Flask-based web frontend.

Key capabilities of the platform include:

- **Adaptive Model Selection**
  Employs intelligent strategies based on similarity and inter-model agreement to dynamically select models and allocate tokens efficiently.

- **Contextual Query Enhancement**
  Utilizes vector embeddings of uploaded files and session context to augment queries, enabling more informed and contextually relevant responses.

- **Session Continuity**
  Maintains conversational context via hierarchical summarization and temporary embedding storage in memory during the session, respecting input limits.

- **User-Friendly Interface**
  Provides a simple, responsive web application through Flask, allowing users to interact seamlessly without technical expertise.

This modular design promotes extensibility, enabling future integration of new models or algorithms without substantial changes to user experience or core orchestration logic.

1.5 Contributions

This thesis presents the design and implementation of a comprehensive platform for orchestrating multiple large language models in response to user queries. The main contributions of this work are summarized as follows:

- **System Architecture:** A modular, multi-layered platform was developed that cleanly separates hardware management, storage, computation, and user interaction. This architecture facilitates seamless integration of models and data pipelines while supporting real-time querying.

- **Dynamic Query-Orchestration Algorithms:** Two token allocation and model selection strategies were designed and integrated:

  - An Overperformers–Underperformers Algorithm (OUA) that evaluates partial outputs from models and dynamically reallocates token budgets to the most relevant responses in real time.
  - A Multi-Armed Bandit (MAB) strategy based on the UCB1 algorithm, which balances exploration of different models with exploitation of promising ones during response generation.

- **Retrieval-Augmented Generation Integration:** The platform embeds user-supplied documents using a vector encoder and stores them in a vector database. At query time, relevant document fragments are retrieved and incorporated into model prompts, enabling informed, document-specific answers.
- **Efficient Resource Usage:** Through selective token allocation and early pruning of underperforming models, the platform reduces wasted compute cycles and avoids unnecessary token generation, resulting in lower latency and improved response quality per token.
- **Practical Implementation:** A full-stack web application was implemented using Flask, Ollama, and ChromaDB, supporting concurrent multi-model inference, file uploads, context summarization, and interactive user sessions via a responsive real-time interface.
- **Experimental Evaluation:** The system was evaluated on benchmark datasets, demonstrating enhanced response accuracy and computational efficiency compared to static single-model baselines. Results confirm that intelligent multi-model orchestration yields superior practical outcomes.

Collectively, these contributions provide a practical framework for improving accessibility and efficiency of large language models in real-world querying scenarios by shifting the paradigm from isolated models to intelligent orchestration within a unified platform.

Chapter 2

Background and Related Work

2.1 Overview of Large Language Models 7

2.1.1 Transformer Foundations 8

2.1.2 Pretraining Objectives and Fine-Tuning Paradigms 10

2.1.3 Scaling Laws and Emergent Behaviors 11

2.1.4 Taxonomy of Model Variants 12

2.1.5 Training Strategies and Optimization 13

2.1.6 Capabilities, Limitations, and Ethical Considerations 14

2.1.7 Emerging Trends and Future Directions 15

2.1.8 Summary 16

2.2 Diversity in Models: Capabilities and Limitations 17

2.3 Multi-Model and Ensemble Approaches 17

2.4 Vector Databases and Retrieval-Augmented Generation (RAG) . . 18

2.5 System-Level Considerations for LLM Platforms 19

2.1 Overview of Large Language Models

Large Language Models (LLMs) are deep neural networks trained to predict the next token in a sequence over massive text corpora [7]. Three factors have really driven their meteoric rise. First, the *Transformer* architecture allows highly parallelized attention computations-no more cumbersome recurrent models. Second, sheer *scale* (both in parameters and training data) consistently boosts capabilities. Third, clever pretraining objectives and post-training refinements (think domain-specific fine-tuning or instruction tuning) help these models adapt to real-world use cases. In the pages that follow, the architectural foundations, training paradigms, scaling behaviors, and practical considerations behind today's LLMs are unpacked.

2.1.1 Transformer Foundations

To see why Transformers became so popular, let's dig into their core building blocks. Unlike earlier RNN or CNN-based sequence models, a Transformer relies entirely on self-attention and feed-forward layers, organized into stacked "encoder" and "decoder" blocks or as *decoder-only* stacks. In other words, the model learns contextual relationships between tokens without processing them sequentially.

- **Multi-Head Self-Attention:** Each Transformer block computes a set of attention heads. For each head h, the input token embeddings $X \in \mathbb{R}^{n \times d}$ (with n tokens and dimension d) are linearly projected into queries Q_h, keys K_h, and values V_h. The attention output is

$$\text{Attention}_h(Q_h, K_h, V_h) = \text{softmax}\Big(\frac{Q_h K_h^\top}{\sqrt{d_k}}\Big)V_h,$$

  where d_k is the per-head key dimension. Multiple heads are concatenated and linearly projected to form the block's attention output [8, 9]. In practice, this mechanism lets the model focus on different parts of the sequence simultaneously.

- **Positional Encoding:** Because self-attention is permutation-invariant, positional encodings are added to the raw token embeddings to inject a notion of token order. Typical choices include sinusoidal encodings [8] or learned positional embeddings, both of which allow the model to discern relative and absolute positions in the sequence.

- **Layer Normalization and Residual Connections:** Each sublayer (multi-head attention or feed-forward network) is wrapped with a residual (skip) connection and layer normalization. Let Sublayer$(\cdot)$ denote either the attention or feed-forward computation. Then a Transformer block applies

$$Y = \text{LayerNorm}\big(X + \text{Sublayer}(X)\big).$$

This design stabilizes training at large depths by preventing vanishing or exploding gradients [8].

- **Feed-Forward Networks:** Following the attention sublayer, each block applies a position-wise feed-forward network of the form

$$\mathrm{FFN}(X) = \mathrm{GELU}(XW_1 + b_1)\,W_2 + b_2,$$

which projects the attention outputs to a higher hidden dimension before projecting back down to the model dimension d [8]. In practice, this expands the representational capacity of each token's contextual embedding.

Figure 2.1 shows a schematic of the encoder-decoder Transformer. In *decoder-only* LLM variants (e.g., GPT-3 [10], GPT-4 [11]), only the decoder side is employed, using masked self-attention to ensure autoregressive generation (i.e., each token attends only to preceding tokens). Decoder-only models tend to be more efficient for purely generative tasks, whereas encoder-decoder (seq2seq) models (e.g., T5 [12]) excel in tasks requiring conditional generation given a source input.

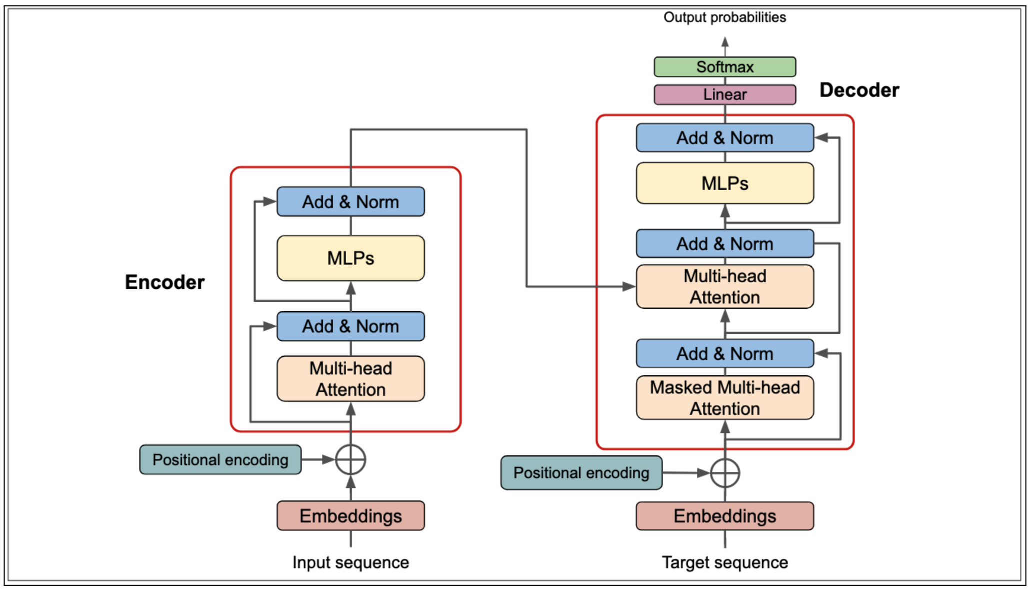


Figure 2.1: Detailed view of the Transformer encoder–decoder architecture.
Source: https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.7648

2.1.2 Pretraining Objectives and Fine-Tuning Paradigms

Having outlined how attention and feed-forward layers fit together inside a Transformer, let us now explore how these models are actually taught to “speak” and “reason.”
Unsupervised pretraining is typically the first stage. A massive corpus-often including web text, books, code, and other documents-is used to optimize the model parameters θ by minimizing a language modeling loss. For a decoder-only architecture, this means maximizing the likelihood of each token given all previous tokens:

$$\mathcal{L}_{\mathrm{LM}}(\theta) = -\mathbb{E}_{(x_1,\dots,x_T)\sim\mathcal{D}} \sum_{t=1}^{T} \log P_\theta\big(x_t \mid x_{<t}\big).$$

Here, $\mathcal{D}$ denotes the distribution over sequences in the pretraining corpus. In encoder-decoder models (e.g., BERT or T5 variants), masked language modeling (MLM) or denoising objectives are used instead: a fraction of input tokens are masked, and the model learns to reconstruct the original tokens [12, 13]. These unsupervised objectives let the model discover syntactic and semantic patterns at scale. In practice, datasets often exceed hundreds of billions of tokens, and model sizes can span from a few hundred million to hundreds of billions of parameters [14–16].
Once pretrained, models undergo post-training refinements to become more aligned with human instructions. Instruction tuning involves fine-tuning on curated (instruction, response) pairs so that the model learns to follow natural language prompts more reliably. For instance, FLAN-T5 [17] and Stanford Alpaca [18] illustrate how a relatively small fine-tuning dataset can drastically improve instruction-following behavior. We often find that instruction-tuned models handle open-ended user queries more gracefully and produce fewer nonsensical outputs.
A further step is Reinforcement Learning from Human Feedback (RLHF). In RLHF, human annotators rank multiple candidate outputs, and these rankings train a reward model. Then, algorithms such as Proximal Policy Optimization (PPO) adjust the LLM’s parameters to maximize expected human satisfaction [19–21]. OpenAI’s GPT-3.5 and GPT-4, as well as Anthropic’s Claude 2, have all used RLHF extensively to reduce harmful content, improve factual accuracy, and align with human values [11, 15]. In practice, this yields a model that feels more cooperative and safer to deploy.

2.1.3 Scaling Laws and Emergent Behaviors

At small parameter counts, a vanilla Transformer can already produce coherent text. But what really sets today's LLMs apart is extreme scaling. Empirical studies show that increasing the number of parameters (N), the size of the training dataset (D), and the compute budget (C) leads to predictable improvements in performance across a wide array of language tasks [22, 23]. Specifically, loss metrics follow approximate power-law relationships:

$$\text{Loss}(N, D) \approx \alpha\, N^{-\eta} + \beta\, D^{-\mu} + \gamma,$$

where $\alpha, \beta, \gamma, \eta, \mu$ are constants that depend on the model architecture and data distribution. In practice, scaling to hundreds of billions-or even trillions-of parameters has become the default way to push benchmark performance.

Beyond just "bigger is better," very large models exhibit *emergent behaviors* [24] that simply don't appear at smaller scales. For example:

- **In-Context Learning:** As demonstrated by GPT-3 (175B) [10], the model can learn to perform a novel task from just a handful of examples provided in the prompt-no gradient updates needed. This few-shot ability emerges only beyond a certain parameter threshold.

- **Chain-of-Thought Reasoning:** Large models such as PaLM (540B) [25] can generate intermediate reasoning steps that help solve multi-step problems (e.g., arithmetic or logical puzzles), significantly improving accuracy.

- **Cross-Lingual Transfer:** Massive multilingual LLMs (e.g., XLM-R [26], BLOOM [27]) show strong zero-shot performance on languages that they rarely or never saw during fine-tuning. This happens because large, shared parameter spaces allow the model to generalize linguistic patterns across languages.

That said, these gains have trade-offs. Larger models demand exponentially more compute during both training and inference, require massive memory footprints, and incur higher latency in real-time applications. This motivates research into more efficient variants-Mixture-of-Experts (MoE) architectures [28, 29], sparse attention mechanisms, quantization methods such as LlamaQuant [30], and distillation techniques like DistilBERT [31]-

all of which aim to preserve most performance while reducing resource costs, or low-latency acceleration (LLMA) [32].

2.1.4 Taxonomy of Model Variants

With an understanding of how LLMs are built and trained, the next question is: Which specific architectures and design choices have practitioners actually built? Contemporary surveys categorize models along several key axes [6, 33, 34]:

1. **Autoregressive vs. Autoencoding.** Autoregressive (decoder-only) models, such as the GPT family [10, 11], generate tokens left-to-right and excel in text synthesis. Autoencoding (encoder-decoder) models, like BART [35] and T5 [12], are trained to reconstruct masked or corrupted inputs, making them powerful for sequence-to-sequence tasks (translation, summarization).

2. **Dense vs. Sparse (Mixture-of-Experts).** Dense Transformers allocate all parameters to each input. In contrast, Mixture-of-Experts (MoE) models-such as MoMQ [36], GLaM [37] and Switch Transformer [28]-route each token through only a subset of "expert" feed-forward sublayers. This lets MoE models reach trillions of parameters while keeping inference costs moderate.

3. **Monolingual vs. Multilingual vs. Multimodal.** Monolingual LLMs (e.g., LLaMA [5, 14], Mistral [16, 38]) are optimized for English or one specific language. Multilingual LLMs (e.g., XLM-R [26], mT5 [39]) are trained on mixed-language corpora. Multimodal models (e.g., Flamingo [40], GPT-4 with vision [11]) accept image or audio inputs alongside text.

