



University of Milano-Bicocca

Department of Informatics, Systems and Communication

Master's degree in Data Science

MALBO: Multi-Agent LLM Bayesian Optimization

Supervisor: Prof. Elisabetta Fersini

Co-supervisor: Prof. Antonio Candelieri, Prof. Francesco Archetti

Master's degree thesis by:

Antonio Sabbatella

n.869435

Academic Year 2024-2025

Abstract

The optimal assignment of Large Language Models (LLMs) to specialized roles in multi-agent systems is a significant challenge, defined by a vast combinatorial search space, expensive black-box evaluations, and an inherent trade-off between performance and cost. Current optimization methods focus on single-agent settings and lack a principled framework for this multi-agent, multi-objective problem.

This thesis introduces MALBO (Multi-Agent LLM Bayesian Optimization), a systematic framework designed to automate the efficient composition of LLM-based agent teams. We formalize the assignment challenge as a multi-objective optimization problem, aiming to identify the Pareto front of configurations between task accuracy and inference cost. The methodology employs multi-objective Bayesian Optimization (MOBO) with independent Gaussian Process surrogate models. By searching over a continuous feature-space representation of the LLMs, this approach performs a sample-efficient exploration guided by the expected hypervolume improvement.

The primary contribution is a principled and automated methodology that yields a Pareto front of optimal team configurations. Our results demonstrate that the Bayesian optimization phase, compared to an initial random search, maintained a comparable average performance while reducing the average configuration cost by over 45%. Furthermore, MALBO identified specialized, heterogeneous teams that achieve cost reductions of up to 65.8% compared to homogeneous baselines, all while maintaining maximum performance. The framework thus provides a data-driven tool for deploying cost-effective and highly specialized multi-agent AI systems.

Contents

1	Introduction	5
1.1	From Monolithic Models to Multi-Agent Systems	5
1.2	The Agent Composition Challenge	5
1.3	The Gap in Existing Research	6
1.4	Proposed Solution: Introducing MALBO	6
1.5	Research Questions and Objectives	8
1.6	Key Contributions	8
1.7	Thesis Outline	9
2	Theoretical background	10
2.1	The Transformer Architecture and Foundations of Large Language Models (LLMs)	10
2.1.1	Input Processing: Embeddings and Positional Encoding .	11
2.1.2	The Self-Attention Mechanism	12
2.1.3	Multi-Head Attention	12
2.1.4	Architectural Variants and Foundational Models	13
2.1.5	From Autocomplete to Assistants	15
2.1.6	Architectural Innovations for Scalability and Efficiency . .	19
2.1.7	The Economics of LLM Inference: Beyond Price-per-Token	19
2.1.8	The Role of Scaling Laws	21
2.1.9	A Statistical Perspective on Modern LLMs	22
2.2	Multi-Agent Systems: Paradigms and Frameworks	24
2.2.1	Definitions and Taxonomies	24
2.2.2	Overview of Open-Source Frameworks	25
2.3	Bayesian Optimization (BO) for Black-Box Functions	26
2.3.1	Gaussian Processes as Surrogate Models	27
2.3.2	Acquisition Functions	28
3	Related works - LLM and Bayesian Optimization	30
3.1	Optimization in the Age of Large Language Models	30
3.2	The Evolution of Bayesian Optimization for LLMs	31
3.2.1	Foundational Applications: Prompt and Instruction Optimization	32

3.2.2	Expanding the Scope: BO for Complex Workflows and Trustworthiness	32
3.3	Limitations of the State of the Art and Motivation for MALBO	35
4	MALBO: Methodology and Mathematical Formulation	37
4.0.1	Problem Description: Optimal LLM Assignment in an Agent Team	37
4.0.2	Vector Representation of LLMs and the Configuration Space	38
4.0.3	Multi-Objective Optimization Problem Formulation	38
4.0.4	The Bayesian Optimization Loop for MALBO	39
4.0.5	From Continuous Space to Discrete Assignment: The Projection Function π	41
5	Experimental Setup	43
5.1	Development and Integration Platform	43
5.2	Evaluation Benchmark: GAIA	44
5.3	Performance Metrics	45
5.4	Optimizer Configuration	46
5.4.1	Initialization Strategy	46
5.4.2	Optimization Loop	46
5.5	Selection of Large Language Models	47
6	Results and Analysis	51
6.1	Evolution and Convergence of the Optimization Process	51
6.1.1	Pareto Front Evolution	51
6.1.2	Granular Analysis of Cost-Performance Trade-offs	53
6.1.3	Analysis of the Acquisition Function’s Behavior	55
6.1.4	Comparison with Homogeneous Baselines	57
6.2	Analysis of Optimal Configurations and LLM Assignments	59
6.2.1	Case Studies of Pareto-Optimal Configurations	59
6.2.2	LLM Assignment Preferences and Patterns	61
6.3	Discussion and Interpretation of Deeper Insights	62
6.3.1	Agent-Level Impact Analysis	62
6.3.2	Granular Feature Analysis	63
7	Conclusion, Limitations, and Future Prospective	65
7.1	Conclusion	65
7.2	Limitations	66
7.3	Future Prospective	66
A	Supplementary Visualizations	76
A.1	2D Pareto Front Evolution	76
A.2	3D Interactive Pareto Front Evolution	77
B	GAIA Benchmark Tasks Used in Evaluation	80

Chapter 1

Introduction

1.1 From Monolithic Models to Multi-Agent Systems

The field of artificial intelligence has undergone a paradigm shift, catalyzed by the advent of Large Language Models (LLMs). Architectures like the Transformer [1] have enabled the creation of foundational models with remarkable capabilities in language understanding, reasoning, and generation. Initially, the focus of this revolution was on scaling these models to be larger and more powerful, treating them as monolithic entities to be prompted for a wide array of tasks.

More recently, the focus has evolved. The first step in this evolution was to enhance the *agentic capabilities* of individual models, equipping them with advanced tool-calling functionalities to interact with external systems. This focus on agentic capabilities has become particularly prominent since the introduction of models like Anthropic’s Claude 3.5, which established new benchmarks for sophisticated, multi-step tool use [2]. The current frontier, however, pushes beyond single-agent execution. Major research labs and the open-source community are now releasing dedicated multi-agent frameworks such as Google’s Agent Development Kit, OpenAI’s Agents SDK, and frameworks like LangGraph designed to orchestrate teams of specialized agents. This paradigm, where complex problems are decomposed and solved collaboratively, marks a significant step towards more autonomous and capable AI systems but also introduces novel challenges in their design and deployment.

1.2 The Agent Composition Challenge

The efficacy of these multi-agent systems is critically dependent on their composition. The central question is no longer just how to prompt a single model, but which LLM to assign to each distinct agent role. An orchestrating ‘manager’

agent may require a model with strong planning skills, a 'tool-using' agent may need proficiency in code generation, and a 'verifier' agent may prioritize factual accuracy.

This assignment task gives rise to a vast combinatorial design space. With a pool of M available LLMs and a team of N agents, the number of possible configurations is M^N . To the best of our knowledge, no systematic or mathematical method for solving this specific problem has been documented in the literature. The selection is typically performed manually, guided by heuristics. This often results in one of two suboptimal strategies: assigning the most capable (and expensive) model from public benchmarks to all roles, or assigning a single, cost-effective model across the board. Both approaches ignore the potential for optimization through specialized, heterogeneous team composition. Navigating this space systematically is hindered by two fundamental obstacles:

1. **The Black-Box Evaluation Problem:** The performance of any given team configuration can only be determined by executing it on a representative task, a process that is computationally expensive and time-consuming. The relationship between a configuration and its outcome is an opaque, black-box function with no accessible analytical form or gradient.
2. **The Multi-Objective Imperative:** A viable solution must balance two inherently conflicting objectives. We aim to **maximize the collective performance** of the agent team on its designated task while simultaneously **minimizing the aggregate operational cost**, which is typically dominated by API inference fees.

This intersection of a combinatorial search space, expensive black-box evaluations, and conflicting objectives defines a challenging new optimization frontier.

1.3 The Gap in Existing Research

The current state of the art in LLM optimization has primarily focused on single-agent or monolithic pipeline settings. Research has yielded powerful techniques for prompt engineering, instruction tuning, and hyperparameter optimization for Retrieval-Augmented Generation (RAG) systems [3], [4], [5]. While these methods are valuable, they do not address the unique challenges posed by multi-agent systems. To the best of our knowledge, the literature still lacks a principled and automated framework for the multi-objective, multi-agent LLM assignment problem. The composition of agent teams is thus often left to manual heuristics, trial-and-error, or intuition, a process that is neither scalable nor guaranteed to find optimal solutions.

1.4 Proposed Solution: Introducing MALBO

This thesis introduces and evaluates **MALBO (Multi-Agent LLM Bayesian Optimization)**, a novel and systematic framework designed to automate the

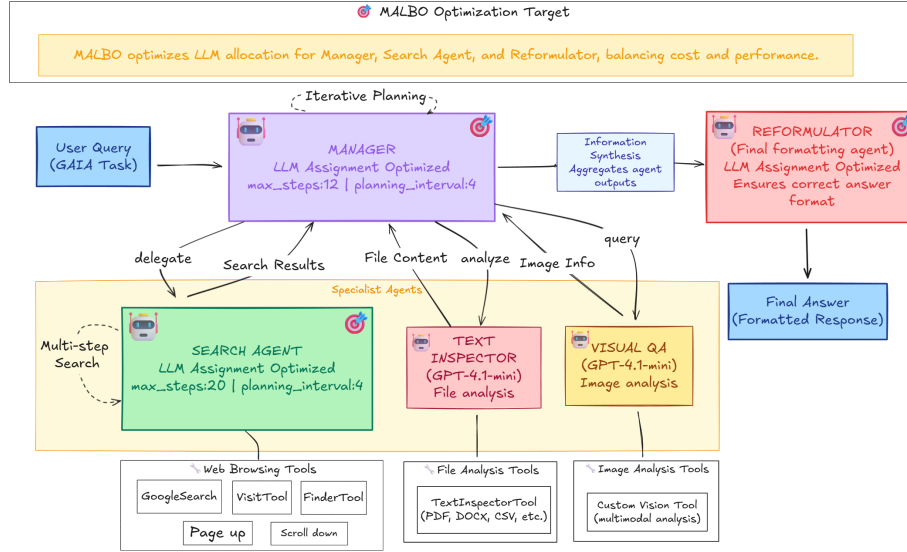


Figure 1.1: Overview of the multi-agent LLM system architecture. The **Manager** orchestrates interactions among specialized agents, the **Search Agent**, **Visual QA**, and **Text Inspector**, each equipped with specific toolsets for web browsing, image analysis, and file analysis, respectively. Both the **Manager** and the **Search Agent** operate in iterative loops: the Manager performs re-planning every 2 steps, while the Search Agent refines its search every 4 steps. The **Manager** aggregates all agent outputs through an information synthesis phase before passing them to the **Reformulator**, which produces the final formatted response. The **MALBO** optimization process dynamically assigns LLM configurations to balance cost and performance across key agents. In the figure, agents are represented by a robot icon at the top left, and those optimized by MALBO are marked with a red target icon at the top right.

efficient composition of LLM-based agent teams (see Fig. 1.1 for a graphical illustration of the DeepResearch Agent Team used in the MALBO validation tests). We formalize the agent assignment challenge as a multi-objective, black-box optimization problem, with the explicit goal of identifying the set of configurations that form the optimal Pareto front between task performance and inference cost.

The MALBO methodology is built on a continuous relaxation of the discrete assignment problem. We first represent each available LLM as a vector in a continuous feature space, capturing its capabilities and costs. A complete team configuration is thus a point in a high-dimensional continuous space. We employ Bayesian Optimization with independent Gaussian Process surrogate models to probabilistically model the two conflicting objectives. The search is guided by the q-Expected Hypervolume Improvement (qEHVI) acquisition function, which intelligently selects new candidate teams to evaluate, ensuring a sample-efficient exploration of the search space. A projection function then maps these "ideal" continuous solutions back to discrete, deployable LLM assignments for evaluation.

1.5 Research Questions and Objectives

To guide our investigation, we formulate the following primary research questions, which address the core challenges and intended outcomes of this work:

1. **RQ1: Formalization.** How can the problem of assigning LLMs to different roles in a multi-agent system be formally cast as a multi-objective, black-box optimization problem suitable for Bayesian Optimization?
2. **RQ2: Efficiency and Optimization.** Can a Bayesian Optimization framework efficiently explore the design space to not only identify a Pareto front, but specifically to improve the **cost-efficiency** of high-performing configurations within a constrained evaluation budget?
3. **RQ3: Structural Insights.** What structural insights can be extracted from the optimization process? Specifically, can this data-driven approach identify which agent roles and which model features are the most **influential drivers** of system performance and cost?

1.6 Key Contributions

The primary contributions of this thesis are delivered across three integrated stages. We begin by introducing a novel formalization of the LLM-to-agent assignment challenge, providing a structured foundation where previously only heuristics existed. Building upon this, we develop and implement the MALBO framework, a practical and sample-efficient methodology for automatically discovering the Pareto front of optimal team configurations. We then demonstrate

the framework’s validity through an empirical study that yields quantifiable insights into the architectural drivers of performance and cost.

1.7 Thesis Outline

The remainder of this thesis is structured as follows:

- **Chapter 2: Theoretical Background** provides a comprehensive overview of the foundational concepts underpinning this research, including the Transformer architecture, the evolution of Large Language Models, the paradigms of Multi-Agent Systems, and the principles of Bayesian Optimization.
- **Chapter 3: Related Works** reviews the existing literature on the application of Bayesian Optimization to LLM-related problems, contextualizing our work and highlighting the research gap that MALBO aims to fill.
- **Chapter 4: MALBO: Methodology and Mathematical Formulation** presents a detailed, formal description of our proposed framework, detailing the vector representation of LLMs, the problem formulation, and the components of the Bayesian optimization loop.
- **Chapter 5: Experimental Setup** describes the complete design of our empirical evaluation, including the software platform, the evaluation benchmark, the performance metrics, the pool of LLMs, and the configuration of the optimizer.
- **Chapter 6: Results and Analysis** presents and analyzes the empirical findings from our experiments, examining both the convergence of the optimization process and the practical insights derived from the optimal configurations discovered.
- **Chapter 7: Conclusion, Limitations, and Future Prospective** summarizes the key findings of this thesis, discusses the limitations of our current work, and proposes promising directions for future research.

Chapter 2

Theoretical background

2.1 The Transformer Architecture and Foundations of Large Language Models (LLMs)

The advent of the Transformer architecture, introduced by Vaswani et al. in their seminal 2017 paper "Attention Is All You Need," marked a paradigm shift in natural language processing (NLP) [1]. Before the Transformer, sequence modeling tasks were dominated by recurrent neural network (RNN) architectures, such as Long Short-Term Memory (LSTM) networks. While effective, RNNs process sequential data step-by-step, an inherently sequential computation that limits parallelization and poses challenges in capturing long-range dependencies due to the "information bottleneck" of a single hidden state vector. While the Transformer architecture, discussed next, abandoned recurrence entirely, it is noteworthy that recent research has begun to. These limitations have recently spurred new research into hybrid architectures, such as Mamba, which combine principles from both recurrent and parallelizable models to offer alternative scaling paradigms [6].

The Transformer architecture proposed a novel solution: to dispense with recurrence entirely and rely solely on an attention mechanism to draw global dependencies between input and output. This design not only yielded superior performance on tasks like machine translation but also enabled significantly more parallelization, allowing for training on much larger datasets than was previously feasible. This scalability is the cornerstone upon which modern Large Language Models (LLMs) are built.

In this section, we provide a detailed overview of the foundational components of the Transformer architecture. We begin with the initial processing of input data and then delve into the core mechanisms that define its operation: the self-attention mechanism, multi-head attention, position-wise feed-forward networks, and the use of residual connections with layer normalization.

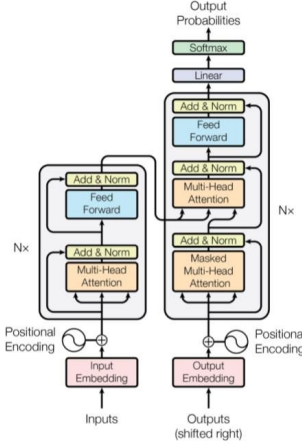


Figure 2.1: The Transformer architecture by [1], illustrating the transformer architecture, including embedding, positional encoding, self-attention mechanism, and Feed-Forward Networks.

2.1.1 Input Processing: Embeddings and Positional Encoding

Unlike RNNs that ingest tokens sequentially, the Transformer processes an entire sequence of tokens at once. To do so, the input text must first be converted into a numerical representation. This involves two primary steps.

Tokenization and Token Embedding First, the raw text is segmented into a sequence of tokens using a subword tokenization algorithm like Byte-Pair Encoding (BPE) [7]. Each token in the vocabulary is then mapped to a unique integer ID. These IDs are used to retrieve a corresponding dense vector representation from an embedding matrix, $W_e \in \mathbb{R}^{d_{\text{model}} \times N_v}$, where N_v is the vocabulary size and d_{model} is the dimensionality of the embedding vectors.

Positional Encoding Because the model contains no recurrence, the self-attention mechanism is inherently permutation-invariant; it has no sense of the order of tokens in the sequence. To provide the model with this crucial information, a positional encoding vector is added to each token embedding. The original Transformer paper employed sinusoidal functions of different frequencies for this purpose:

$$\text{PE}_{(\text{pos}, 2i)} = \sin(\text{pos}/10000^{2i/d_{\text{model}}}) \quad (2.1)$$

$$\text{PE}_{(\text{pos}, 2i+1)} = \cos(\text{pos}/10000^{2i/d_{\text{model}}}) \quad (2.2)$$

where pos is the position of the token in the sequence and i is the dimension index of the embedding. This method allows the model to learn to attend to

relative positions, as the positional encoding for any position can be represented as a linear function of any other. Other models, such as BERT, use learned positional embeddings instead [8]. The final input representation for each token is the sum of its token embedding and its positional encoding.

2.1.2 The Self-Attention Mechanism

The core innovation of the Transformer is the self-attention mechanism. It allows the model to weigh the importance of different tokens in the input sequence when producing a representation for each token. The mechanism is based on the concepts of Query (Q), Key (K), and Value (V). For each input token, we create three vectors: a Query vector, a Key vector, and a Value vector by multiplying its embedding by three distinct, learnable weight matrices (W_Q , W_K , W_V).

The attention score is computed as the dot product of the Query vector of the current token with the Key vectors of all other tokens in the sequence. This score determines how much attention the current token should pay to every other token. These scores are then scaled, passed through a softmax function to create a probability distribution, and used to compute a weighted sum of the Value vectors. The complete operation, known as Scaled Dot-Product Attention, is concisely expressed as:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \quad (2.3)$$

Here, Q, K, V are matrices containing the stacked query, key, and value vectors for all tokens in the sequence. The scaling factor $\sqrt{d_k}$, where d_k is the dimension of the key vectors, is crucial for stabilizing gradients during training. A visualization of this mechanism is shown in Figure 2.2. From a theoretical standpoint, recent work has begun to interpret the self-attention dynamics as a form of Wasserstein gradient flow, providing a deeper mathematical grounding for the architecture’s behavior [9].