4. **Open-Source vs. Proprietary.** Open-source ecosystems-Hugging Face Transformers [41, 42], Meta's LLaMA [14], MosaicML's MPT [43]-foster community-driven innovation. Proprietary APIs (OpenAI [11], Anthropic [15], Google PaLM2 [25]), Google Vertex-AI [44], Gemini [45], Gemini Flash [46] often provide larger, more heavily aligned models behind paywalls.

Table 2.1 summarizes representative models across these categories. By placing each model within this framework, we can better compare capabilities, licensing constraints, and typical use cases.

Table 2.1: Representative LLMs: Key architecture, data, and usage characteristics.

Model (Year)	**Size**	**Type**	**Training Data**	**Highlights**
GPT-2 [47] (2019)	1.5B	Dec-Only	WebText (40 GB)	First open large decoder, good zero-shot performance
GPT-3 [10] (2020)	175B	Dec-Only	CCrawl, Books, Wiki	Enabled few-shot learning via prompts
PaLM [25] (2022)	540B	Dec-Only	Multilingual mix	Strong chain-of-thought reasoning
BERT [13] (2018)	340M	Enc-Only	Books, Wiki	Bidirectional masked LM, widely used
T5 [12] (2020)	11B	Enc-Dec	C4 Corpus	Treats all NLP tasks as text generation
LLaMA [14] (2023–25)	7–70B	Dec-Only	Filtered CCrawl	Open models for fine-tuning and research
Mistral [16] (2023)	7B	Dec-Only	Web, Code	Small, fast, competitive on benchmarks
Qwen 2.5 [48] (2025)	7B	Dec-Only	Internal, Code	Multilingual support, strong in coding tasks
Claude 2 [15] (2023)	$\sim$100B	Dec-Only	Anthropic corpora	Focused on alignment via Constitutional AI
GLaM [37] (2022)	1.2T	Sparse MoE	Web, Books	Trillion-param MoE with efficient inference
Flamingo [40] (2022)	80B	Multimodal	Text, Images	Zero-shot visual question answering

2.1.5 Training Strategies and Optimization

With the taxonomy in mind, the next question is: how do we train these massive models efficiently? In practice, training LLMs requires careful orchestration of hardware, software, and optimization techniques:

- **Mixed-Precision and Distributed Training.** Most modern LLMs use mixed-precision (FP16 or bfloat16) to reduce memory usage and increase throughput [49]. Data parallelism (e.g., AllReduce) and pipeline model parallelism (e.g., Megatron-LM [50]) distribute parameters and activations across multiple GPUs or nodes.

- **Learning Rate Schedules and Warm-Up.** A linear or cosine learning rate warm-up followed by decay helps prevent early training instability [8]. In practice, schedulers often ramp up over several thousand steps before decaying.

- **Gradient Checkpointing.** To reduce memory footprint, intermediate activations are not stored for all layers. Instead, selective recomputation is used during backpropagation [51], allowing models to trade extra compute for lower memory usage.

- **Tokenization and Vocabulary.** Subword tokenizers like Byte Pair Encoding (BPE) or SentencePiece enable variable-length text encoding while capping vocabulary size (e.g., 50K–100K tokens) [52]. In practice, a well-designed tokenizer can dramatically reduce out-of-vocabulary errors and improve downstream performance.

2.1.6 Capabilities, Limitations, and Ethical Considerations

From a user's perspective, LLMs can feel almost magical-but these capabilities come with trade-offs.

- **Zero-Shot and Few-Shot Learning.** By providing natural language instructions or a few examples within a prompt, LLMs can generalize to novel tasks without additional gradient updates [10]. In practice, this means you often get useful answers from an LLM-even if you haven't fine-tuned it for your specific domain.
- **Multi-Tasking.** A single pretrained LLM can handle translation, summarization, question answering, and code generation simply by adjusting the prompt and decoding constraints.
- **Long-Range Coherence.** Models with extended context windows (e.g., 32K tokens or more in GPT-4, PaLM 2) maintain coherence across multi-paragraph or multi-page document generation [11].
- **Emergent Reasoning.** As discussed earlier, phenomena such as chain-of-thought prompting and symbolic manipulation emerge only at very large scales [53].

Limitations and Risks.

- **Hallucinations.** LLMs sometimes generate plausible but incorrect or nonsensical statements, especially when producing factual content without grounding [54]. In practice, users must verify critical outputs against reliable sources. Holistic evaluation benchmarks (e.g., HELM) have been proposed to more comprehensively measure such risks [55], and recent benchmarks such as TruthfulQA [56] specifically measure the "truthfulness" of generated text.
- **Bias and Toxicity.** Pretraining data often contain societal biases, stereotypes, and offensive language. Even aligned models can reflect these biases unless explicitly mitigated [57].

- **Resource Intensity.** Training and serving large models demand substantial computational power, making them inaccessible to smaller organizations and increasing environmental impact [58, 59].
- **Security and Misuse.** LLMs can be repurposed to generate disinformation, spam, or targeted harassment at scale if not properly safeguarded [60].
- **Lack of True Understanding.** Although LLMs can mimic reasoning through pattern recognition, they do not possess genuine comprehension or grounded knowledge. They can fail on tasks that require real-world common sense or precise symbolic logic [61].

These challenges have spurred extensive research on alignment, robustness, and ethical guardrails for LLM deployment [62, 63]. In practice, deploying an LLM responsibly means combining technical mitigations (e.g., data filtering, fine-tuning on safe corpora) with human-in-the-loop processes.

Ethical and Privacy Considerations.

- **Data Privacy.** Models trained on public text may inadvertently memorize personal data (e.g., names, addresses). Careful scrubbing, differential privacy techniques, or on-device inference can mitigate this risk [64].
- **Accountability and Transparency.** It is crucial to provide explanations or provenance for model outputs, especially in high-stakes domains such as healthcare or law [65].
- **Regulatory Compliance.** Emerging EU AI Act guidelines and other privacy/data protection regulations require auditability and fairness assessments [66].

2.1.7 Emerging Trends and Future Directions

With the basics covered, it is worth looking ahead at where LLM research is headed. The rapid evolution suggests several promising avenues:

- **Memory and Retrieval Integration.** Combining parametric knowledge (model weights) with nonparametric retrieval (RAG [67]) can reduce hallucinations and keep models up-to-date [68]. In practice, systems like Retrieval-Augmented Generation embed documents into a vector database and fetch relevant context at inference time.

- **Continual Learning and Adaptation.** Letting LLMs learn from user interactions over time-while avoiding catastrophic forgetting-remains an open challenge [69]. In production, continually fine-tuning or calibrating models could help them adapt to evolving user needs.

- **Efficient and Sparse Architectures.** Research on Mixture-of-Experts, streamlined attention (e.g., FlashAttention [70]), and quantization & pruning methods aims to democratize LLM inference on commodity hardware [32].

- **Multimodal and Situated AI.** Integrating vision, audio, and real-world grounding (e.g., robotics, sensor data) will broaden LLM applicability beyond text [71]. In practice, this means building agents that can see, hear, and act-much like humans do.

- **Regulated, Explainable AI.** As LLMs become more pervasive, frameworks for interpretability, fairness, and safety-supported by transparent partnerships between academia, industry, and regulatory bodies-will be essential [72].

2.1.8 Summary

In this chapter, the key ingredients that make modern LLMs work have been covered : Transformer architectures built on self-attention, large-scale pretraining combined with instruction tuning and RLHF, and the empirical scaling laws that have driven models to hundreds of billions of parameters. Popular models also classified along axes such as autoregressive vs. autoencoding, dense vs. sparse (MoE), and monolingual vs. multilingual vs. multimodal. Finally, practical training strategies, capabilities, limitations, and ethical considerations, and sketched emerging trends-especially memory integration, continual learning, efficient architectures, and multimodal AI were discussed.
Together, these insights set the stage for Chapter 3, introducing multi-model orchestration platform. By treating each LLM as a modular component and combining them with vector-based retrieval and dynamic routing, aimed to build a system that leverages the best of each

model while managing cost, latency, and user experience in interactive applications.

2.2 Diversity in Models: Capabilities and Limitations

No two large language models are identical. Variations in architecture, training corpora, tokenization methods, and alignment techniques result in significant differences in model performance across diverse tasks. Some models are specialized for coding applications (e.g., CodeLLaMA [73] or GPT-Neo [74]) and excel in syntax-sensitive environments, whereas others are optimized for open-domain dialogue, question answering, or multilingual reasoning. For example, Qwen-2 is noted for its strong performance on reasoning-intensive and factual queries, while LLaMA-3 demonstrates fluent and polite conversational abilities, likely influenced by alignment and post-training methods.

System-level differences are also critical. Inference speed, maximum context window size, memory usage, and support for streaming outputs vary notably between models. These factors impact both computational efficiency and suitability for real-time interactive applications. Moreover, alignment strategies such as supervised fine-tuning or preference modeling further shape model responses, particularly for ambiguous, multi-step, or user-directed prompts.

This heterogeneity presents both opportunities and challenges. While specialization enables optimized performance for specific tasks, it complicates the selection of the most suitable model for a given query without explicit evaluation or dynamic orchestration.

2.3 Multi-Model and Ensemble Approaches

There is growing research interest in leveraging the complementary strengths of multiple large language models. Early approaches primarily focused on response fusion, where outputs from multiple models are generated and then either selected or combined to form a final answer [75]. For example, systems that rank responses from a pool of models and fuse the most relevant ones have been proposed. Other frameworks like LLM-Blender [76]) introduce model selection mechanisms driven by cost optimization, aiming to minimize inference overhead while maintaining answer quality.

Early response fusion frameworks ZOOTER [77] and benchmarking-driven routing [78] have shifted towards dynamic model selection and routing, seeking to identify in real time which model or subset of models should process a given query. Some ensemble systems employ pre-trained classifiers or reward-based scoring to select among models, while others utilize reinforcement learning to optimize long-term utility.Recent work such as MetaLLM proposes a dynamic wrapper around multiple LLMs, enabling on the fly routing and cost-based invocation of different models [79].

However, many existing ensemble strategies operate primarily in batch settings or rely on large token budgets, limiting their suitability for live, interactive applications. Moreover, most lack fine-grained control over partial generation, real-time feedback, or token-aware pruning during inference. These limitations underscore the need for interactive platforms capable of efficiently orchestrating multiple models dynamically, rather than depending solely on static ensembles or post-hoc ranking methods.

2.4 Vector Databases and Retrieval-Augmented Generation (RAG)

Large language models are typically trained on static corpora, which can lead to outdated or limited knowledge. Retrieval-Augmented Generation (RAG) addresses this limitation by injecting external, dynamic context into the model's prompt at inference time, so LLMs can optimize query rewrite or diagnostics [80]. This is achieved by embedding relevant documents or user-provided files into high-dimensional vectors, which are then stored in a vector database.

Upon receiving a user query, the system embeds the query and performs a similarity search to identify the most relevant document fragments. These fragments are retrieved from the vector database and incorporated into the prompt given to the language model, enabling responses that are contextually grounded and relevant. This technique is especially valuable in scenarios involving private, domain-specific, or time-sensitive information, such as real-time analysis of uploaded documents or PDFs.

Vector databases like ChromaDB [81] (and more VDBMS techniques [82]) are optimized for low-latency similarity search and integrate seamlessly with Python-based backends. They constitute a critical infrastructure component for scalable RAG implementations, which require efficient cloud-edge collaboration like VELO [83] and lightweight VDBMS design like Bhakti [84] and are essential in multi-model platforms where consistent retrieval context must be shared across multiple language models.

2.5 System-Level Considerations for LLM Platforms

Designing a real-world system capable of coordinating multiple large language models introduces architectural and operational challenges that extend beyond the scope of a single model or static ensemble. These challenges include:

- **Hardware Limitations:** Running multiple large models concurrently requires substantial memory and computational resources, particularly on GPU-based systems.
- **Latency and Cost Management:** Efficient token utilization and adaptive token budgeting are essential when managing inference costs, quotas, or latency requirements.
- **Model Routing and Orchestration:** A control layer must dynamically decide which models to invoke, determine token allocations, and decide when to stop or switch models based on real-time response feedback.
- **Session and Context Management:** In multi-turn interactions, long conversation histories must be intelligently compressed or summarized to retain relevance while adhering to model input length constraints.
- **Modular Integration:** A scalable system should enable plug-and-play integration of new models, vector databases, and orchestration algorithms with minimal friction and disruption.

Few existing solutions provide comprehensive end-to-end orchestration. Platforms such as OpenAI Playground or Hugging Face Transformers simplify the use of models, but they do not address multi-model coordination or cost-aware token management. These considerations show the necessity for platforms that treat models as modular services within a coordinated system, while search engine approaches like LLM-MS [85] attempts at multi-model orchestration

Chapter 3

Platform Architecture

3.1 Design Principles **20**

3.2 Hardware Layer **21**

3.3 Storage Layer **21**

3.4 Computation Layer **22**

3.5 Application Layer **22**

3.6 Extensibility and Integration **22**

3.1 Design Principles

The architecture of the platform was guided by several principles aimed at ensuring scalability, modularity, and efficiency. First, the system uses a layered structure that clearly separates responsibilities among hardware management, data storage, computation, and application logic. This modularity facilitates plug-and-play integration of open source models [86], without requiring major refactoring.

Second, the platform emphasizes multi LLM orchestration over static LLM. Rather than routing all queries through a fixed LLM, the system adapts in real time based on query domain, and feedback from model responses.

Resource efficiency is a critical objective and by implementing selective token allocation and pruning mechanisms,it minimizes unnecessary inference overhead.

Robustness and fault tolerance are ensured through session state persistence and embedding normalization techniques.
Finally, user experience is prioritized by designing for low-latency responses and supporting long-session interactions via summarization and contextual awareness.