2.1.3 Multi-Head Attention

Rather than performing a single attention function, the authors found it beneficial to linearly project the queries, keys, and values h times with different, learned linear projections. This allows the model to jointly attend to information from different representation subspaces at different positions. This mechanism is called Multi-Head Attention (MHA).

Each of these projected versions of queries, keys, and values is fed to an attention function in parallel, producing h output vectors. These are then concatenated and once again projected with a final weight matrix W_O to produce the final output of the MHA layer.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O \quad (2.4)$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \quad (2.5)$$

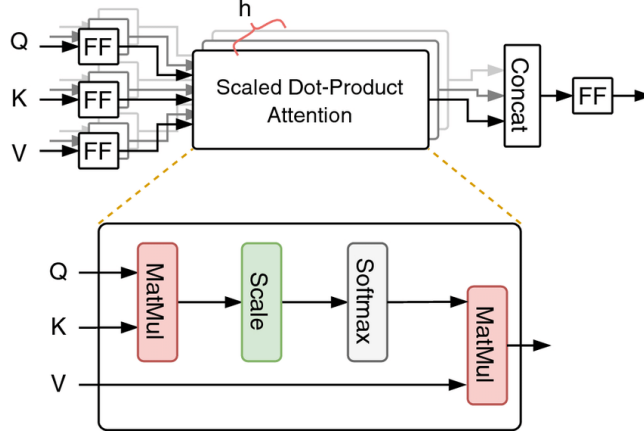


Figure 2.2: The Scaled Dot-Product Attention mechanism. The dot product of queries and keys is scaled and passed through a softmax function to obtain weights for the value vectors.

The projection matrices W_i^Q , W_i^K , and W_i^V are unique for each attention head i . MHA enhances the model's ability to focus on different positions and different types of relationships (e.g., syntactic vs. semantic).

Position-wise Feed-Forward Networks In addition to attention sub-layers, each layer of the encoder and decoder contains a fully connected feed-forward network (FFN), which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (2.6)$$

This component adds non-linearity to the model, increasing its expressive power.

The Transformer architecture is composed of a stack of these core components, organized into an encoder and a decoder.

Residual Connections and Layer Normalization Each sub-layer in the model (both the MHA and the feed-forward network) is followed by a residual connection and a layer normalization step. The output of each sub-layer is thus $\text{LayerNorm}(x + \text{Sublayer}(x))$, where $\text{Sublayer}(x)$ is the function implemented by the sub-layer itself. These residual connections are vital for training very deep networks by allowing gradients to flow more directly through the network.

2.1.4 Architectural Variants and Foundational Models

The core components of the Transformer (Multi-Head Attention, Position-wise Feed-Forward Networks, residual connections, and layer normalization) are stacked

to form deeper, more powerful models. The specific arrangement and utilization of the encoder and decoder blocks give rise to three primary architectural families, each suited for different classes of tasks [10].

Encoder-Decoder Architectures The full Transformer architecture, which incorporates both the encoder and decoder stacks, is primarily employed for sequence-to-sequence (seq2seq) tasks. In this configuration, the encoder processes the entire input sequence to generate a set of contextual representations. The decoder then attends to these representations (via cross-attention) while autoregressively generating the output sequence. This architecture is the standard for tasks like machine translation and text summarization. Foundational models built on this design include **T5 (Text-to-Text Transfer Transformer)** [11] and **BART (Bidirectional and Auto-Regressive Transformers)** [12].

Encoder-Only Architectures This variant utilizes only the encoder stack. The model processes an input sequence and outputs a rich, contextualized embedding for each token. These embeddings serve as powerful features for downstream tasks that require a deep understanding of the input text, such as text classification, named entity recognition, or sentiment analysis. The canonical example of this family is **BERT (Bidirectional Encoder Representations from Transformers)** [8], which is pre-trained using a Masked Language Modeling (MLM) objective. By predicting randomly masked tokens, BERT learns deep bidirectional context, a significant departure from the unidirectional context of traditional language models. Its successor, **RoBERTa**, further optimized the pre-training process to achieve improved performance [13].

Decoder-Only Architectures This architecture, illustrated in Figure 2.3, uses only the decoder stack, with the cross-attention mechanism removed. These models are inherently generative and are pre-trained on an autoregressive language modeling objective: predicting the next token given all previous tokens. The unification of diverse NLP tasks into this straightforward format of the next-token prediction, along with the scalability of this architecture, has led to its widespread architecture being used across various language tasks.

The seminal model in this family is the **GPT (Generative Pre-trained Transformer)** series [14], [15], [16]. The success of this paradigm has inspired a vast ecosystem of decoder-only models, including Meta’s **Llama** series [17], [18] and Mistral AI’s models [19]. These models were instrumental in pioneering the field of high-performance open-source LLMs, which has since expanded to include significant contributions from other major research labs such as Google, Microsoft, OpenAI, Alibaba, and DeepSeek AI. The ability of these models to perform new tasks in a zero-shot or few-shot manner, significantly enhanced by techniques like instruction tuning [20], has cemented the decoder-only architecture as the prevailing design for modern LLMs. The theoretical underpinnings of this success are an active area of research, with recent studies modeling

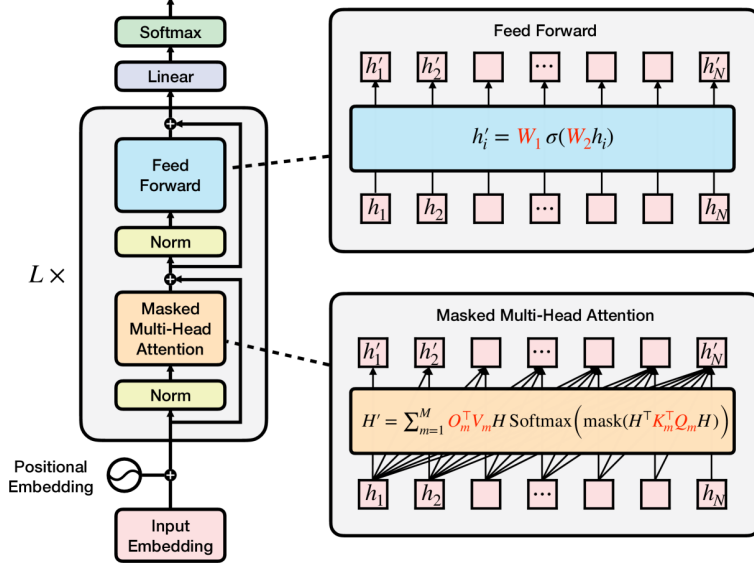


Figure 2.3: The architecture of a decoder-only Transformer, which forms the basis of most modern LLMs. The model consists of a stack of decoder layers, each containing a masked multi-head self-attention module and a feed-forward network. The cross-attention module is omitted. In the diagram, L represents the number of stacked decoder layers, W_1 and W_2 are the weight matrices of the position-wise feed-forward network, while Q_m , K_m , V_m , and O_m denote the query, key, value, and output projection matrices of the multi-head attention mechanism, respectively. Figure adapted from [10].

transformers as universal in-context learners that operate over probability distributions, further explaining their expressive power [21].

These pre-trained models, regardless of their architectural family, serve as the foundation of transfer learning in NLP. They can be adapted to specific downstream tasks via fine-tuning on relatively small, task-specific datasets, achieving state-of-the-art performance across a wide range of benchmarks.

2.1.5 From Autocomplete to Assistants

The fundamental Transformer architectures generate what are called "base models," which are potent engines for pattern completion and possess vast knowledge about the world. The transition to the practical LLM "assistants" that characterize the present landscape was driven by the development of methods to align the model's behavior with human intentions. This alignment is essential for integrating the model's intrinsic skills, acquired in pre-training, with its function as an effective and safe conversational agent.

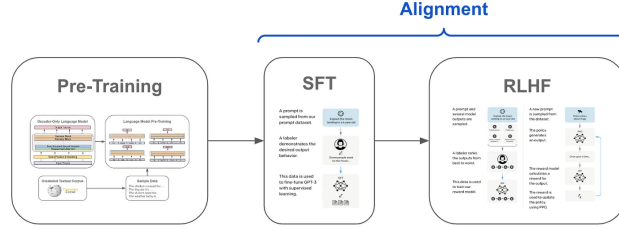


Figure 2.4: Overview of the training pipeline for instruction-following language models, adapted from Ouyang et al. (2022) [22]. The process consists of three main stages: (1) **Pre-Training**, where a decoder-only Transformer is trained on large unlabeled text corpora; (2) **Supervised Fine-Tuning (SFT)**, where human labelers provide examples of desired model behavior; and (3) **Reinforcement Learning from Human Feedback (RLHF)**, where human preferences are used to train a reward model and iteratively improve the policy via reinforcement learning.

Pre-training: Train a Base Model The first step in creating an LLM is pre-training. In this phase, a Transformer architecture is trained on a colossal corpus of text, often comprising trillions of tokens. The training objective is typically self-supervised. For decoder-only models, this objective is autoregressive language modeling: predicting the next token in a sequence. The result of this process is a **base model**, a powerful pattern-completion engine that possesses immense world knowledge but lacks an inherent understanding of user instructions.

From a statistical perspective, this self-supervised objective of predicting the next token is equivalent to minimizing the negative log-likelihood of the training corpus. Formally, the loss function for a sequence of tokens $\mathbf{x}$ is given by:

$$\mathcal{L}_{\text{LM}} = - \sum_{t=1}^T \log P(x_t | x_{<t}; \theta) \quad (2.7)$$

where θ represents the model’s parameters. This formulation grounds the pre-training process in the well-established principle of Maximum Likelihood Estimation (MLE), where the model parameters are optimized to maximize the probability of observing the training data [10].

Pre-training Data The capabilities of a base model are fundamentally shaped by the data on which it is trained. Pre-training corpora are vast, heterogeneous mixtures of data sourced from the public web (e.g., Common Crawl), books, academic articles (e.g., arXiv), and source code (e.g., GitHub) [23]. The composition of this data mixture is a decisive factor for model performance.. For instance, including a significant portion of high-quality code data has been shown to improve not only programming abilities but also general reasoning skills [24].

The process of constructing these datasets involves sophisticated data selection and mixture strategies. Data is typically filtered and weighted at multiple levels of granularity (token, sample, and source group) to optimize the final data mixture. While early methods relied on manual mixing ratios [25], more recent approaches employ learned mixture strategies, where proxy models or the LLM itself are used to determine the optimal composition of data sources to maximize performance under a fixed computational budget [26], [27]. This curation of data is as important to the final model’s capabilities as the architectural design itself.

Alignment: From Base Models to Instruction-Tuned Models Transforming a base model into a useful and safe assistant requires a process known as alignment. This process fine-tunes the model to follow instructions and adhere to desired human behaviors. Alignment is typically a multi-stage process, with preference tuning being the most critical phase.

- **Supervised Fine-Tuning (SFT):** The first stage involves fine-tuning the base model on a curated dataset of high-quality instruction-response pairs. SFT teaches the model the general format of following instructions and provides a strong foundation for helpfulness, yielding an initial policy denoted as π^{SFT} .
- **Preference Tuning:** After SFT, we further refine the model’s behavior using human preference data. The two dominant paradigms for this stage are Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO).

Reinforcement Learning from Human Feedback (RLHF) RLHF is a reward-based method that involves three phases [22]. First, a reward model (RM), $r_\phi(\mathbf{x}, y)$, is trained to predict human preferences. To collect data for the RM, the SFT model is prompted with an input $\mathbf{x}$ to generate a pair of responses (y_1, y_2) . A human labeler then indicates which response is preferred, denoted $y_w \succ y_l$. The preferences are assumed to be generated by a latent reward function r^* , and are often modeled using the Bradley-Terry model:

$$P^*(y_w \succ y_l | \mathbf{x}) = \frac{\exp(r^*(\mathbf{x}, y_w))}{\exp(r^*(\mathbf{x}, y_w)) + \exp(r^*(\mathbf{x}, y_l))} \quad (2.8)$$

The reward model r_ϕ is trained via maximum likelihood on a dataset of such preferences $\mathcal{D} = \{(\mathbf{x}^{(i)}, y_w^{(i)}, y_l^{(i)})\}_{i=1}^N$. This is equivalent to minimizing the binary cross-entropy loss:

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(\mathbf{x}, y_w, y_l) \sim \mathcal{D}} [\log \sigma(r_\phi(\mathbf{x}, y_w) - r_\phi(\mathbf{x}, y_l))] \quad (2.9)$$

where σ is the logistic function. In the final phase, the language model policy, π_θ , is optimized using reinforcement learning (e.g., Proximal Policy

Optimization, PPO) to maximize the expected reward from the learned RM, while a KL-divergence penalty term prevents the policy from deviating too far from the initial SFT policy π^{SFT} :

$$\max_{\pi_{\theta}} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}, y \sim \pi_{\theta}(y|\mathbf{x})} [r_{\phi}(\mathbf{x}, y)] - \beta D_{KL}[\pi_{\theta}(y|\mathbf{x}) || \pi^{\text{SFT}}(y|\mathbf{x})] \quad (2.10)$$

Direct Preference Optimization (DPO) DPO is a more recent, reward-free method that simplifies the alignment process by eliminating the need to explicitly train a separate reward model [28]. It leverages a specific parameterization of the reward model that allows the optimal policy to be extracted in closed form. This insight enables direct optimization of the language model policy on the preference dataset $\mathcal{D}$. The DPO loss function is formulated as:

$$\mathcal{L}_{\text{DPO}}(\pi_{\theta}; \pi_{\text{ref}}) = -\mathbb{E}_{(\mathbf{x}, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_{\theta}(y_w|\mathbf{x})}{\pi_{\text{ref}}(y_w|\mathbf{x})} - \beta \log \frac{\pi_{\theta}(y_l|\mathbf{x})}{\pi_{\text{ref}}(y_l|\mathbf{x})} \right) \right] \quad (2.11)$$

Here, π_{ref} is a reference policy, typically the SFT model π^{SFT} . By minimizing this loss, the model π_{θ} learns to increase the likelihood of preferred responses (y_w) and decrease the likelihood of dispreferred responses (y_l) relative to the reference policy, thus achieving alignment more directly and with greater training stability than RLHF.

Models that have undergone this alignment pipeline are known as **instruction-tuned** or **preference-tuned** models and form the basis of most practical applications, including the agent-based systems we analyze in this work.

Self-Alignment and Its Challenges As models approach or surpass human-level capabilities in specific domains, the reliance on human feedback for alignment becomes a significant bottleneck. This has motivated a new research direction in **self-alignment**, where an LLM is leveraged to generate its own training signals, reducing the dependence on human annotators.

One prominent approach is the use of **self-rewarding language models** [29]. In this paradigm, a model iteratively improves itself by generating responses to synthetic instructions and then scoring these responses using an "LLM-as-a-Judge" mechanism [30]. The highest and lowest-scoring responses are used as preference pairs to further fine-tune the model with DPO, creating an autonomous improvement loop. Extensions like meta-rewarding models refine this process by also fine-tuning the model's ability to judge, improving both its generation and evaluation skills concurrently [31].

However, this reliance on synthetic data and self-generated rewards introduces its own set of challenges. One critical issue is **reward misspecification**, where over-optimization against an imperfect, LLM-generated reward signal can amplify biases. These include verbosity bias (favoring longer answers) and self-enhancement bias (favoring its own style of response) [30], [32]. Another significant challenge is **distributional shift**, where training exclusively

on model-generated data can lead to a loss of diversity and a degradation of performance, a phenomenon known as "model collapse" [33]. Current research suggests that maintaining a mixture of human-authored and synthetic data is crucial for mitigating these risks and preserving model quality [34].

2.1.6 Architectural Innovations for Scalability and Efficiency

The immense scale of modern LLMs necessitated further evolution of the Transformer architecture itself. These innovations focus on improving training stability and, particularly for our work, enhancing inference efficiency to manage computational and memory costs.

The Dense and Efficient Lineage Many state-of-the-art models follow a dense architecture, where all parameters are used for every token. Meta’s LLaMA 3 models exemplify this trend by incorporating **Grouped-Query Attention (GQA)** [35]. GQA reduces the computational and memory burden of the Key-Value (KV) cache, a primary bottleneck during inference, thus enabling longer sequences and larger batch sizes.

The Sparse Path: Mixture of Experts (MoE) The Mixture of Experts (MoE) architecture decouples a model’s knowledge capacity from its inference cost [36]. An MoE layer replaces the dense FFN with a collection of parallel "expert" networks and a learned "router" that directs each token to a small subset of them. This creates two critical metrics: **total parameters** (reflecting knowledge capacity) and **active parameters** (reflecting inference cost).

This architecture is not only central to leading open-source models but is also widely understood to be the design behind proprietary state-of-the-art models like GPT-4. In the open space, Mixtral 8x7B demonstrated the efficacy of this approach, achieving the performance of a 70B dense model with only 13B active parameters [37]. The more recent **DeepSeek-V3** pushes this paradigm further, employing a 671B total parameter MoE architecture that activates 37B parameters per token. It introduces advanced routing strategies to balance expert utilization, achieving high performance while managing computational load [38].

2.1.7 The Economics of LLM Inference: Beyond Price-per-Token

The architectural choices described above have direct and significant consequences on the cost of using these models. The total cost of an LLM query is a function of its pricing model and its efficiency for a given task. The fundamental cost equation is:

$$\text{Total Cost} = (\text{Input Tokens} \times \text{Price}_{\text{in}}) + (\text{Output Tokens} \times \text{Price}_{\text{out}}) \quad (2.12)$$

However, a model’s price-per-token is an incomplete measure of its true operational cost. A more critical factor is its **token efficiency**: the number of input and output tokens a model requires to successfully complete a specific task. As demonstrated by industry benchmarks like the Artificial Analysis Intelligence Index, there is often a divergence between a model’s token pricing and its total cost to perform a complex evaluation suite.

This divergence is particularly pronounced in tasks that require sophisticated **reasoning**. Such tasks often necessitate longer, more structured prompts (e.g., using chain-of-thought or reasoning models), which inflates the input token count. The model, in turn, may need to generate verbose, step-by-step reasoning to arrive at a correct answer, increasing the output token count. Some providers even use tiered pricing, charging a premium for models or endpoints optimized for reasoning. An example is DeepSeek, whose models such as DeepSeek-V1 and DeepSeek-V3 share the same architecture but differ significantly in inference cost per token. Therefore, even with a lower per-token cost, a model that lacks token efficiency may incur higher overall expenses for complex tasks, as it may require longer reasoning traces compared to a more advanced, though higher-priced, alternative.

This interplay between architecture (which determines active parameters and capabilities), pricing, and token efficiency for a given task creates the complex optimization space that this thesis addresses. The goal is to find configurations that minimize total cost while satisfying performance constraints, a problem that cannot be solved by evaluating token prices alone.

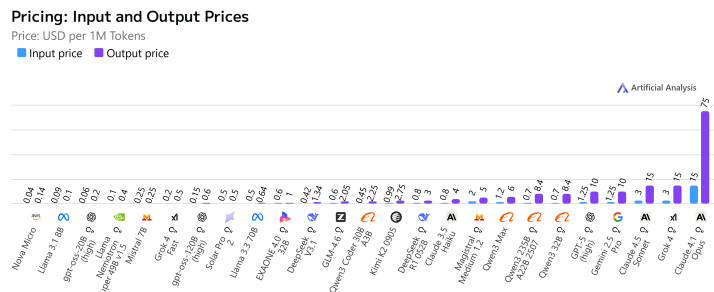


Figure 2.5: Input and output prices per token (in USD) across leading LLMs as reported by the Artificial Analysis Intelligence Index. Despite large differences in per-token pricing — for example, Grok-4 charges \$0.003/\$0.015 per input/output token, compared to Opus-4.1’s \$0.015/\$0.075 — these prices alone do not reflect true inference cost. Models such as Qwen-3 and Sonnet-4.5 show that lower token prices do not necessarily lead to cheaper execution when token usage efficiency is considered.