3.2 Hardware Layer

The hardware layer forms the foundation of the platform's performance, managing computational resources across GPUs and CPUs. Large language model inference, especially when running multiple models concurrently, requires high memory bandwidth and parallel processing capabilities. The platform utilizes a heterogeneous hardware stack, employing NVIDIA GPUs to enable concurrent execution of models. In scenarios where GPU resources are unavailable, the system falls back to CPU-based inference, ensuring uninterrupted service availability.
To facilitate effective resource management, NVIDIA's System Management Interface (SMI) is employed to monitor GPU utilization and temperature. This information enables intelligent load balancing and helps prevent resource over-utilization.
The hardware layer is fully abstracted from the upper architectural layers, such that application logic does not directly handle low-level device configurations. This abstraction enhances portability, allowing deployment across diverse environments including cloud infrastructure and on-premise servers.

3.3 Storage Layer

The storage layer manages the system's persistent components, including local language model files, session metadata, and vectorized document embeddings. Supported models are stored on disk within an ext4-formatted filesystem and managed by Ollama's model server.
A key feature of this layer is the integration of a vector database. Specifically, ChromaDB enables Retrieval-Augmented Generation (RAG) capabilities. Uploaded user files are preprocessed into high-dimensional embeddings using vector encoders, which are then indexed in ChromaDB to facilitate efficient similarity-based retrieval.

At query time, relevant document fragments are retrieved via cosine similarity search and incorporated into the prompts provided to the language models. An embedding normalization process ensures consistency across all vector representations, enhancing interoperability among different models.
Session persistence is also supported within this layer. Local storage is used to maintain session histories and hierarchical summarizations, enabling sustained multi-turn interactions without exceeding memory limits and preserving conversational continuity.

3.4 Computation Layer

The computation layer is responsible for orchestrating inference across the various large language models. This functionality is implemented through the Ollama daemon (version 0.4.5) [14], which serves as the backend model runtime. The layer receives constructed prompts, executes the selected models, and streams token-by-token outputs back to the application layer. This layer hosts the primary orchestration algorithms, specifically the Overperformer–Underperformer Allocation (OUA) and the Multi-Armed Bandit (MAB) strategies.

3.5 Application Layer

The application layer serves as the system's coordination hub and user-facing endpoint. It is implemented as a Flask web application running on an Apache server, facilitating query intake, file uploads, and real-time streaming of outputs.

3.6 Extensibility and Integration

A key strength of the platform by using its modular architecture, new models can be incorporated by updating the Ollama model registry and registering corresponding API endpoints. Similarly, the vector database schema is designed to be flexible, enabling support for additional embedding formats or metadata as needed.
The system provides standardized interfaces for integrating alternative orchestration algorithms, embedding pipelines. This design aims future-proofing and facilitates experimentation with emerging components as the landscape of large language models evolves.

Chapter 4

Computation Layer

4.1 Overview **23**
4.2 LLM-MS OUA (Overperformers–Underperformers Algorithm) . **24**
4.2.1 Algorithm Description 24
4.2.2 Pseudocode 25
4.3 LLM-MS MAB (Multi-Armed Bandit Algorithm) **25**
4.3.1 Algorithm Description 26
4.3.2 Pseudocode 26

4.1 Overview

The computation layer orchestrates the execution of multiple large language models (LLMs) through the Ollama daemon (version 0.4.5) [14] within the platform. It manages dynamic model selection, token allocation, and streaming inference, aiming to optimize both response quality and computational efficiency.
This layer operates on heterogeneous hardware, leveraging GPUs and CPUs for parallel model execution. It abstracts hardware details and exposes APIs to upper layers and by using the OUA and MAB algortihms enables flexible and efficient multi-model querying.

4.2 LLM-MS OUA (Overperformers–Underperformers Algorithm)

4.2.1 Algorithm Description

Initially, a fixed token budget λ_{max} is divided evenly among N models, with each model receiving at maximum$\lambda_m = \frac{\lambda_{\text{max}}}{N}$. Models generate tokens in a round-robin fashion, producing partial outputs.
Each partial response is vectorized, and similarity scores are computed combining:

- Cosine similarity with the user query embedding in order to determine the alignment of the model response with the question
- Inter-model similarity to other candidate responses in order to enforce a consensus

Models with scores significantly lower than others are pruned to conserve tokens and allocate them to rest beyond each models maximum allowance. The algorithm terminates early if a model's response surpasses others by a defined margin and completes generation, or is the one last standing producing the best answer.

4.2.2 Pseudocode

Algorithm 1 Overperformers–Underperformers Algorithm (OUA)

```
**Input:** Set of LLMs $\{LLM_1, LLM_2, ..., LLM_N\}$, User Prompt $p$, Max Tokens $\lambda_{\text{max}}$
**Output:** Best Response $r_p$
 1: $\alpha \leftarrow 0.7, \beta \leftarrow 0.3$
 2: $\lambda \leftarrow \lambda_{\text{max}}/N$
 3: Initialize: $responses \leftarrow \{\}$, $embeddings \leftarrow \{\}$, $scores \leftarrow \{\}$, $doneReasons \leftarrow \{\}$,
    $activeModels \leftarrow$ all LLMs
 4: **while** $|activeModels| > 0$ **do**
 5:     **for** each $LLM_i \in activeModels$ **do**
 6:         $(response_i, doneReason_i) \leftarrow getChunk(LLM_i, p, \lambda)$
 7:         $responses[LLM_i] \leftarrow response_i$
 8:         $doneReasons[LLM_i] \leftarrow doneReason_i$
 9:     **end for**
10:     **for** each $LLM_i \in activeModels$ **do**
11:         $embedding_i \leftarrow vectorize(response_i)$
12:         $qScore \leftarrow cosineSimilarity(embedding_i, vectorize(p))$
13:         $interScore \leftarrow InterModelSimilarity(embedding_i, embeddings)$
14:         $scores[LLM_i] \leftarrow \alpha \cdot qScore + \beta \cdot interScore$
15:     **end for**
16:     $bestModel \leftarrow$ argmax$(scores)$
17:     **if** $scores[bestModel] >$ secondBest $+ 0.5$ **and** $doneReasons[bestModel] ==$ “stop” **then**
18:         **return** $responses[bestModel]$
19:     **end if**
20:     $worstModel \leftarrow$ argmin$(scores)$
21:     **if** secondWorstScore $- scores[worstModel] > 0.5$ **then**
22:         Remove $worstModel$ from $activeModels$
23:     **end if**
24: **end while**
25: **return** $responses[argmax(scores)]$
```

4.3 LLM-MS MAB (Multi-Armed Bandit Algorithm)

The Multi-Armed Bandit (MAB) approach treats each model as an arm of a bandit problem, pulling tokens by balancing exploration and exploitation.

4.3.1 Algorithm Description

Tokens are not pre-allocated but pulled one at a time to models selected by the UCB1 policy. The reward for each token allocation is computed based on semantic similarity with the query and consensus among models.

The exploration coefficient γ decreases as tokens are consumed, shifting priority towards exploitation of well-performing models. Models with persistently low rewards naturally receive fewer tokens and are phased out.

4.3.2 Pseudocode

Algorithm 2 Multi-Armed Bandit Algorithm (MAB) with UCB1

```
    Input: Set of LLMs, Query $q$, Token Budget $\lambda_{\text{max}}$
    Output: Best Model Response
 1: Initialize: $rewards_i = 0, pulls_i = 0$ for each model
 2: for each round do
 3:     for each model $i$ do
 4:         $UCB_i = \frac{rewards_i}{pulls_i} + \gamma \cdot \sqrt{\frac{2 \cdot \ln(totalPulls)}{pulls_i}}$
 5:     end for
 6:     Choose model $i$ with maximum $UCB_i$
 7:     Generate next token, update response
 8:     Compute reward:
 9:     $r = \alpha \cdot sim(query, response) + \beta \cdot avg\_inter\_model\_similarity$
10:     Update $rewards_i, pulls_i$
11:     $\gamma = 0.3 \cdot \left(1 - \frac{usedTokens}{\lambda_{\text{max}}}\right)$
12:     if termination condition met then
13:         break
14:     end if
15: end for
16: return response from model with highest reward
```

Chapter 5

Application Layer

5.1 Landing Page and Main Query Interface 27
5.2 Session Management 27
5.3 Algorithm Selection and Settings 28
5.4 Model Selection and Chat Interface 29
5.5 Contextual Memory and Mobile Layout 29

5.1 Landing Page and Main Query Interface

The platform presents users with a clean and intuitive landing page that serves as the primary gateway for submitting queries. Here, users can effortlessly input their questions and select from available model orchestration strategies, streamlining the interaction process. The interface emphasizes usability and accessibility, guiding users through query submission while providing clear options for configuring inference behavior. This is exemplified in Figure 5.1, which depicts the landing page and query input interface.

5.2 Session Management

To support continuous and organized user interactions, the platform incorporates a comprehensive session management sidebar. This feature allows users to maintain, revisit, and manage ongoing and historical chat sessions efficiently. Users are empowered to ini-

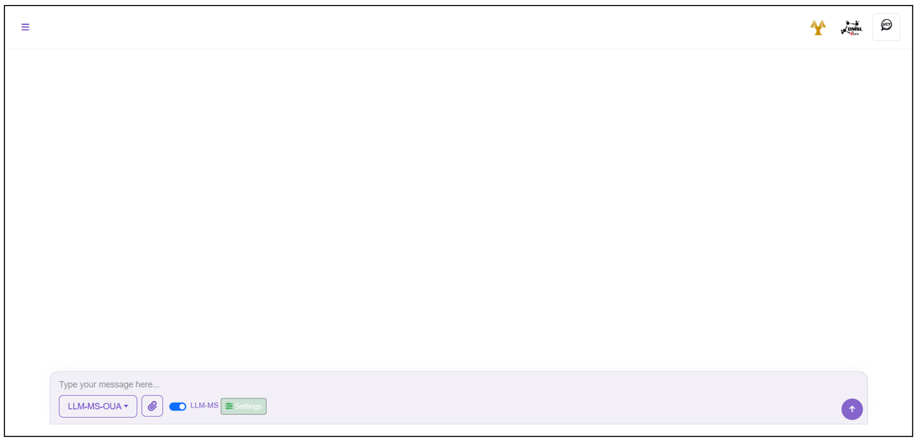


Figure 5.1: Landing page of the platform

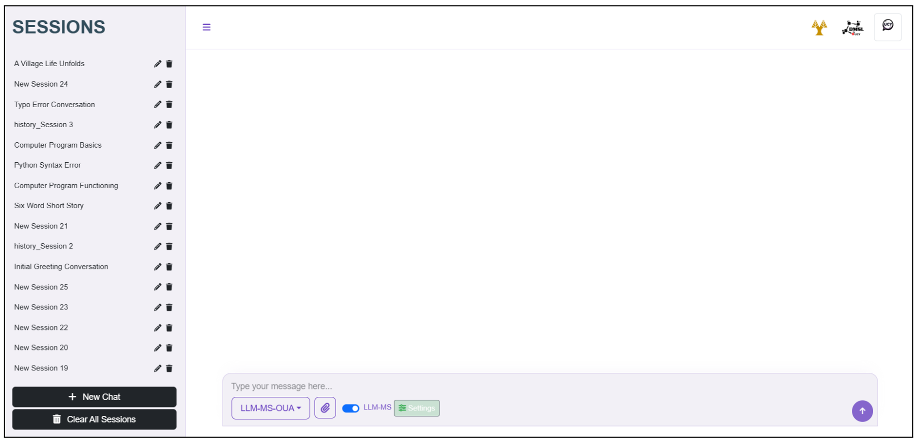
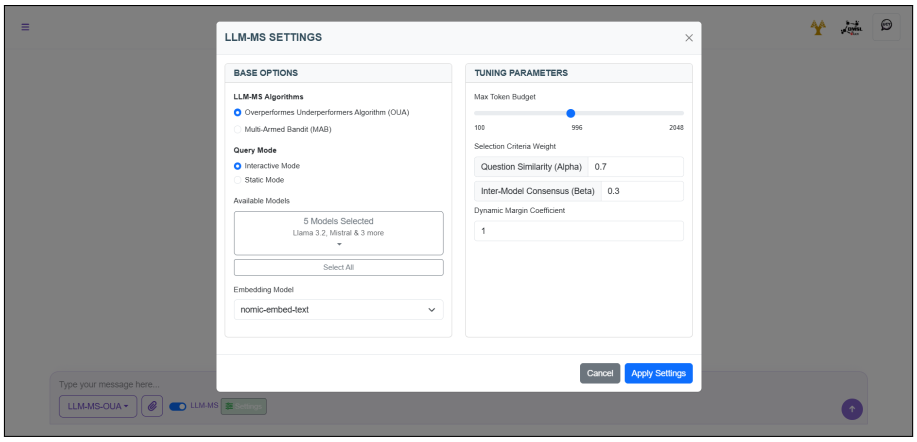


Figure 5.2: The sessions sidebar panel

tiate new conversations or clear prior sessions, facilitating structured multi-turn dialogues and enhancing the overall user experience through persistent conversational context. The sessions sidebar is illustrated in Figure 5.2.

5.3 Algorithm Selection and Settings

A dedicated settings panel empowers users to tailor the platform's inference behavior to their specific requirements. Within this panel, users can select between the two principal orchestration algorithms, the Overperformers–Underperformers Algorithm (OUA) and the Multi-Armed Bandit (MAB), each offering strategies for dynamic model selection and token allocation. Additionally, users can parameters such as token budgets, similarity weighting, and enable or disable individual models, thereby enhancing customization of the multi-model inference process. This settings interface is shown in Figure 5.3.