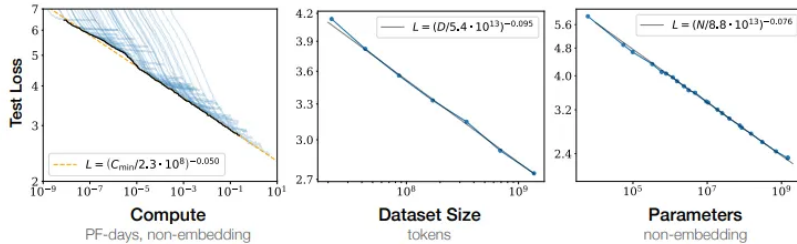


Figure 2.7: Compute-optimal scaling relationships for large language models, from Hoffmann et al. (2022) [40]. The plots illustrate (left) the training loss as a function of total compute (FLOPs) across models of different sizes, (center) the relation between compute and model parameters, and (right) the relation between compute and the number of training tokens. These scaling laws show that, for a fixed compute budget, optimal performance is achieved when model size and the number of training tokens are scaled proportionally.

2.1.9 A Statistical Perspective on Modern LLMs

While advances in deep learning architectures and computational scale have been the primary drivers of LLM success, a deeper engagement with the field of statistics is required to address emerging challenges related to their trustworthiness and reliability [10]. This work identifies two primary directions for this synergy: applying statistical principles to improve LLMs and, conversely, leveraging LLMs to augment statistical workflows.

Improving LLMs with Statistical Rigor While the probabilistic nature of LLM outputs provides a foundation for reasoning about their behavior, it does not inherently guarantee statistical validity. A critical challenge is that models can generate factually incorrect statements, or *hallucinations*, with high confidence as measured by token probability [43]. This discrepancy between probabilistic confidence and factual correctness necessitates the application of rigorous statistical methods to build trustworthy systems. We identify three key areas where such methods are essential:

- **Uncertainty Quantification (UQ):** The primary goal of UQ is to develop principled methods for quantifying the reliability of LLM outputs. While classical metrics like entropy can be computed over the next-token distribution [44], they often fail to capture semantic or factual uncertainty. More advanced approaches aim to address this by incorporating semantic features or internal model states [45]. A particularly robust framework for UQ is **Conformal Prediction (CP)** [46]. CP is a distribution-free method that can construct prediction sets with formal, finite-sample statistical guarantees on coverage. Its application to LLMs, while challenging due to large output spaces, has shown promise in controlling hallucination

risks and providing calibrated confidence estimates for tasks ranging from question answering to machine translation [47], [48].

- **Interpretability and Fairness:** LLMs often function as opaque, black-box models, making it difficult to understand their decision-making processes. Statistical tools for model interpretation, such as those that identify influential training data or internal model "circuits," can be adapted to probe their mechanisms. This is a prerequisite for diagnosing and mitigating the societal biases (e.g., related to gender or race) that are invariably inherited from large-scale, unstructured training data.
- **Principled Alignment:** The process of aligning LLM behavior with human values, exemplified by Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO), is fundamentally a problem of statistical inference from preference data. A rigorous statistical framework, often grounded in preference models like the Bradley-Terry model, is necessary to develop more robust, sample-efficient, and theoretically understood alignment techniques.

Augmenting Statistical Workflows with LLMs Conversely, LLMs can serve as powerful components within traditional statistical analysis pipelines. Their capabilities in natural language understanding enable new applications in:

- **Automated Data Processing:** LLMs can be employed for tasks such as automated data cleaning, feature engineering, and the extraction of structured variables from unstructured text sources like clinical records or financial reports.
- **Synthetic Data Generation:** When real data is scarce, private, or imbalanced, LLMs can be used to generate high-fidelity synthetic datasets. These datasets can then be used to train downstream statistical models while preserving privacy or correcting for class imbalances [10].

The integration of these two fields is essential for advancing both the theoretical foundations and the practical, trustworthy application of these transformative models.

The application of rigorous statistical methods extends beyond improving LLMs to drawing parallels with other complex learning systems, such as the human brain. Recent work has shown strong correlations between the internal representations of LLMs and human fMRI brain responses during language tasks, suggesting a degree of representational similarity that is specific to models trained on human language [49]. Furthermore, analytical frameworks rooted in Bayesian optimization and Pareto rationality have been successfully applied to model human decision-making under uncertainty. These studies analyze how human learners manage the exploration-exploitation trade-off, using tools like Wasserstein distance to represent their behavioral patterns as probability distributions [50], [51].

2.2 Multi-Agent Systems: Paradigms and Frameworks

The concept of distributing intelligence among multiple autonomous entities is a foundational paradigm in computer science, originating from the field of Distributed Artificial Intelligence (DAI) [52]. A Multi-Agent System (MAS) is a computerized system composed of multiple interacting, intelligent agents designed to solve problems that are difficult or impossible for a single agent or a monolithic system to solve [53]. This section provides a formal definition of MAS, explores key taxonomies for classifying them, and presents an overview of the open-source frameworks that enable their practical implementation, particularly in the context of LLMs.

2.2.1 Definitions and Taxonomies

An “agent” is generally defined as an autonomous entity that can perceive its environment, act upon it to achieve goals, and communicate with other agents [54]. A system becomes a MAS when it features multiple such agents interacting within a shared environment. Key characteristics of a MAS include:

- **Autonomy:** Each agent has control over its own actions and internal state [55].
- **Local Views:** No single agent possesses a global view of the system; its knowledge is limited to its own state and communications.
- **Decentralization:** There is typically no central controller; collective behavior emerges from local interactions [56].

It is critical to distinguish Multi-Agent Systems from Agent-Based Models (ABMs). While both involve simulating agents, their objectives differ. ABMs are primarily used as a scientific tool to *understand* the emergent collective behavior of agents, often in natural or social systems [57]. In contrast, MAS are an engineering paradigm used to *build* complex, distributed systems to solve specific practical problems [53].

The advent of Large Language Models has catalyzed a new era for MAS. LLMs serve as the reasoning “brain” for each agent, enabling them to perform complex planning, tool use, and, most importantly, coordinate through natural language [58]. This has shifted the primary challenge from designing rigid communication protocols to orchestrating sophisticated, language-based collaboration [59].

To navigate the design space of these systems, several taxonomies have been proposed. At a high level, MAS architectures can be classified by their control structure [60]:

- **Centralized:** A single coordinating agent or “orchestrator” assigns tasks and manages the information flow between other agents.

- **Decentralized:** Decision-making authority is dispersed, and agents coordinate through peer-to-peer interactions [61].

For LLM-powered systems specifically, a more granular taxonomy is required to capture the nuances of their architecture. Recent work proposes a multi-dimensional framework for analyzing these systems across four key axes: **task management**, **agent composition**, **collaboration**, and **context interaction** [62]. This framework helps to classify how a system decomposes problems, defines agent roles, manages communication, and maintains a shared understanding of the task.

Complementing these formal taxonomies are practical design philosophies from industry leaders. Anthropic, for instance, advocates for building agentic systems using simple, composable patterns, differentiating between fixed “workflows” and dynamic “agents” that direct their own processes [63]. Their multi-agent research system utilizes a hierarchical structure with a lead agent for planning and sub-agents for parallel information gathering, demonstrating a practical application of a centralized control paradigm [64].

2.2.2 Overview of Open-Source Frameworks

The theoretical paradigms of multi-agent systems are put into practice through software frameworks that abstract the complexities of agent creation, communication, and orchestration. The open-source community has produced a diverse ecosystem of such frameworks, each with a distinct architectural philosophy. We introduce two representative examples, OpenManus and SmolAgents, and then provide a comparative overview of other prominent frameworks.

OpenManus is an open-source framework designed to replicate and democratize the capabilities of advanced autonomous AI agents [65]. Its architecture is explicitly modular, comprising distinct layers for Agents, Tools, Prompts, and LLM Interaction [66]. A key feature of OpenManus is its support for two execution modes: a flexible *Direct Agent Execution* mode and a more structured *Flow Orchestration Execution* mode, which separates task planning from execution [67]. This dual-mode design allows it to handle both simple, reactive tasks and complex, pre-defined workflows.

SmolAgents, developed by Hugging Face (fig. 2.8), represents a minimalist and code-centric design philosophy [68]. The core idea is that agents “think in code”; instead of relying on complex tool-calling APIs, agents generate and execute Python code to perform actions [69]. This approach aims to improve accuracy and efficiency by leveraging the structured and expressive nature of a programming language. The framework is intentionally lightweight, LLM-agnostic, and designed for simplicity, making it a powerful tool for developers who prefer direct, code-based control over agent behavior [70].