LLM-MS SETTINGS

BASE OPTIONS

LLM-MS Algorithms

[✓ Overperformes Underperformers Algorithm (OUA)]

Multi-Armed Bandit (MAB)

Query Mode

[✓ Interactive Mode]

Static Mode

Available Models

5 Models Selected
Llama 3.2, Mistral & 3 more

Embedding Model: nomic-embed-text

TUNING PARAMETERS

Max Token Budget

100 996 2048

Selection Criteria Weight

Question Similarity (Alpha): 0.7

Inter-Model Consensus (Beta): 0.3

Dynamic Margin Coefficient: 1

Figure 5.3: Settings panel

5.4 Model Selection and Chat Interface

The platform supports flexible querying modalities, allowing users to interact either with single models directly or through the advanced multi-model orchestration framework. The chat interface is designed to provide real-time, token-by-token streaming of model-generated responses, offering a highly interactive experience. Integration with retrieval-augmented generation (RAG) further enriches response relevance by grounding answers in user-provided documents, thereby improving contextual accuracy and informativeness. Figure 5.4 shows the model selection dropdown with multi-model search disabled, highlighting available individual models. The chat interface for a single model conversation is demonstrated in Figure 5.5, while Figure 5.6 illustrates multi-model sequential participation within the same session. Retrieval-Augmented Generation in action is depicted in Figure 5.7, and the multi-model response comparison interface is presented in Figure 5.8.

5.5 Contextual Memory and Mobile Layout

To preserve conversational coherence, the platform maintains session context through hierarchical summarization. This enables models to generate contextually relevant responses that reflect the dialogue history. Moreover, the design adapts to varying screen sizes, providing the user experience on both desktop and mobile devices. Figure 5.9 illustrates contextual memory retention, and Figures 5.10 demonstrate the responsive mobile layout.

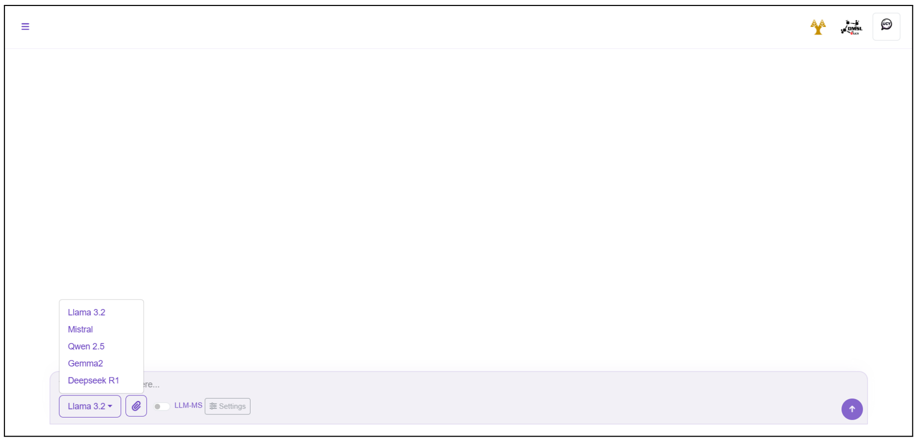
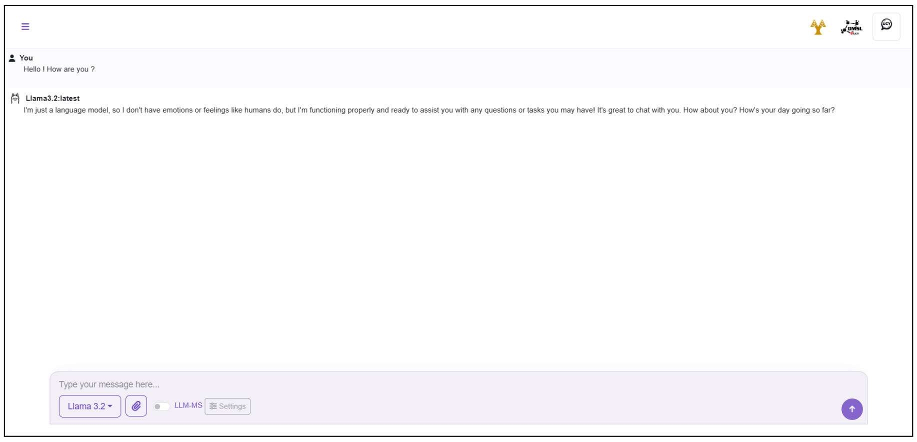


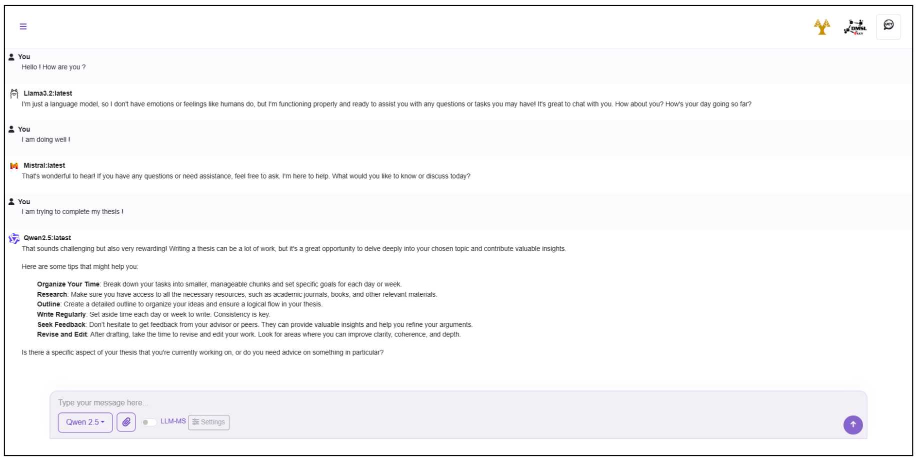
Figure 5.4: Model selection dropdown with the multi-model search (LLM-MS) disabled, showing available individual models.



You
Hello ! How are you ?

Llama3.2:latest
I'm just a language model, so I don't have emotions or feelings like humans do, but I'm functioning properly and ready to assist you with any questions or tasks you may have! It's great to chat with you. How about you? How's your day going so far?

Figure 5.5: Sample chat interface displaying a conversation with a single model (Llama 3.2).



You
Hello ! How are you ?

Llama3.2:latest
I'm just a language model, so I don't have emotions or feelings like humans do, but I'm functioning properly and ready to assist you with any questions or tasks you may have! It's great to chat with you. How about you? How's your day going so far?

You
I am doing well !

Mistral:latest
That's wonderful to hear! If you have any questions or need assistance, feel free to ask. I'm here to help. What would you like to know or discuss today?

You
I am trying to complete my thesis !

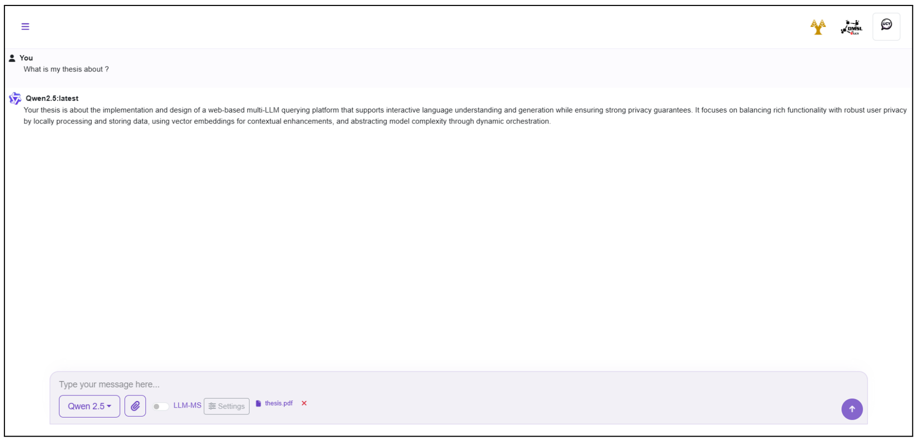
Qwen2.5:latest
That sounds challenging but also very rewarding! Writing a thesis can be a lot of work, but it's a great opportunity to delve deeply into your chosen topic and contribute valuable insights.

Here are some tips that might help you:

Organize Your Time: Break down your tasks into smaller, manageable chunks and set specific goals for each day or week.
Research: Make sure you have access to all the necessary resources, such as academic journals, books, and other relevant materials.
Outline: Create a detailed outline to organize your ideas and ensure a logical flow in your thesis.
Write Regularly: Set aside time each day or week to write. Consistency is key.
Seek Feedback: Don't hesitate to get feedback from your advisor or peers. They can provide valuable insights and help you refine your arguments.
Revise and Edit: After drafting, take the time to revise and edit your work. Look for areas where you can improve clarity, coherence, and depth.

Is there a specific aspect of your thesis that you're currently working on, or do you need advice on something in particular?

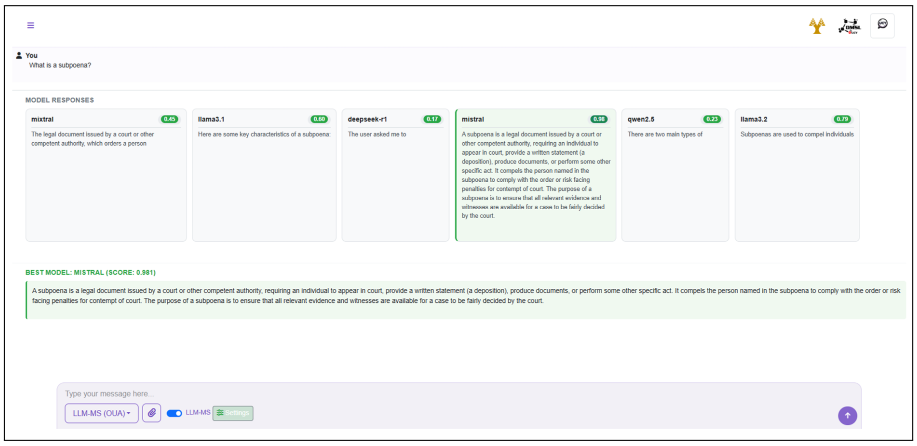
Figure 5.6: Sample chat interface where different models participate sequentially within the same session.



You
What is my thesis about ?

Qwen2.5:latest
Your thesis is about the implementation and design of a web-based multi-LLM querying platform that supports interactive language understanding and generation while ensuring strong privacy guarantees. It focuses on balancing rich functionality with robust user privacy by locally processing and storing data, using vector embeddings for contextual enhancements, and abstracting model complexity through dynamic orchestration.

Figure 5.7: Example RAG



You
What is a subpoena?

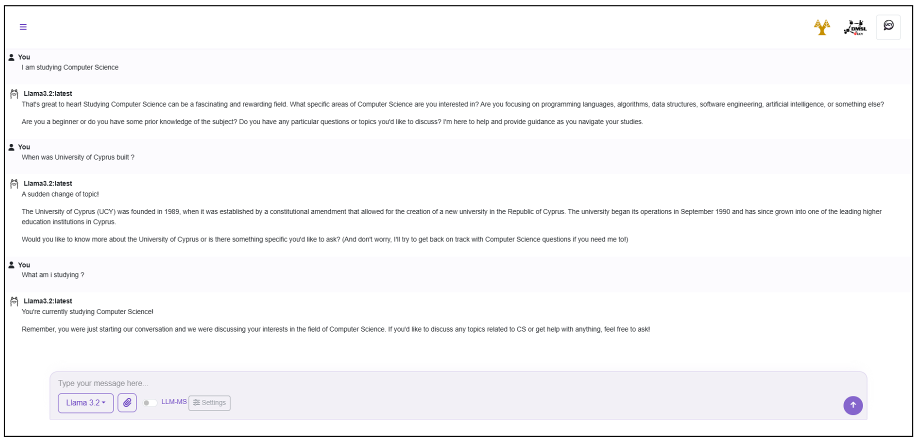
MODEL RESPONSES

mixtral 0.45	llama3.1 0.60	deepseek-r1 0.17	mistral 0.98	qwen2.5 0.23	llama3.2 0.79
The legal document issued by a court or other competent authority, which orders a person	Here are some key characteristics of a subpoena:	The user asked me to	A subpoena is a legal document issued by a court or other competent authority, requiring an individual to appear in court, provide a written statement (a deposition), produce documents, or perform some other specific act. It compels the person named in the subpoena to comply with the order or risk facing penalties for contempt of court. The purpose of a subpoena is to ensure that all relevant evidence and witnesses are available for a case to be fairly decided by the court.	There are two main types of	Subpoenas are used to compel individuals

BEST MODEL: MISTRAL (SCORE: 0.981)

A subpoena is a legal document issued by a court or other competent authority, requiring an individual to appear in court, provide a written statement (a deposition), produce documents, or perform some other specific act. It compels the person named in the subpoena to comply with the order or risk facing penalties for contempt of court. The purpose of a subpoena is to ensure that all relevant evidence and witnesses are available for a case to be fairly decided by the court.

Figure 5.8: Interface showing parallel responses



You
I am studying Computer Science

Llama3.2:latest
That's great to hear! Studying Computer Science can be a fascinating and rewarding field. What specific areas of Computer Science are you interested in? Are you focusing on programming languages, algorithms, data structures, software engineering, artificial intelligence, or something else?

Are you a beginner or do you have some prior knowledge of the subject? Do you have any particular questions or topics you'd like to discuss? I'm here to help and provide guidance as you navigate your studies.

You
When was University of Cyprus built ?

Llama3.2:latest
A sudden change of topic!

The University of Cyprus (UCY) was founded in 1989, when it was established by a constitutional amendment that allowed for the creation of a new university in the Republic of Cyprus. The university began its operations in September 1990 and has since grown into one of the leading higher education institutions in Cyprus.

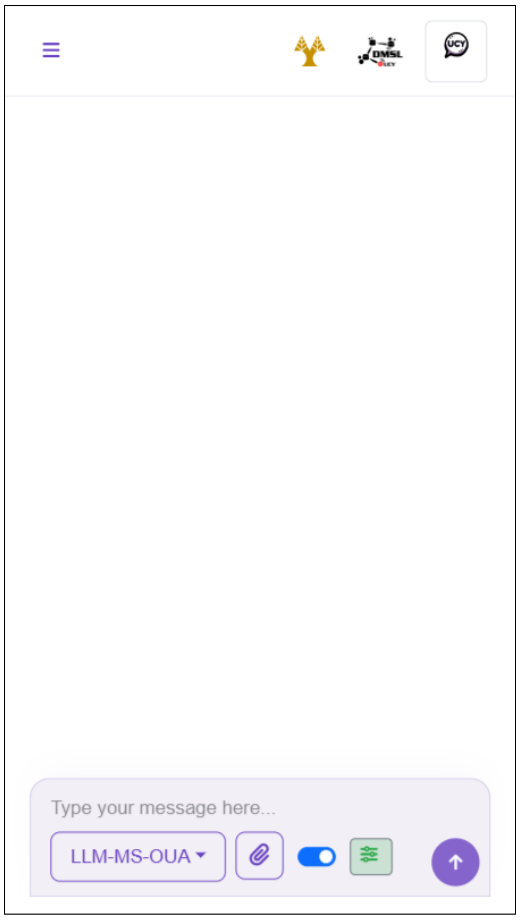
Would you like to know more about the University of Cyprus or is there something specific you'd like to ask? (And don't worry, I'll try to get back on track with Computer Science questions if you need me to!)

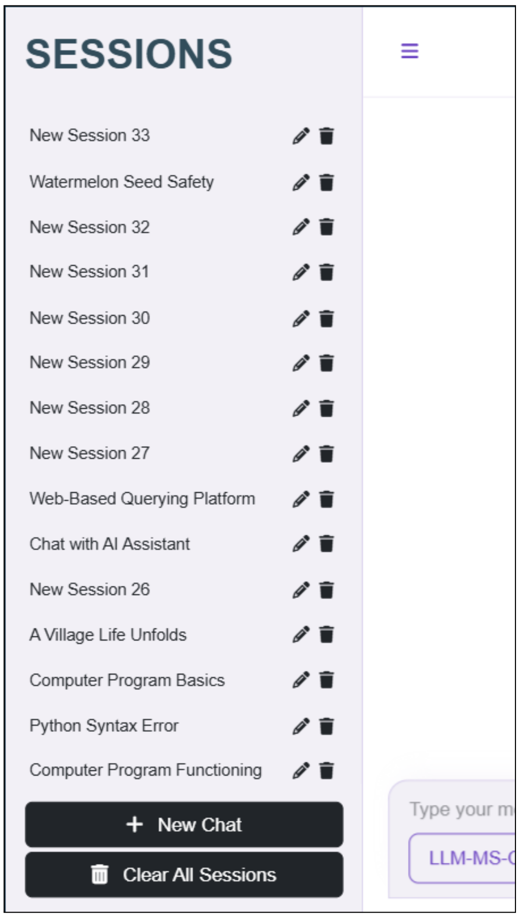
You
What am i studying ?

Llama3.2:latest
You're currently studying Computer Science!

Remember, you were just starting our conversation and we were discussing your interests in the field of Computer Science. If you'd like to discuss any topics related to CS or get help with anything, feel free to ask!

Figure 5.9: Sample chat




SESSIONS

New Session 33
Watermelon Seed Safety
New Session 32
New Session 31
New Session 30
New Session 29
New Session 28
New Session 27
Web-Based Querying Platform
Chat with AI Assistant
New Session 26
A Village Life Unfolds
Computer Program Basics
Python Syntax Error
Computer Program Functioning

Figure 5.10: Mobile layout

Chapter 6

Platform Querying Process

6.1 Query Lifecycle 33
6.2 Preprocessing and Embedding Generation 34
6.3 Dynamic Model Selection and Token Allocation 34
6.4 Response Aggregation and Delivery 35
6.5 Context and Session Management 36
6.6 User Interface Overview 36

6.1 Query Lifecycle

A typical query within the platform undergoes a structured multi-stage lifecycle designed to ensure efficiency, adaptability, and modularity. The lifecycle begins at the user interface, where a query, optionally accompanied by file uploads, is submitted via a web client. The backend pipeline then orchestrates this input through several sequential stages: preprocessing, context enhancement via retrieval, dynamic model selection and token allocation, partial and complete response generation, aggregation, and session-level summarization. This process emphasizes iterative refinement. Rather than forwarding queries to a single model, the system evaluates them within a pool of available language models, each with distinct capabilities. By treating query handling as a multi-step interaction instead of a single static pass, the platform flexibly allocates computational resources and token budgets

while optimizing for answer quality. This architectural choice is particularly important when working with models that exhibit different alignment behaviors, inference costs, or prompt sensitivities.
Furthermore, by leveraging both semantic embeddings and historical interaction patterns, the system constructs highly contextualized prompts and prioritizes continuity across conversation turns. This lifecycle supports not only high-quality single-turn responses but also rich, coherent multi-turn user experiences.

6.2 Preprocessing and Embedding Generation

Upon query submission, the initial step is preprocessing. When users upload external files-such as PDFs, DOCs, or images containing embedded text-these documents are parsed using Python libraries (e.g., PyMuPDF, pdfminer, pytesseract). The extracted text is then segmented into semantically coherent chunks.
Both these text segments and the user query are encoded into fixed-size vector representations using a pre-trained embedding model, such as mxbai-embed-large [87] or a variant of SentenceTransformers such as Sentence-BERT [88]. The resulting embeddings are stored or queried within a vector database, specifically ChromaDB [81] which leverages FAISS [89], which facilitates efficient similarity search.
During the retrieval phase, the system employs cosine similarity to identify the top-k document chunks most relevant to the query. These retrieved chunks are incorporated into the prompt, either prepended or appended, during model inference to enable Retrieval-Augmented Generation (RAG) [90]. This hybrid approach ensures that generated responses are informed by both the static knowledge of the language models and the dynamic, user-specific document context.

6.3 Dynamic Model Selection and Token Allocation

Unlike conventional systems that route queries to a fixed model, the platform employs intelligent selection through two primary strategies: the Overperformers–Underperformers Algorithm (OUA) and the Multi-Armed Bandit (MAB) approach building on recent surveys [75]. Both methods are designed for real-time, incremental evaluation and resource allocation.

The OUA algorithm initially distributes a fraction of the total token budget (e.g., 2048 tokens) equally among all candidate models. Each model generates partial responses in a round-robin fashion. These outputs are embedded and scored by computing cosine similarity against the original query. Models that consistently underperform-due to incoherence or divergence-are pruned from further consideration. Conversely, top-performing models receive additional tokens in subsequent rounds. The process terminates when a model exceeds a predefined similarity threshold or when the token budget is exhausted.
In contrast, the MAB algorithm models each candidate model as an arm of a multi-armed bandit. Using the Upper Confidence Bound (UCB1) [91] policy, it balances exploration (sampling less-tested models) with exploitation (prioritizing promising ones). Token allocations are dynamically adjusted based on accumulated reward, defined as a weighted sum of similarity to the query and agreement with other models:

$$\text{reward} = \alpha \cdot \text{sim}(\text{query}, \text{response}) + \beta \cdot \text{average inter-model similarity} \tag{6.1}$$

This approach enables the system to efficiently converge on the optimal model while minimizing token wastage.

6.4 Response Aggregation and Delivery

After candidate responses are generated, the platform aggregates them based on similarity scores and generation quality. When multiple high-performing models produce outputs, the responses are re-ranked and either the top-ranked answer is selected.
The platform supports real-time streaming of partial outputs, allowing users to observe responses as they are progressively generated. This functionality is implemented through asynchronous API communication between the backend, running the Ollama daemon, and the web interface, built with Flask and mod_wsgi. This design enhances transparency and user interactivity.
Furthermore, token consumption and inference latency are logged for traceability, facilitating subsequent analysis and optimization of system performance in future sessions.

6.5 Context and Session Management

Modern large language model applications increasingly require continuity, especially in chat-based interfaces. This platform includes a session management module that tracks historical exchanges to maintain conversational coherence. To prevent context overflow during extended interactions, the system employs hierarchical summarization techniques that condense earlier messages into concise semantic summaries while preserving essential meaning.

User privacy is a central design principle. All conversation history and session metadata are stored locally on the client using browser-based session storage. No raw user inputs, file contents, or generated messages are retained on the server after the processing.

To support contextual enhancements such as Retrieval-Augmented Generation (RAG), all uploaded documents and conversation summaries are converted into vector embeddings on the server side. These embeddings are stored temporarily in a vector database instance (ChromaDB) running within an isolated read-only Docker container. The embeddings exist solely for the duration of the query-response cycle and are discarded immediately after response delivery or session expiration. No long-term persistence of user-derived data is maintained.

This architecture balances rich functionality with strong privacy guarantees. It enables document-grounded querying, follow-up referencing, and contextual personalization while ensuring that user-specific data remains confined within the secure scope of the active containerized environment.

6.6 User Interface Overview

The platform features a clean, intuitive user interface designed to facilitate seamless interaction with the multi-model querying system. The UI is structured around several key components that correspond closely with the platform's querying and orchestration workflow:

- **Landing Page and Query Input:** Users begin at the landing page (see Figure 5.1), where queries are entered into a text input area. This panel includes controls for selecting orchestration algorithms, toggling the multi-model search (LLM-MS), and uploading supporting files.

- **Session Management:** A sidebar lists saved chat sessions, allowing users to create new sessions, edit existing ones, or clear history (see Figure 5.2). This facilitates organized, multi-turn interactions over time.
- **Model Selection and Settings:** The platform offers flexible model selection via a dropdown menu that supports individual models or multi-model orchestration (LLM-MS). Users can configure model orchestration strategies and tune parameters in the settings panel (see Figures 5.4 and 5.3).
- **Chat Interface and Response Streaming:** Conversations occur in a chat-style interface where users input queries and receive real-time streamed responses from models. The UI supports single-model chats (see Figure 5.5) as well as multi-model sequential responses within a shared session (see Figure 5.6).
- **Retrieval-Augmented Generation (RAG):** Users can upload documents to supplement queries. The system retrieves relevant content dynamically to inform responses (see Figure 5.7).
- **Multi-Model Response Comparison:** When enabled, the interface displays parallel responses from multiple models with quality scores, highlighting the best answer selected by the orchestration algorithm (see Figure 5.8).
- **Contextual Memory and Continuity:** The UI maintains conversational context across turns, enabling models to remember prior inputs and provide coherent answers throughout the session (see Figure 5.9).
- **Mobile Responsiveness:** The platform adapts gracefully to smaller screens, offering a mobile-optimized layout for querying and session management (see Figure 5.10).

Together, these elements create a user-friendly, feature-rich interface that abstracts the complexity of multi-model orchestration, supports rich user interactions, and enhances the overall usability and accessibility of the platform.

Chapter 7

Implementation Details

7.1 Technologies Used 38

7.2 Data Flow and Interaction Between Components 39

7.3 User Interface Features 40

7.1 Technologies Used

Building on a layered, full-stack design, our prototype uses the following concrete components:

- **Flask (v2)** for the REST API that receives user queries and orchestrates downstream processing. All endpoints-including query intake, file ingestion, and health checks-are defined as modular Blueprints.
- **Ollama Daemon (v0.4.5)** as the local LLM runtime. Ollama exposes a streaming REST interface: each model request returns Server-Sent Events (SSE), so we can forward token batches to the client immediately upon generation.
- **ChromaDB** to store and index 1536-dimensional text embeddings. Cosine similarity with an HNSW index is used to retrieve the top-*k* document chunks in sub-millisecond time, powering our RAG pipeline [81].

- **Apache + mod_wsgi** for production hosting, fronting the Flask application to leverage mature process management and zero-downtime restarts.
- **Docker** containers-one per component (Flask, Ollama, ChromaDB)-to guarantee environment parity, simplify deployment, and isolate dependencies.
- **NVIDIA SMI** for real-time GPU monitoring, allowing us to track VRAM usage during concurrent inference and ensure smooth operation under load.

7.2 Data Flow and Interaction Between Components

Figure 7.1 depicts our elements that are involed in the data flow:

1. *Query Submission:* The web client issues a POST to api endpoint. Any attached files are sent in parallel to our ingestion endpoint.
2. *Parsing & Embedding:* Uploaded documents are parsed (PDF, TXT, DOCX) into text chunks. Each chunk is embedded and upserted into ChromaDB.
3. *Context Retrieval:* Upon completing the query embedding, we issue a vector search to ChromaDB, retrieving the top-k semantically relevant chunks.
4. *Prompt Construction:* The system builds an enhanced prompt by combining the user's query with retrieved context.
5. *Orchestration Decision:* The orchestration engine (using either LLM-MS OUA or LLM-MS MAB) evaluates partial outputs, allocates token budgets, and determines which models Ollama should invoke.
6. *Inference via Ollama:* The selected models are invoked in parallel through the Ollama daemon. Token budgets are enforced, and partial responses are streamed back.
7. *Streaming to Client & Caching:* As SSE batches arrive from Ollama, the Flask layer forwards them over a StreamingObject to the browser. Token usage, model scores, and session state are cached locally for continuity and analysis.

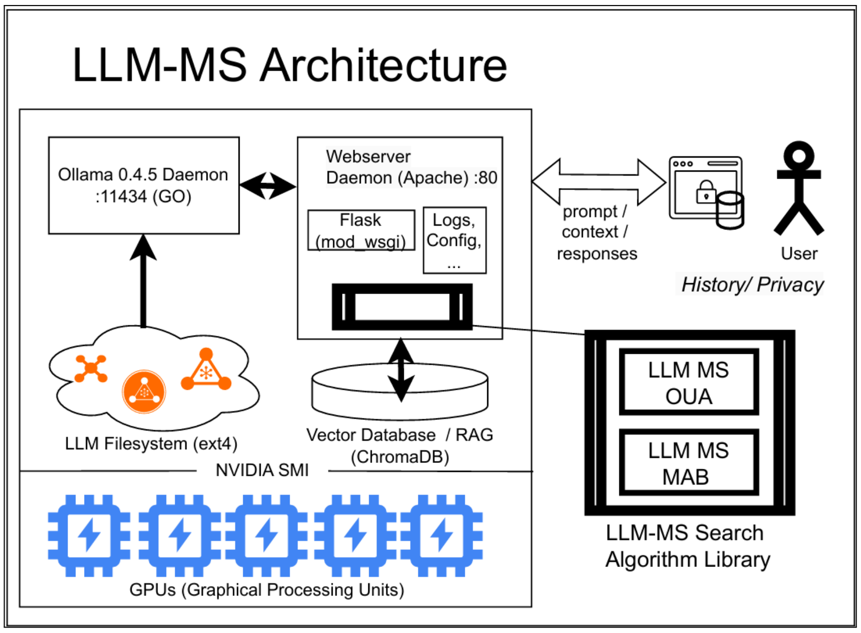

LLM-MS Architecture
Ollama 0.4.5 Daemon :11434 (GO)
Webserver Daemon (Apache) :80
Flask (mod_wsgi)
Logs, Config, ...
prompt / context / responses
User
History/ Privacy
LLM Filesystem (ext4)
Vector Database / RAG (ChromaDB)
NVIDIA SMI
LLM MS OUA
LLM MS MAB
LLM-MS Search Algorithm Library
GPUs (Graphical Processing Units)

Figure 7.1: System architecture and data flow overview.