Beyond these examples, the landscape of open-source frameworks is rich with different architectural approaches. Table 2.1 provides a comparative summary of several influential frameworks, highlighting their core paradigms and key features. This diversity underscores the active research and development

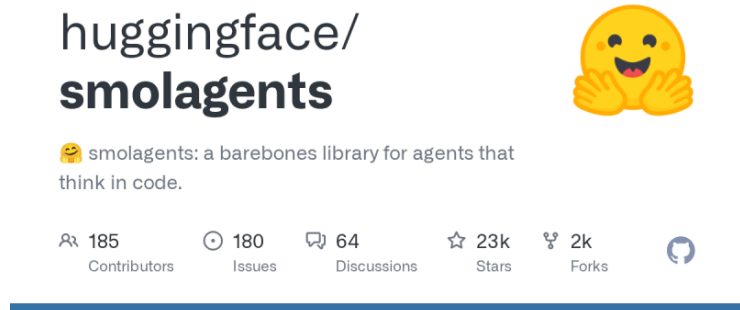


Figure 2.8: The open-source repository for the Smol-Agents framework. The multi-agent architecture employed in this thesis for validating MALBO is derived from the ‘open-deep-research’ code provided within this repository. The codebase was subsequently forked and adapted to integrate our optimization loop and configuration-based model swapping.

in the field, as different architectures are suited for different types of problems, from structured workflows to dynamic, conversational problem-solving [71]. The design of MALBO, as detailed in Chapter 3, is informed by this existing landscape, drawing on principles of modularity and orchestration to address our optimization challenge.

2.3 Bayesian Optimization (BO) for Black-Box Functions

Many foundational problems in science and engineering involve optimizing a function that is computationally or financially expensive to evaluate. Examples range from tuning hyperparameters of deep learning models to discovering novel materials or optimizing the design of complex systems [77], [78]. In these scenarios, the objective function $f(\mathbf{x})$ lacks a known analytical form and its derivatives are typically unavailable. We can only evaluate it at a point $\mathbf{x}$ to receive a (potentially noisy) observation y . This defines the problem of **black-box optimization**.

When function evaluations are expensive, standard optimization methods like grid search are infeasible due to the prohibitive number of evaluations required. Bayesian Optimization (BO) is a sequential, model-based strategy designed specifically for the global optimization of such expensive black-box functions [79].

The core strategy of BO is to build a probabilistic surrogate model of the objective function, which captures our beliefs about $f(\mathbf{x})$. This surrogate is cheap to evaluate and is updated with each new observation from the true func-

Table 2.1: Comparative Overview of Prominent Open-Source Multi-Agent Frameworks.

Framework	Paradigm	Features & Use Cases
AutoGen [72]	Conversation-Driven	Agents solve tasks via automated conversations. Highly flexible and extensible. Supports human-in-the-loop.
LangGraph [73]	Graph-Based, State Machine	Represents workflows as a graph. Enables cyclical control flows, explicit state management, and long-running agents.
CrewAI [74]	Role-Playing & Hierarchical	Agents are assigned specific roles and goals, collaborating like a human team. Uses a manager to orchestrate tasks.
MetaGPT [75]	SOPs & Software Company Simulation	Implements Standardized Operating Procedures (SOPs) to simulate a software development team for code generation.
ChatDev [76]	Waterfall Model Simulation	Simulates a virtual software company with distinct roles (CEO, programmer, tester) following a structured waterfall process.

tion. To decide where to sample next, BO uses an **acquisition function** that leverages the surrogate’s predictions and, crucially, its uncertainty estimates. This function guides the search by balancing **exploitation** (sampling in regions predicted to have high-performing outcomes) and **exploration** (sampling in regions where uncertainty is high). The point that maximizes the acquisition function is chosen for the next expensive evaluation of $f(\mathbf{x})$. This intelligent, sequential search process allows BO to find a global optimum with a significantly smaller number of function evaluations compared to other methods. The framework is composed of two primary components: the surrogate model and the acquisition function.

2.3.1 Gaussian Processes as Surrogate Models

The most common and effective surrogate model used in Bayesian Optimization is the **Gaussian Process (GP)** [80]. A GP is a non-parametric model that defines a distribution over functions. It is a generalization of the multivariate Gaussian distribution to an infinite-dimensional space of functions.

Formally, a Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution [80]. A GP is fully specified by a mean function $m(\mathbf{x})$ and a covariance function, or **kernel**, $k(\mathbf{x}, \mathbf{x}')$:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \quad (2.13)$$

The mean function $m(\mathbf{x})$ represents the expected value of the function at input $\mathbf{x}$, and is often assumed to be zero for simplicity. The kernel $k(\mathbf{x}, \mathbf{x}')$ models the covariance between the function values at two points, $\mathbf{x}$ and $\mathbf{x}'$. The choice of kernel is critical as it encodes our prior beliefs about the properties of the function, such as its smoothness or periodicity. A common choice is the Matérn family of kernels, which allows for controlling the smoothness of the modeled function.

Given a set of n observations $\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, the GP framework allows us to compute a posterior distribution over the function f . A key property of GPs is that the posterior predictive distribution for the function value f_* at a new test point $\mathbf{x}_*$ is also a Gaussian distribution:

$$P(f_* | \mathbf{x}_*, \mathcal{D}_n) = \mathcal{N}(\mu_n(\mathbf{x}_*), \sigma_n^2(\mathbf{x}_*)) \quad (2.14)$$

Here, $\mu_n(\mathbf{x}_*)$ is the posterior mean and $\sigma_n^2(\mathbf{x}_*)$ is the posterior variance. The posterior mean serves as the current best estimate of the function at $\mathbf{x}_*$, while the posterior variance provides a measure of uncertainty about that estimate. This principled quantification of uncertainty is what enables the intelligent exploration-exploitation trade-off managed by the acquisition function.

2.3.2 Acquisition Functions

The acquisition function, $\alpha(\mathbf{x})$, uses the posterior distribution provided by the GP to quantify the utility of evaluating the black-box function at a candidate point $\mathbf{x}$. The next point to be evaluated is selected by maximizing this function:

$$\mathbf{x}_{n+1} = \arg \max_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x} | \mathcal{D}_n) \quad (2.15)$$

While numerous acquisition functions exist for single-objective optimization, such as Expected Improvement (EI) and Upper Confidence Bound (UCB), the focus of this thesis is on the more complex multi-objective setting.

In Multi-Objective Optimization (MOO), the goal is not to find a single optimal point but to identify the set of optimal trade-offs known as the **Pareto front**. A solution is Pareto-optimal if no objective can be improved without degrading at least one other objective. A standard metric for evaluating the quality of a Pareto front approximation is the **Hypervolume (HV)** indicator, which measures the volume of the objective space that is dominated by the front and bounded by a reference point [81].

A principled acquisition function for MOO is, therefore, the **Expected Hypervolume Improvement (EHVI)**. EHVI measures the expected increase in the hypervolume of the current approximate Pareto front that would result from evaluating a new candidate point (or a batch of points) [82], [83]. By maximizing EHVI, the BO algorithm is directly guided to select points that are most likely to expand the dominated hypervolume, thus efficiently mapping the true Pareto front. While the computation of EHVI has historically been a significant bottleneck, particularly for parallel evaluations, recent advances in differentiable programming have made its optimization tractable and highly

effective, forming a core component of the methodology proposed in this thesis [83].

Chapter 3

Related works - LLM and Bayesian Optimization

3.1 Optimization in the Age of Large Language Models

The proliferation of pre-trained Large Language Models (LLMs) has marked a significant shift in natural language processing. These foundation models exhibit remarkable general-purpose capabilities, yet their performance on specific downstream tasks is highly sensitive to how they are prompted and configured. The initial approach for adapting LLMs to a specific task was model fine-tuning. While effective, this process presents considerable computational overhead and data privacy challenges, hindering its practical application in many scenarios [4].

This has led to the rise of *prompt engineering*, a more lightweight and flexible paradigm for model adaptation. A prompt is a sequence of symbols or tokens, selected from a vocabulary, which is prepended or concatenated to a user’s query to guide the model’s output. The challenge of discovering an optimal prompt sequence can be framed as a complex combinatorial optimization problem. The search space, defined by the vocabulary size $|V|$ raised to the power of the prompt length L (i.e., $|V|^L$), is often intractably large, necessitating efficient search strategies [3].

Prompt optimization methods are broadly categorized into two families:

1. **Soft Prompt Tuning (SPT):** These ”white-box” methods require access to the model’s internal states and gradients. They operate in the continuous embedding space of the model, directly optimizing prompt embeddings while keeping the core model parameters frozen. While parameter-efficient, SPT is incompatible with the growing trend of accessing LLMs via restricted, black-box APIs.
2. **Hard Prompt Tuning (HPT):** This ”black-box” approach directly

searches for an optimal sequence of discrete tokens. HPT is critically important in the *Model-as-a-Service (MaaS)* ecosystem, where users only have query-level access to powerful proprietary models. This black-box constraint aligns with both user needs for simplicity and provider needs for security and intellectual property protection [4].

The optimization challenge, however, extends beyond prompts. The modern LLM landscape is rich with complex decision-making problems, including the dynamic routing of queries to the most suitable model [84], the efficient merging of model checkpoints to combine skills, and the enforcement of complex logical constraints on text generation. Across these varied problems, a central theme emerges: the need for sample-efficient optimization algorithms that can navigate vast, high-dimensional, and often discrete search spaces under a black-box constraint. Among the various techniques available, Bayesian Optimization has emerged as a dominant and principled approach to address these challenges.

3.2 The Evolution of Bayesian Optimization for LLMs

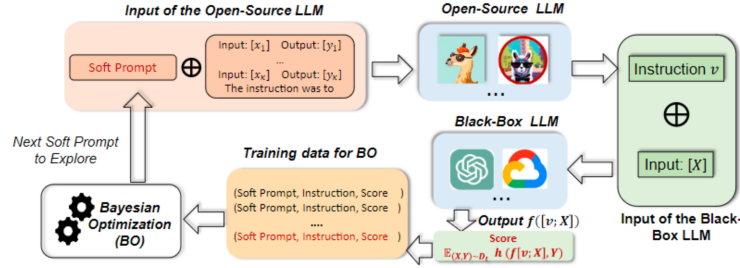


Figure 3.1: The iterative workflow of the InstructZero [85] and BOInG [4] frameworks, an example of ”white-box” instruction optimization. Bayesian Optimization is used to tune a continuous soft prompt for an open-source LLM, which acts as an instruction generator. The quality of the generated instruction is then evaluated by a separate, black-box LLM (the task solver). The resulting performance score provides the feedback signal to guide the optimization loop. Figure adapted from Chen et al. [85].