7.3 User Interface Features

Our front-end abstracts complexity behind a clean, responsive chat interface:

- **Real-Time Streaming:** Token-by-token display via SSE, giving users the feeling of an unfolding conversation.
- **Upload Panel:** Support for PDFs, text, and DOCX; client-side parsing ensures the main thread remains responsive.
- **Model Routing Transparency:** An optional overlay shows which models were selected, their similarity scores, and token allocations-empowering power users without cluttering the main view.
- **Session History & Summarization:** Conversation turns are stored in browser localStorage. After every five messages, earlier context is replaced by an AI-generated summary to keep performance optimal.
- **Mobile Optimization & Accessibility:** Fully responsive layout, with ARIA roles and keyboard navigation. The UI adapts gracefully to small screens, ensuring consistency across devices.

Chapter 8

Experimental Evaluation

8.1 Experimental Setup **41**

8.2 Metrics **44**

8.3 Results **46**

8.3.1 Average Reward per Model 46

8.3.2 Average F1 Score per Model 46

8.3.3 Average Reward to Tokens Ratio per Model 46

8.4 Analysis **47**

8.1 Experimental Setup

The evaluation of the multi-model querying platform focused on verifying the functionality and comparative efficiency of the implemented orchestration strategies-OUA and MAB-against a fixed single-model baseline. The emphasis was on measuring answer quality, token efficiency, and system responsiveness in a realistic but controlled setting using locally deployed language models.

Hardware Environment

All tests were conducted on a virtual server hosted by the Data Management Systems Laboratory (DMSL) at the University of Cyprus. The virtualized environment runs on VMware

with an allocated **Intel Xeon Gold 6230 CPU** featuring 40 virtual cores at 2.10 GHz. The server is provisioned with **98 GB of RAM** to support large-scale model inference and data processing tasks.
A dedicated **NVIDIA Tesla V100 GPU** with 32 GB of VRAM is attached to the virtual machine via PCI passthrough, providing hardware acceleration for deep learning workloads. The GPU operates under NVIDIA driver version 560.35.03 with CUDA 12.6 support.
Storage resources include a 1 TB NVMe SSD mounted at /media/nvme1TB and a 1 TB primary SSD for OS and applications, delivering fast I/O performance necessary for the platform's retrieval and caching mechanisms.
The server runs on **Ubuntu 24.04.2 LTS (Noble)** Linux, with all necessary CUDA and PyTorch dependencies installed. The multi-model inference platform uses **Ollama 0.4.5** for dynamic loading and execution of quantized models, enabling efficient GPU-based inference and token streaming over a REST API.
This virtual server setup offers a scalable and flexible testbed for executing and evaluating multi-LLM orchestration strategies in a controlled environment.

Software Stack

- **Ollama**: Used as the LLM backend, serving quantized GGUF models, and exclusively for embedding generation using models such as nomic-embed-text and mxbai to create vector representations of queries and documents.
- **ChromaDB**: Serves as the vector database for semantic retrieval during Retrieval-Augmented Generation (RAG), enabling fast similarity search over embeddings.
- **Flask**: Implements the backend web server logic, deployed under Apache HTTP Server with mod_wsgi for reliable production use.

Models Evaluated

The evaluation utilized three prominent open-source large language models, selected based on their demonstrated performance across various NLP benchmarks, compatibility with local GPU inference, and representativeness of different model architectures and capabilities:

- **LLaMA 3 8B**: A state-of-the-art transformer-based model developed by Meta, known for its balanced trade-off between performance and computational efficiency. The 8-billion parameter variant provides high-quality natural language understanding and generation while being manageable on a single GPU.
- **Mistral 7B**: A competitive 7-billion parameter model designed for efficiency and robustness in diverse language tasks. Its smaller size compared to LLaMA allows faster inference, making it suitable for resource-constrained environments without significantly sacrificing accuracy.
- **Qwen-2 7B**: Developed by Alibaba DAMO Academy, Qwen-2 is optimized for multilingual reasoning and knowledge-intensive tasks. The 7-billion parameter version balances multilingual capabilities and inference speed, offering complementary strengths to the other evaluated models.

All models were loaded and managed via the Ollama daemon, which supports quantized GGUF model formats optimized for fast local inference. The models shared access to the server's NVIDIA Tesla V100 GPU, enabling parallel execution and dynamic orchestration during multi-model querying. This setup ensured efficient resource utilization while maintaining low latency in real-time interaction scenarios.

Query Types

The evaluation was performed solely using the **TruthfulQA** dataset, a benchmark designed to assess the truthfulness and accuracy of language model responses. This dataset includes a diverse set of challenging questions aimed at exposing false or misleading answers, focusing primarily on factual recall and reasoning.

Execution Modes Compared

Each query from the TruthfulQA dataset was tested under the following inference strategies:

1. **Single-model baseline** – Each query was answered by one model without orchestration.

2. **OUA (Overperformers–Underperformers Algorithm)** – Models were initially allocated equal token budgets; underperforming models were pruned dynamically based on semantic similarity scores.

3. **MAB (Multi-Armed Bandit Algorithm)** – Token allocation among models was adjusted dynamically using a UCB1-based bandit policy guided by a reward function incorporating query relevance and inter-model agreement.

Data Collected

For the evaluation of LLM-MS, data collection focused exclusively on metrics directly relevant to model performance and computational efficiency. The primary data recorded for each experiment included:

- **Total tokens used**: For every query, the number of tokens generated by all participating models was recorded. This metric captures the computational cost required to arrive at a final answer

- **Qualitative accuracy**: Manual evaluation was performed by comparing generated responses against the ground truth and correct answers from the TruthfulQA benchmark. This assessment allowed us to gauge whether the answers produced were not only computationally efficient but also aligned with factual correctness as established by the dataset.

All experiments were performed using the TruthfulQA benchmark on a fixed laboratory testbed, and every metric was calculated according to the definitions and procedures outlined in our methodology section .

8.2 Metrics

The efficiency and effectiveness of LLM-MS were evaluated using a concise set of metrics grounded directly in our experimental design. These metrics reflect both resource usage and answer quality, and their definitions and roles in our evaluation are as follows:

1. Token Usage

Token usage was measured as the number of tokens generated in the final answer for each query. This value represents the computational cost for producing the system's output and was used as a basic metric for comparing efficiency between different model strategies.

2. Answer Quality

The quality of generated answers was evaluated using reward scoring and the F1 metric. For each query, the system generated response was compared against the reference answers provided in the TruthfulQA dataset. Evaluation did not rely on human or manual review, instead, it used a formalized computational approach:

- **Reward Formula**: The reward for each response was computed using a weighted cosine similarity, considering three factors: similarity with the golden answer, similarity with all correct answers, and dissimilarity with known incorrect answers. Specifically, the reward is calculated as the 8.1 formula below is stated
- **F1 Score**: To further quantify answer accuracy, the F1 score was used to measure the overlap between generated responses and the correct answers. This metric accounts for both precision and recall, providing a standard measure of response quality based on the dataset's ground truth.

All answer quality metrics are computed automatically without subjective intervention, ensuring that the evaluation remains objective and reproducible, and strictly grounded in the annotated labels and scoring mechanisms present in the TruthfulQA benchmark

3. Reward Score

The reward score was defined as follows:

$$\text{Reward} = w_1 \, \text{sim}(\text{response, golden}) + w_2 \, \text{sim}(\text{response, correct}) - w_3 \, \text{sim}(\text{response, incorrect}) \tag{8.1}$$

where $w_1 = 1$, $w_2 = 0.5$, $w_3 = 0.5$, and $\text{sim}(\cdot, \cdot)$ represents the cosine similarity between embedding vectors.

4. F1 Score

The F1 score was used to further evaluate the quality of generated responses, measuring the overlap between each model's answer and the correct answers in the dataset. This provided a well-established metric for assessing precision and recall.

In summary, our evaluation of LLM-MS focused on the following set of metrics: total tokens used, qualitative accuracy (F1), reward score. All results are strictly based on these definitions, reflecting only what is reported and validated in our experiments

8.3 Results

The platform was evaluated on the TruthfulQA dataset to assess answer quality and token efficiency. The evaluation compared single-model baselines with the two multi-model orchestration strategies: OUA and MAB. This section presents the quantitative outcomes of our evaluation, comparing single-model baselines with the proposed multi-model orchestration strategies-OUA and MAB-on the TruthfulQA benchmark.

8.3.1 Average Reward per Model

As shown in Figure 8.1, the LLM-MS MAB orchestration strategy achieves the highest average reward over the TruthfulQA dataset. This indicates that MAB effectively allocates tokens to models generating semantically relevant answers, maximizing response quality through dynamic exploration and exploitation.

8.3.2 Average F1 Score per Model

Figure 8.2 illustrates the average F1 scores for each evaluated model and orchestration strategy. The OUA (LLM-MS OUA) algorithm achieves the highest average F1, reflecting its ability to consistently select models producing accurate and coherent responses.

8.3.3 Average Reward to Tokens Ratio per Model

Figure 8.3 shows the average ratio of reward to tokens consumed, highlighting the efficiency of each model and orchestration approach. The OUA algorithm demonstrates

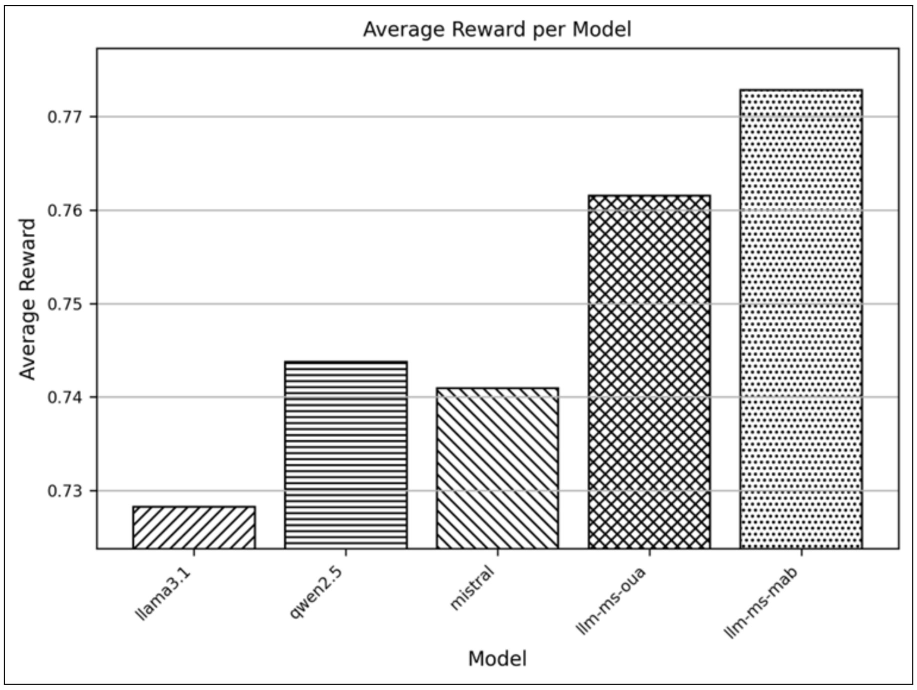


Figure 8.1: Average reward per model over the TruthfulQA dataset

the best trade-off between token usage and answer quality, confirming its efficient use of computational resources.

8.4 Analysis

The experimental results confirm that dynamic multi-model orchestration offers significant benefits over single-model querying in terms of quality. The two implemented strategies, OUA and MAB, exhibited distinct operational behaviors, strengths, and trade-offs, which is analyzed below.

OUA vs. MAB Performance

The Overperformers–Underperformers Algorithm (OUA) performed well in scenarios where one model demonstrated a clear semantic advantage early in the generation process. By distributing tokens evenly at the outset and pruning low-similarity outputs quickly, OUA conserved computational resources. This made it particularly effective in short factual queries, where one model typically dominates in relevance.
In contrast, the Multi-Armed Bandit (MAB) approach was more exploratory. It incrementally adjusted token allocation based on observed rewards, defined as a weighted combi-

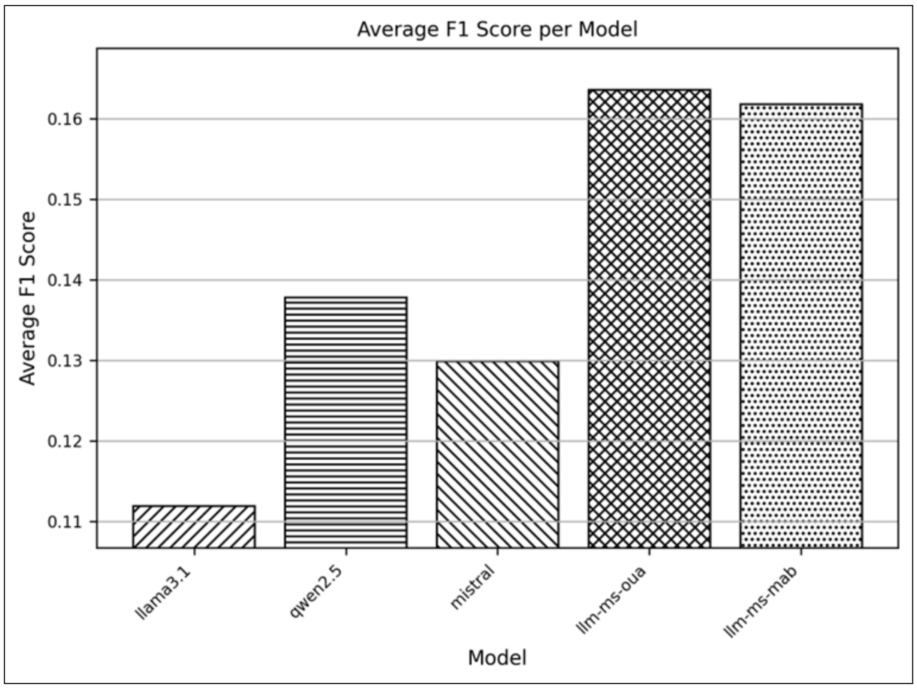


Figure 8.2: Average F1 score per model

nation of query similarity and inter-model agreement. This led to a more balanced use of tokens in ambiguous queries where multiple models provided partial value. Although slightly more computationally intensive due to finer granularity, MAB delivered better consistency in queries requiring broader coverage or complex reasoning.

Impact of Embedding-Based Scoring

Both algorithms relied heavily on embedding-based similarity scores for reward calculation and pruning. This approach performed well for retrieval-based and factual queries, where semantic proximity correlated with answer quality. However, it was less reliable for open-ended or creative prompts, where response quality is more subjective and less aligned with vector similarity metrics.

Trade-Offs in Orchestration

The primary trade-off observed was between early pruning (OUA) and adaptive allocation (MAB). OUA was more efficient in straightforward cases, while MAB was more robust to uncertainty and noise. A hybrid approach could potentially leverage the advantages of both methods.

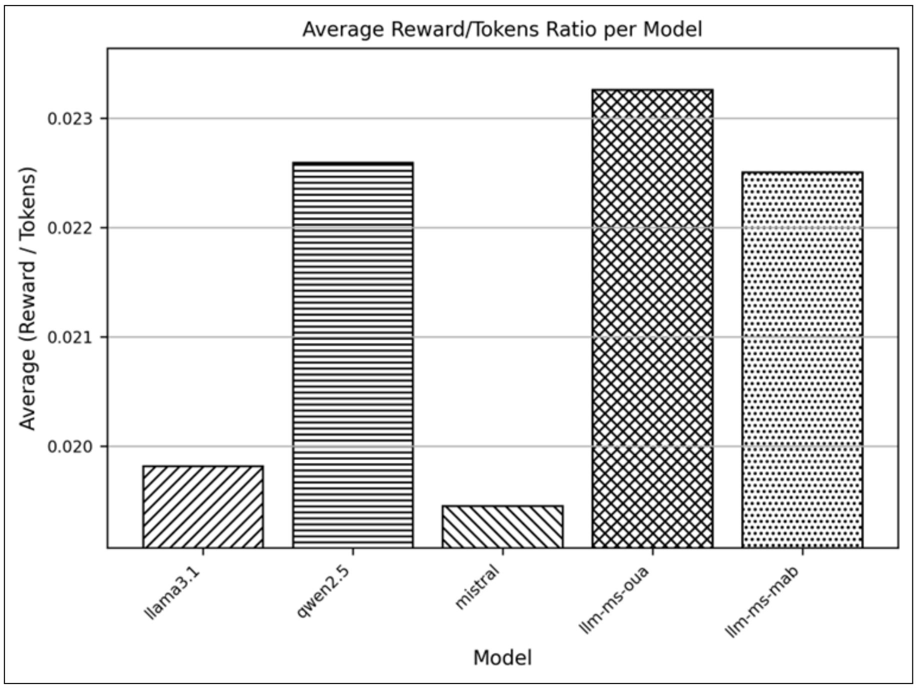


Figure 8.3: Average reward-to-tokens ratio per model

System-Level Implications

From a system perspective, orchestration contributed two key improvements:

- **Better resource utilization:** Token and GPU usage were significantly conserved.
- **Improved user experience:** Streaming partial answers from the selected model led to faster perceived response times and higher-quality interactions.

However, orchestration also introduces overhead in session state management, token tracking, and embedding computation. The evaluation indicated these costs were manageable within the constraints of a single-node deployment using an NVIDIA Tesla V100 GPU. Scaling beyond this setup would require optimization or distributed inference strategies.

Limitations and Edge Cases

The system showed reduced benefits in scenarios where:

- All models failed due to lack of knowledge or misalignment.
- The best model required many tokens before generating a high-quality answer, making early pruning disadvantageous.

- The similarity metric could not distinguish between superficially similar but semantically incorrect answers.

Despite these limitations, the combined results strongly support the conclusion that multi-model orchestration, guided by real-time feedback, can produce better and more efficient answers across a wide range of query types.

Chapter 9

Conclusion

9.1 Summary **51**

9.2 Experimental Insights **52**

9.3 Benefits of a Platform Approach **52**

9.4 Challenges and Limitations **53**

9.5 Future Extensions **53**

9.6 Final Remarks **55**

9.1 Summary

This thesis presented the design, implementation, and evaluation of a modular, dynamic platform for querying multiple large language models (LLMs). The system provides a unified interface through which user queries, optionally accompanied by documents, are processed, routed, and answered via real-time orchestration across a pool of specialized LLMs.

Unlike single-model systems, the platform employs intelligent strategies (Overperformers–Underperformers Algorithm and Multi-Armed Bandit) for model selection and token allocation. It also integrates retrieval-augmented generation (RAG) through a vector database to incorporate context from user-uploaded content. The layered architecture ensures clear separation of concerns, while the front-end offers an intuitive, chat-like interface with

streaming response capabilities.
Experimental evaluation, conducted with open-source models on a local NVIDIA Tesla V100 setup, demonstrated significant improvements in response quality, token efficiency, and system flexibility. Across a diverse set of queries, the platform consistently yielded better or more efficient results than any individual model alone.

9.2 Experimental Insights

Analysis of the orchestration algorithms revealed strengths. The Overperformers–Underperformers Algorithm (OUA) is efficient for factual queries and enables early pruning of underperforming models, minimizing token waste. The Multi-Armed Bandit (MAB) approach is more robust to uncertainty and adapts incrementally using a reward function based on both query similarity and inter-model agreement.
Embedding-based similarity metrics performed well in practical scenarios but exhibit limitations when applied to subjective or creative tasks. The streaming interface, combined with the asynchronous system architecture, enabled partial responses to be delivered in real time, enhancing user experience without significantly increasing latency.
This platform demonstrates how large language models can be treated not as static endpoints but as dynamic components within a larger intelligent orchestration system. This "orchestrator-first" approach represents a paradigm shift in the deployment and evaluation of LLMs in real-world applications.

9.3 Benefits of a Platform Approach

- **Modularity:** New models and retrieval methods can be integrated into the system without major reconfiguration.
- **Efficiency:** Dynamic resource allocation reduces computational waste and optimizes performance and cost.
- **Usability:** Users are abstracted from model internals, as the system manages selection and integration transparently.
- **Flexibility:** Support for document uploads, context management, and session-based

querying enables richer and more personalized interactions.

These benefits underscore the value of platform-centric approaches in the large language model ecosystem: intelligent orchestration can outperform traditional model-centric designs.

9.4 Challenges and Limitations

Despite its strengths, the system faces several practical and architectural challenges:

- **GPU and Memory Constraints:** Even on a Tesla V100, concurrent execution of multiple models can be resource-limited.
- **Token Cost Management:** Although optimized, high token consumption remains a concern for scalability.
- **Scoring Function Generality:** Cosine similarity may not adequately capture answer quality for creative or subjective queries.
- **Real-Time Complexity:** Multi-model orchestration introduces overhead that must be carefully balanced against response latency.
- **Storage Lifecycle Management:** Temporary embedding storage in Docker containers requires careful lifecycle and resource management.

These challenges highlight directions for future work focusing on scalability, precision, and robustness improvements.

9.5 Future Extensions

The platform establishes a strong architectural baseline that can be extended with a number of advanced features aimed at making multi-LLM systems more intelligent, adaptive, and user-aligned:

- **Self-Improving Orchestration:** Introduce reinforcement learning or feedback-driven refinement, enabling the orchestration to learn from user feedback so it gradually favors models that give better answers.

- **Federated and Secure Model Integration:** Allow queries to models hosted in secure, decentralized environments, such as on-premise servers or isolated cloud endpoints, so sensitive models can stay local while still being part of the system [92].
- **Natural Language Configuration Interface:** Provide a user-friendly text box where anyone can type clear instructions, "avoid using slow models," "prioritize our legal model," or "keep responses under 200 words", and the platform automatically interprets these rules, filters out unwanted models, and adjusts output style.
- **Cognitive Routing with Semantic Task Indexing:** Add a simple intent detector (like tagging a request as 'summarize' versus 'fact lookup') and keep a small index of which models are best at each task. When a new question comes in, look up its intent and send it only to the model that's known to handle that kind of job. [93].
- **Multi-Agent Collaboration Framework:** Break complex questions into smaller tasks handled by different workers, for example, one module gathers background info, another figures out how to piece an answer together, and a third double-checks for errors. They can work in sequence or side by side, so even multi-step queries become easier to manage. This approach is similar to what AutoGen [94] and LangGraph [95] use.
- **Game-Theoretic Model Coordination:** Treat each model as a "player" that earns points based on answer quality-track simple metrics (e.g., confidence or correctness) and let models compete or collaborate to pick the best response.
- **Contextual Memory Graphs:** Rather than just storing chat logs in order, build a small in-memory graph that links similar questions and answers. Over time, you can pull in past relevant conversations to help the LLM give a more personalized, consistent reply.
- **Transparent Orchestration Logs:** Show users a simple log: 'We asked Model A first, it got 60% confidence; then we asked Model B, it got 75% and won.' Having that level of transparency is especially important if you're using this in law, banking, or medical settings

These extensions would evolve the platform from a modular querying tool into a comprehensive orchestration intelligence layer, capable of mediating between user needs, LLM capabilities, and dynamic data sources in real time.

9.6 Final Remarks

The landscape of large language models is evolving rapidly, and static deployment of single models cannot keep pace with the diversity of tasks and user expectations. This thesis contributes a working prototype and conceptual framework for a multi-model orchestration platform that is responsive, efficient, and extensible.
By bridging the gap between model diversity and usability, the platform demonstrates that intelligent systems can emerge not only from improved models, but also from better architectural design. As the field progresses, orchestration and integration will become as critical as the models themselves.

Bibliography

[1] 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

[2] OpenAI, "Chatgpt," 2023. [Online]. Available: https://chat.openai.com

[3] ——, "Gpt-2," 2019. [Online]. Available: https://openai.com/research/gpt-2

[4] Anthropic, "Claude," 2023. [Online]. Available: https://claude.ai

[5] MetaAI, "Llama," 2023. [Online]. Available: https://www.llama.com/

[6] R. AI, "A taxonomy of llm architectures," 2024. [Online]. Available: https://re-cinq.com/blog/llm-architectures

[7] 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

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017, pp. 6000–6010. [Online]. Available: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[9] Z. Fu, W. Lam, Q. Yu, A. M. C. So, S. Hu, Z. Liu, and N. Collier, "Decoder-only or encoder-decoder? interpreting language model as a regularized

encoder-decoder," *arXiv preprint arXiv:2304.04052*, 2023. [Online]. Available: https://arxiv.org/abs/2304.04052

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, VIRTUAL, 2020, pp. 1877–1901.

[11] OpenAI, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[12] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, Minneapolis, Minnesota, USA, 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[14] H. Touvron, T. Lavril, G. Izacard, X. Martinet, and M.-A. t. Lachaux, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[15] Anthropic, "Claude 2 model card and evaluations," 2023. [Online]. Available: https://cdn.anthropic.com/100/Claude-2-Model-Card.pdf

[16] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. Le Scao, T. Lavril, T. Wang, T. Lacroix, and W. El Sayed, "Mistral 7B: A 7-billion-parameter language model," *arXiv preprint arXiv:2310.06825*, 2023.

[17] J. Wei, M. Bosma, V. Zhao, K. Guu, A. Yu, B. Lester, N. Du, W. Dakka, Q. Yu, and Q. V. Le, “Finetuned language models are zero-shot learners,” *arXiv preprint arXiv:2109.01652*, 2021, available at https://arxiv.org/abs/2109.01652.

[18] R. K. Taori, M. Mitchell, E. Lee, J. Miller, S. Gururangan, S. Singh, and S. R. Bowman, “Stanford alpaca: An instruction-following model and dataset,” Stanford University, Tech. Rep., 2023, available at https://crfm.stanford.edu/2023/03/13/alpaca.html.

[19] N. Stiennon, L. Ouyang, J. Wu, D. M. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano, “Learning to summarize with human feedback,” in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, Virtual, 2020, pp. 3008–3021.

[20] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, R. Kelcey, B. Miller, E. Simens, A. Askell, P. Yorum, G. Krueger, J. So, R. McElreath, J. Thomason, J. Glaese, W. Weng, T. Kinney, N. de Freitas, J. Leike, and P. Christiano, “Training language models to follow instructions with human feedback,” *arXiv preprint arXiv:2203.02155*, 2022, available at https://arxiv.org/abs/2203.02155.

[21] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, “Training language models to follow instructions with human feedback,” *arXiv preprint arXiv:2203.02155*, 2022.

[22] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, “Scaling laws for neural language models,” *arXiv preprint arXiv:2001.08361*, 2020, available at https://arxiv.org/abs/2001.08361.

[23] J. Hoffmann, S. Borgeaud, A. Mensch, S. Bond-Taylor, E. Hasler, E. Castillo, J. Eckle-Kohler, B. McGraw, J. Clark, J. Ring, A. Carson, R. Leblond, N. Elhage, O. Hubbard, B. McGrew, K. Millican, A. Ramesh, J. Sanchez, J. Zhang, J. Schneider, G. Krueger, L. Engstrom, R. Munos, Y. Shen, L. Scholz, B. Yetman, A. Azaria,

S. Murphy, A. Lewkowycz, M. Rohrbach, J. Nauta, S. van Steenkiste, P. Hashemi, T. White, T. Clarke, A. Alama, S. Otterbacher, A. Koul, S. Hadfield, M. Martens, and D. Amodei, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022, available at https://arxiv.org/abs/2203.15556.