Bayesian Optimization (BO) has become the de-facto standard for sample-efficient black-box optimization, particularly when function evaluations are expensive—a condition that is quintessentially true for queries to large-scale LLMs. Its primary advantage lies in its strategy for managing the exploration-exploitation trade-off. By building a probabilistic surrogate model of the objective function (typically a Gaussian Process, GP), BO can quantify uncertainty and intelligently select the next point to evaluate, minimizing the number of costly queries required to find an optimal solution [86].

3.2.1 Foundational Applications: Prompt and Instruction Optimization

Initial applications of BO in the LLM space focused squarely on the problem of Hard Prompt Tuning. In our prior work [3], [4], [87], [88], we demonstrated the feasibility of using a "vanilla" BO algorithm for HPT. The core methodological innovation was a **continuous relaxation** of the discrete token search space. Instead of searching directly over tokens, the algorithm searches over a continuous space of token indices, optimizes a continuous acquisition function, and then rounds the resulting solution back to the nearest integer indices to retrieve the discrete tokens. This technique, illustrated in Figure 3.2, unlocked the power of standard GP-based BO for a fundamentally combinatorial problem.

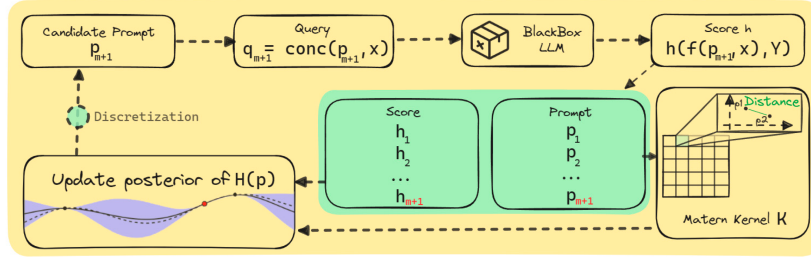


Figure 3.2: The general workflow of Hard Prompt Tuning (HPT) via Bayesian Optimization. A surrogate model (GP) is iteratively updated with new prompt evaluations to guide the search for an optimal discrete prompt. Figure adapted from [3].

Building on this foundation, more sophisticated methods have emerged. Our work on **BOInG (Bayesian Optimization for Instruction Generation)** [4] leverages two black-box LLMs: an *instruction generator* and a *task solver*. BO is used to find an optimal hard prompt for the generator, which in turn produces a natural language instruction for the solver. Its key innovation is a penalty term incorporated into the BO acquisition function, which pushes the search towards continuous representations that are close to the embeddings of known, valid tokens, thereby improving the coherence of the generated instructions. This black-box-centric approach (Figure 3.3) stands in contrast to methods like InstructZero [85], which also employ a two-LLM setup but require at least one model to be "white-box" to access its internal states.

3.2.2 Expanding the Scope: BO for Complex Workflows and Trustworthiness

As the field has matured, the focus of BO has expanded beyond optimizing single prompts to configuring entire LLM-powered systems and addressing the inherent challenges of reliability and cost. This progression reflects a move towards more complex, real-world deployment scenarios.

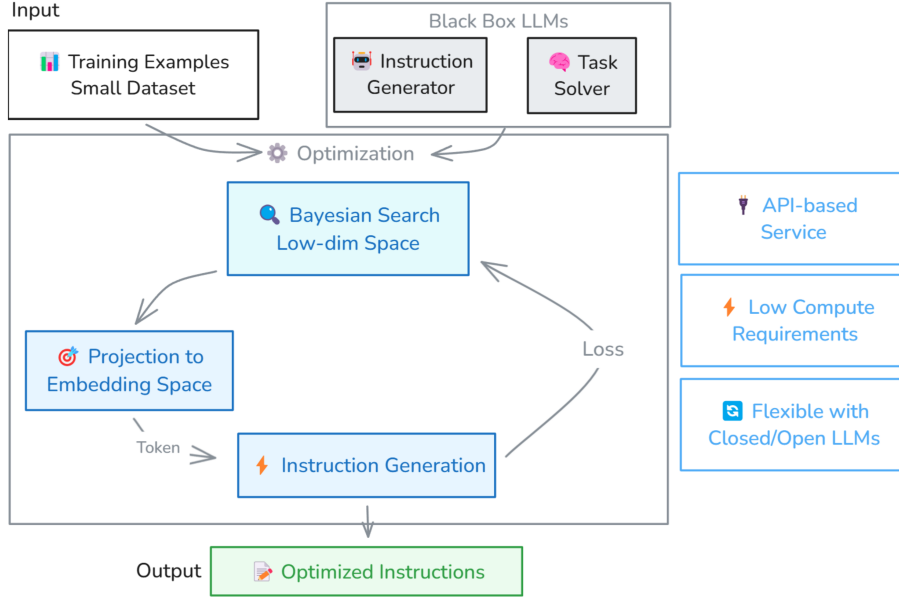


Figure 3.3: High-level workflow of the BOInG framework. Bayesian Optimization is used to find an optimal hard prompt for an Instruction Generator LLM, which in turn provides instructions to a Task Solver LLM. Both models can be treated as black-boxes. Figure adapted from [4].

A significant step in this direction involves optimizing multi-component pipelines, such as those used in Retrieval-Augmented Generation (RAG). Barker et al. [5] introduce a framework for the multi-objective optimization of a complete RAG system. The search space is no longer limited to prompt tokens but includes a wide array of hyperparameters, such as the choice of the LLM and embedding models, chunk size, and re-ranker thresholds, as illustrated in Figure 3.4. They frame this as a multi-objective problem to simultaneously optimize for cost, latency, and performance metrics like safety and alignment. We note that they employ a BO approach with the qLogNEHVI acquisition function, which is specifically designed to handle the noisy objective evaluations inherent in stochastic LLM outputs. This work demonstrates that BO can effectively navigate the high-dimensional, mixed-variable search space of a full pipeline to identify a Pareto-optimal set of configurations.

Parallel to optimizing system complexity, a second research thrust has focused on improving the trustworthiness of LLM-driven optimization. Relying solely on an LLM as an optimizer is risky due to its lack of calibrated uncertainty and its opaque internal reasoning, which undermines theoretical tractability **Chang2025LLINBO**. To address this, Chang et al. **Chang2025LLINBO** propose **LLINBO**, a hybrid framework that combines the strengths of LLMs with the principled uncertainty quantification of statistical surrogates like GPs.

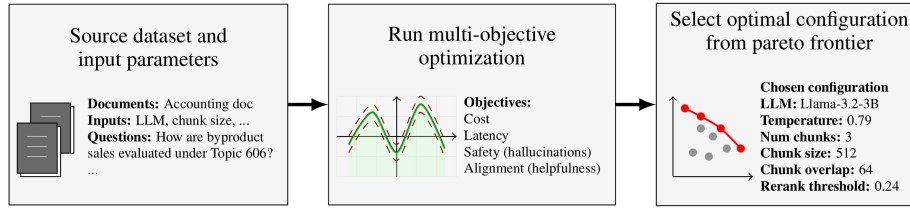


Figure 3.4: A high-level overview of multi-objective Bayesian Optimization for a RAG pipeline. The optimization process considers system-level parameters, including the choice of LLM, to find a Pareto front of optimal configurations balancing cost, latency, and performance. Figure adapted from [5].

Their core philosophy is to “leverage contextual reasoning strengths of LLMs for early exploration, while relying on principled statistical models to guide efficient exploitation”. They introduce three distinct mechanisms for this collaboration (Figure 3.5):

- **LLINBO-Transient:** A strategy that initially favors LLM suggestions and gradually transitions to GP-guided suggestions as more data becomes available.
- **LLINBO-Justify:** A rejection-sampling mechanism where the GP-based acquisition function acts as a verifier, discarding LLM suggestions that are deemed substantially suboptimal.
- **LLINBO-Constrained:** A method that treats the LLM’s suggestion as a soft constraint, refining the GP posterior to favor regions around the suggested point.

This hybrid approach demonstrates that integrating LLMs as collaborators within a principled BO loop, rather than as standalone optimizers, leads to more robust and reliable performance.

A third direction has focused on optimizing the dynamic selection of models from a pool of available experts, a task known as LLM routing. Shirkavand et al. [84] frame this as a similarity search problem and introduce **CSCR (Cost-Spectrum Contrastive Routing)**. Their method learns a shared embedding space where both prompts and models are represented. Models are described by compact, efficiently computed “fingerprints” (either from logits for open models or perplexity scores for black-box APIs). A contrastive learning objective, named Cost-Spectrum InfoNCE, is trained to pull prompt embeddings towards the cheapest expert that can solve the task correctly. At inference time, routing becomes a highly efficient k-Nearest Neighbors (k-NN) lookup. This work explicitly incorporates cost into the representation learning process, creating a metric space that inherently balances the accuracy-cost trade-off.