[24] S. Bird, R. Bommasani, T. B. Brown, O. Levy, D. Ruggero, H. L. Saxena, Y. Feng, V. Ng, D. Selsam, and M. Chang, "Emergent behaviors in large language models: A survey," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 889–921, 2022.

[25] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, M. Phillips, J. Romano, R. Robinson, K. Vanommeslaeghe, F. Syed, B. Luke, R. Koncel-Kedziorski, A. Raichuk, P. von Platen, C. Lara, D. Bhatt, P. Liu, Z. Lu, U. Arooj, A. Kamath, H. B. McMahan, L. Zettlemoyer, and J. Dean, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022, available at https://arxiv.org/abs/2204.02311.

[26] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised cross-lingual representation learning at scale," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pp. 8440–8451, 2020.

[27] BigScience Collaboration, T. L. Scao, C. Raffel, A. Roberts, H. Larochelle, A. Lyubetskaya, M. Ott, Z. J. Zhang, Kell, P. Kim, N. Goyal, S. Kim, Ring, Tejas, P. Xie, C. Raffel, A. Roberts, L. Pawlicki, A. Saxena, M. U. Gutmann, X. Zhao, A. Conneau, B. Rozière, D. Britz, Y. Chen, Y. Xue, S. Liu, G. Wenzek, S. Ostadhossieni, F. Guzmán, A. Joulin, and E. Grave, "BLOOM: A 176b-parameter open-access multilingual language model," *arXiv preprint arXiv:2211.05100*, 2022, available at https://arxiv.org/abs/2211.05100.

[28] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *arXiv preprint arXiv:2101.03961*, 2021, available at https://arxiv.org/abs/2101.03961.

[29] ——, “Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity,” 2022. [Online]. Available: https://arxiv.org/abs/2101.03961

[30] T. Dettmers and L. Zettlemoyer, “Llamaquant: 4-bit consistent quantization of large language models,” *arXiv preprint arXiv:2305.14314*, 2023, available at https://arxiv.org/abs/2305.14314.

[31] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter,” in *Proceedings of the 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing (EMNLP Workshop) 2020*, Online, 2020, pp. 1–6.

[32] T. Dettmers, E. Le, and L. Zettlemoyer, “Llma: Low latency model acceleration for large language models,” *arXiv preprint arXiv:2304.07121*, 2023, available at https://arxiv.org/abs/2304.07121.

[33] W. X. Zhao, K. Zhou, J. Li, T. Tang, and X. t. Wang, “A survey of large language models,” *arXiv preprint arXiv:2303.18223*, 2023.

[34] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, “Large language models: A survey,” *arXiv preprint arXiv:2402.06196*, 2024.

[35] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, “Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, Minneapolis, Minnesota, USA, 2019, pp. 787–804. [Online]. Available: https://aclanthology.org/N19-1423

[36] Z. Lin, Y. Liu, Z. Luo, J. Gao, and Y. Li, “Momq: Mixture-of-experts enhances multi-dialect query generation across relational and non-relational databases,” 2024. [Online]. Available: https://arxiv.org/abs/2410.18406

[37] N. Du, Y. Huang, A. M. Dai, S. Tong, and D. t. Lepikhin, “Glam: Efficient scaling of language models with mixture-of-experts,” in *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2022.

[38] MistralAI, “Mistral ai,” 2023. [Online]. Available: https://mistral.ai/

[39] L. Xue, N. Constant, A. Roberts, J. Choi, C. Hawthorne, X. Song, Q. V. Le, and C. Raffel, “mt5: A massively multilingual pre-trained text-to-text transformer,” *arXiv preprint arXiv:2010.11934*, 2021, available at https://arxiv.org/abs/2010.11934.

[40] J.-B. Alayrac, A. Verzuh, J. Donahue, X. Chen, M. Smith, Z. Akata, C. Feichtenhofer, A. Kirillov, G. Ostrouchov, A. Pundir, E. Tzeng, A. Steiner, E. Hafen, R. Han, Y. Li, G. Cardoso, B. Vincent, M. Simon, Y. Lin, Y. Zhang, A. Gilberti, A. Angelova, L. Fei-Fei, T. Sainath, A. Torralba, and P. Hasler, “Flamingo: a visual language model for few-shot learning,” in *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, New Orleans, Louisiana, USA, 2022, pp. 24 008–24 023. [Online]. Available: https://arxiv.org/abs/2204.14198

[41] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Brew, H. Plu, C. Ma, M. Xu, Q. Lhoest, and A. M. Rush, “Transformers: State-of-the-art natural language processing,” *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[42] H. Face, “Hugging face,” 2023. [Online]. Available: https://huggingface.co

[43] MosaicML Research, “Mpt-7b: A reference implementation of a gpt-style foundation model,” 2023, available at https://www.mosaicml.com/blog/mpt-7b.

[44] Google, “Vertex-ai,” 2025. [Online]. Available: https://cloud.google.com/vertex-ai/docs

[45] G. AI, “Gemini,” 2023. [Online]. Available: https://ai.google.dev/gemini-api/docs/models/gemini

[46] Google, “Gemini 2.0 flash,” 2025. [Online]. Available: https://deepmind.google/technologies/gemini/flash/

[47] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, pp. 1–24, 2019. [Online]. Available: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[48] A. D. Academy, "Qwen 2.5 7b," 2025. [Online]. Available: https://qwenlm.github.io/blog/qwen2.5/

[49] P. Micikevicius, S. Narang, G. Alben, G. Diamos, E. Elsen, D. Garcia, G. Ginsburg, M. Houston, O. Kuchaiev, K. Venkatesh, and H. Wu, "Mixed precision training," *International Conference on Learning Representations (ICLR) Workshop Track*, 2018. [Online]. Available: https://arxiv.org/abs/1710.03740

[50] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2019)*, Washington, D.C., USA, 2019, pp. 1–12. [Online]. Available: https://dl.acm.org/doi/10.1145/3293883.3295713

[51] T. Chen, B. Xu, S. Singh, G. Dudek, O. Polozov, S. Chu, P. Khaitan, and P. Malkin, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016, available at https://arxiv.org/abs/1604.06174.

[52] T. Kudo and J. Richardson, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, 2018. [Online]. Available: https://aclanthology.org/D18-2025

[53] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Le, E. Zhou, R. Pascanu, H. Michalewski, and T. Rocktäschel, "Chain of thought prompting elicits reasoning in large language models," *arXiv preprint arXiv:2201.11903*, 2022, available at https://arxiv.org/abs/2201.11903.

[54] Z. Ji, W. Lee, R. Frieske, A. Yu, Y. Su, T. Xu, D. Ishani, A. King, D. Sahrawat, L. Higgins, M. Reid, M. Wang, J. Devlin, K. Gopalakrishnan, J. Maynez, and E. Pavlick, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–35, 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3565815

[55] P. Liang, R. Bommasani, T. Lee, and D. t. Tsipras, "Holistic evaluation of language models," *arXiv preprint arXiv:2211.09110*, 2022.

[56] T. Authors, "Truthfulqa: A dataset for evaluating ai's truthfulness," 2025. [Online]. Available: https://huggingface.co/datasets/truthfulqa/truthful_qa/tree/main

[57] E. Sheng, K.-W. Chang, P. Natarajan, and N. Peng, "The woman worked as a babysitter: On biases in language generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021, pp. 3409–3436. [Online]. Available: https://aclanthology.org/2021.emnlp-main.267

[58] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019)*, 2019, pp. 3645–3650. [Online]. Available: https://aclanthology.org/P19-1355

[59] 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. [Online]. Available: https://ieeexplore.ieee.org/document/10363447

[60] I. Solaiman, M. Brundage, J. Clark, A. Askell, A. Herbert-Voss, J. Wu, A. Radford, G. Krueger, L. McGuffey, T. Schick, H. Lee, J. Schulman, D. M. Ziegler, S. Jain, A. Shaw, J. Ludwig, and L. Wang, "Release strategies and the social impacts of language models," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 2019, pp. 7839–7846. [Online]. Available: https://www.ijcai.org/proceedings/2019/1091.pdf

[61] G. Marcus and E. Davis, *Rebooting AI: Building Artificial Intelligence We Can Trust*. New York, NY, USA: Vintage, 2022.

[62] E. M. Bender, T. Gebru, A. McMillan-Major, and M. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT '21)*, 2021, pp. 610–623. [Online]. Available: https://doi.org/10.1145/3442188.3445922

[63] N. Bostrom, *Superintelligence: Paths, Dangers, Strategies*. Oxford, UK: Oxford University Press, 2014.

[64] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. B. Brown, D. Song, □. Erlingsson, A. Oprea, and C. Raffel, "Extracting training data from large language models," in *Proceedings of the 30th USENIX Security Symposium (USENIX Security 2021)*, 2021, pp. 2633–2650. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/carlini

[65] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019. [Online]. Available: https://www.nature.com/articles/s42256-019-0048-x

[66] L. Floridi, J. Cowls, M. Beltrametti, R. Chatila, P. Chazerand, V. Dignum, C. Luetge, R. Madelin, U. Pagallo, F. Rossi, B. Schafer, P. Valcke, and E. Vayena, "Translating principles into practices: A data ethics framework," *Philosophy & Technology*, vol. 33, no. 4, pp. 659–695, 2020.

[67] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, Q. Guo, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *ArXiv*, vol. abs/2312.10997, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:266359151

[68] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, T. Yeh, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, Virtual, 2020, pp. 9459–9474.

[69] C. Liu, J. W. Rae, S. Jayakumar, Z. Li, J. Raiman, A. Cassirer, J. Rae, and C. Re, “Compressive transformers for efficient long-range sequence modeling,” *arXiv preprint arXiv:2202.06972*, 2022, available at https://arxiv.org/abs/2202.06972.

[70] T. Dao, T. Tang, Q. V. Le, P. Vu, H. Pham, T. Tran, and T. Pham, “Flashattention: Fast and memory-efficient exact attention with io-awareness,” *arXiv preprint arXiv:2205.14135*, 2022, available at https://arxiv.org/abs/2205.14135.

[71] A. Birhane, V. U. Prabhu, D. Frost, S. Sen, A. Duval, M. Cisse, and T. Gebru, “Multimodal machine learning: An ethical and practical survey,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 11, pp. 11 842–11 851, 2022.

[72] F. Doshi-Velez and B. Kim, “Towards a rigorous science of interpretable machine learning,” *arXiv preprint arXiv:1702.08608*, 2017, available at https://arxiv.org/abs/1702.08608.

[73] M. AI, “Code llama,” 2023. [Online]. Available: https://github.com/facebookresearch/codellama

[74] EleutherAI, “Gpt-neo,” 2021. [Online]. Available: https://www.eleuther.ai/artifacts/gpt-neo

[75] Z. Chen, J. Li, P. Chen, Z. Li, K. Sun, Y. Luo, Q. Mao, D. Yang, H. Sun, and P. S. Yu, “Harnessing multiple large language models: A survey on llm ensemble,” *arXiv preprint arXiv:2502.18036*, 2025.

[76] 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/

[77] 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/

[78] T. Shnitzer, A. Ou, M. Silva, K. Soule, Y. Sun, J. Solomon, N. Thompson, and M. Yurochkin, "Large language model routing with benchmark datasets," 2024. [Online]. Available: https://openreview.net/forum?id=LyNsMNNLjY

[79] Q. Nguyen, D. Hoang, J. Decugis, S. Manchanda, N. Chawla, and K. Doan, "Metallm: A high-performant and cost-efficient dynamic framework for wrapping llms," 07 2024.

[80] 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

[81] Chroma, "Chroma: The ai-native vector database," 2023. [Online]. Available: https://www.trychroma.com

[82] 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

[83] 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. [Online]. Available: https://ieeexplore.ieee.org/document/10493277

[84] Z. Wu, "Bhakti: A lightweight vector database management system for endowing large language models with semantic search capabilities and memory," 2025. [Online]. Available: https://arxiv.org/abs/2504.01553

[85] K. Krasovitskiy, S. Christou, and D. Zeinalipour-Yazti, "Llm-ms: A multi-model llm search engine," in *1st International Workshop on Coupling of Large Language Models with Vector Data Management (LLM+Vector Data), collocated with the*

40th IEEE International Conference on Data Engineering, ser. LLMVDB‘25, Hong Kong SAR, China | May 19 – 23, 2025, May 2025, conference, p. 8 pages. [Online]. Available: https://chatucy.cs.ucy.ac.cy/

[86] T. Wolf, L. Debut, V. Sanh, J. Chaumond, and C. t. Delangue, “Transformers: State-of-the-art natural language processing,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 38–45.

[87] MXBAI, “mxbai-embed-large,” 2024. [Online]. Available: https://ollama.com/library/mxbai-embed-large

[88] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using siamese BERT-networks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019, pp. 3980–3990.

[89] J. Johnson, M. Douze, and H. Jégou, “Faiss: A library for efficient similarity search and clustering of dense vectors,” *arXiv preprint arXiv:1702.08734*, 2017.

[90] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[91] 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. [Online]. Available: https://doi.org/10.1023/A:1013689704352

[92] E. Song and G. Zhao, “Privacy-preserving large language models: Mechanisms, applications, and future directions,” *arXiv preprint arXiv:2412.06113*, 2024.

[93] C. Varangot-Reille, C. Bouvard, A. Gourru, M. Ciancone, M. Schaeffer, and F. Jacquenet, “Doing more with less – implementing routing strategies in llm-based systems: An extended survey,” *arXiv preprint arXiv:2502.00409*, 2024.

[94] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, “Autogen: En-

abling next-gen llm applications via multi-agent conversation," *arXiv preprint arXiv:2308.08155*, 2023.

[95] LangChain Inc., "Langgraph: A controllable framework for agentic applications," 2024. [Online]. Available: https://blog.langchain.dev/langgraph-studio-the-first-agent-ide/