Pushing the paradigm even further, recent research has explored using LLMs for meta-optimization—that is, to automatically generate the optimization algorithm itself. Li et al. [90] introduce **LLaMEA-BO**, a framework that uses an

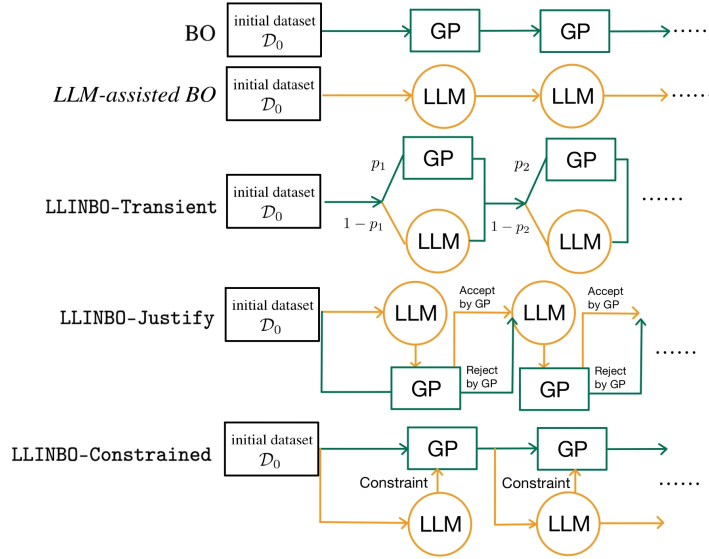


Figure 3.5: Diagram illustrating the three hybrid mechanisms proposed in the LLINBO framework, which combine suggestions from a traditional BO loop (with a GP) and an LLM-assisted loop. Figure adapted from [89].

evolution strategy to guide an LLM to write, combine, and mutate full Python implementations of BO algorithms. The LLM generates candidate algorithms, which are then evaluated on a benchmark test suite. The performance scores are used to select the best candidates, which are then "bred" (via crossover prompts) or "mutated" (via refinement prompts) to create the next generation of algorithms. This work demonstrates that LLM-generated algorithms can outperform state-of-the-art, human-designed BO baselines, positioning LLMs not just as tools within an optimization loop, but as co-designers of the optimization process itself. These advanced applications collectively show the field's rapid progression towards tackling more complex, system-level, and even meta-level optimization problems.

3.3 Limitations of the State of the Art and Motivation for MALBO

While the works we have reviewed demonstrate considerable progress in applying Bayesian Optimization to LLM-related tasks, they predominantly operate under a single-agent or monolithic pipeline paradigm. The existing literature largely overlooks the distinct and significantly more complex challenge of configuring and deploying an entire **team of collaborative LLM agents**.

The limitations of the current state of the art can be summarized in three key areas:

1. **The Multi-Agent Dimension:** Prior methods focus on optimizing a single artifact, be it a prompt, a set of hyperparameters, or a routing policy for a single query. They do not address the combinatorial complexity inherent in assigning specific models to distinct roles within a multi-agent system. The problem is no longer finding the "best prompt" or "best model," but finding the *optimal configuration of models* across a team of agents, where inter-agent dependencies and collaborative task structures are paramount. The search space explodes from optimizing a single vector to optimizing a matrix of assignments ($X \in \mathbb{R}^{N \times D}$, where N is the number of agents and D is the feature dimension of the models), a challenge existing methods are not designed to handle.
2. **The Multi-Objective Imperative:** While some recent work has begun to address multiple objectives like cost and latency [5], [84], this is not yet a standard approach in the field. Many frameworks still optimize for a single performance metric. A principled approach to multi-agent systems must have multi-objective optimization at its core, as the central challenge is precisely the trade-off between the collective performance of the team and its aggregate operational cost. This necessitates a framework capable of discovering a Pareto front of solutions.
3. **A Fundamentally Different Search Space:** The search space in prior work is typically tied to the token vocabulary (for HPT) or a flat list of system hyperparameters. In the multi-agent assignment problem we address, the decision variables are not tokens or simple scalar values but abstract, continuous representations of entire LLMs. The search space is a structured, high-dimensional space of *model capabilities*, where each point represents a potential team configuration. A new approach is required to effectively model this abstract configuration space.

In summary, the state of the art lacks a framework that can perform sample-efficient, black-box, multi-objective optimization for the multi-agent LLM assignment problem. This thesis introduces **Multi-Agent LLM Bayesian Optimization (MALBO)** to fill this gap. We design MALBO to tackle the combinatorial complexity of assigning different LLMs to various agent roles, while simultaneously optimizing for the critical trade-off between task performance and API costs. The following chapters will detail the mathematical formulation and methodology of the approach.

Chapter 4

MALBO: Methodology and Mathematical Formulation

Leveraging teams of LLM-based agents to solve complex problems is a promising frontier in artificial intelligence [59]. However, the effective composition of these teams presents a significant optimization challenge. The effectiveness of a multi-agent system hinges on the specific LLM assigned to each agent role, resulting in a vast and combinatorially complex design space. Furthermore, evaluating the performance of any given team configuration is an expensive black-box operation, requiring extensive simulation or real-world deployment.

This chapter introduces our proposed methodology, Multi-Agent LLM Bayesian Optimization (MALBO), designed to navigate this complex trade-off space. We formalize the problem of optimal LLM assignment as a multi-objective black-box optimization task. Our approach seeks to identify a set of agent configurations that represent the optimal frontier between task performance and the associated computational and financial costs. We detail each component of the MALBO framework, from the vector representation of LLMs to the multi-objective Bayesian optimization loop that drives the search for optimal solutions.

4.0.1 Problem Description: Optimal LLM Assignment in an Agent Team

We consider a multi-agent system composed of N agents, where each agent is assigned a distinct role (e.g., planner, tool-user, verifier). We have access to a pool of M unique Large Language Models, each with different capabilities, performance profiles, and inference costs. The core problem is to assign one LLM from this pool to each of the N agent roles to optimize the overall system’s performance on a given task.

This assignment problem is fundamentally combinatorial. With M available models and N agent roles, the total number of possible team configurations is M^N . Evaluating a single configuration requires executing the entire multi-

agent workflow on a benchmark task and measuring its performance and cost, which is an expensive, black-box process. The goal is not to find a single best team but rather to uncover the set of assignments that offer the best trade-offs between performance and cost. This framing naturally leads to a multi-objective optimization problem, where we aim to co-optimize two conflicting objectives: maximizing task accuracy and minimizing operational cost [91], [92].

4.0.2 Vector Representation of LLMs and the Configuration Space

To apply Bayesian optimization, we must first map the discrete choice of LLMs into a continuous space. We achieve this by representing each LLM as a point in a multi-dimensional feature space, a technique validated in hyperparameter optimization and universal black-box optimization [93], [94].

The Feature Space $\mathcal{F} \subset \mathbb{R}^D$

We define a feature space $\mathcal{F} \subset \mathbb{R}^D$, where each of the D dimensions corresponds to a key, quantifiable characteristic of a Large Language Model. These features are chosen to capture the aspects of a model that are most likely to influence its performance and cost within an agentic system. The features include:

- **Performance Metrics:** Standardized scores on academic benchmarks (e.g., MMLU-Pro, LiveCodeBench, GPQA Diamond) that proxy for reasoning, coding, and language understanding capabilities.
- **Architectural Properties:** Parameters such as context window length, total and active parameter counts (for MoE models), and model type (e.g., dense, sparse).
- **Economic Factors:** API inference costs, measured in price per input and output token.

Each LLM is thus represented by a feature vector $\mathbf{f} \in \mathcal{F}$ that quantifies its profile.

The Matrix of Available Models $\mathbf{L} \in \mathbb{R}^{M \times D}$

The set of M concrete, deployable LLMs available for assignment is represented as a matrix $\mathbf{L} \in \mathbb{R}^{M \times D}$. Each row $\mathbf{l}_j$ of $\mathbf{L}$ is the D -dimensional feature vector for the j -th LLM in our pool. This matrix constitutes the ground truth of our discrete search space.

4.0.3 Multi-Objective Optimization Problem Formulation

With the continuous feature space defined, we can now formally state the optimization problem. A complete team configuration is a selection of one LLM for each of the N agent roles. In our continuous relaxation, a configuration is a

point $\mathbf{x} \in \mathcal{F}^N$, which is an $N \times D$ matrix where each row represents the feature vector of the LLM assigned to the corresponding agent role. The search space for our optimization is therefore $\mathcal{X} = \mathcal{F}^N \subset \mathbb{R}^{N \times D}$.

The Black-Box Objective Function $f : \mathcal{X} \rightarrow \mathbb{R}^2$

We define a vector-valued black-box objective function f that maps a team configuration $\mathbf{x}$ to a two-dimensional output vector representing its performance and cost:

$$f(\mathbf{x}) = [y_{\text{accuracy}}(\mathbf{x}), y_{\text{cost}}(\mathbf{x})] \quad (4.1)$$

The components of this function are:

- $y_{\text{accuracy}}(\mathbf{x})$: A performance metric (e.g., task success rate on the GAIA benchmark) which we aim to **maximize**.
- $y_{\text{cost}}(\mathbf{x})$: The total operational cost (e.g., cumulative API costs in dollars) incurred to complete the task, which we aim to **minimize**.

Since we seek to maximize accuracy and minimize cost, we formulate the optimization problem as maximizing the vector $[y_{\text{accuracy}}(\mathbf{x}), -y_{\text{cost}}(\mathbf{x})]$.

Defining the Pareto Front

Because the objectives of accuracy and cost are conflicting, there is typically no single configuration that is optimal for both. Instead, we seek the set of Pareto-optimal solutions [92]. An objective vector $\mathbf{v}_1$ *Pareto-dominates* another vector $\mathbf{v}_2$, denoted $\mathbf{v}_1 \succ \mathbf{v}_2$, if it is better or equal on all objectives and strictly better on at least one. A configuration $\mathbf{x}$ is Pareto-optimal if no other configuration $\mathbf{x}'$ exists such that $f(\mathbf{x}') \succ f(\mathbf{x})$.

The set of all such non-dominated objective vectors constitutes the **Pareto front**, denoted $\mathcal{P}^*$. The goal of MALBO is to find a diverse set of configurations whose objective values provide a high-quality approximation of this true Pareto front.

4.0.4 The Bayesian Optimization Loop for MALBO

MALBO uses a multi-objective Bayesian optimization (MOBO) loop to efficiently search the configuration space $\mathcal{X}$. The loop iteratively builds surrogate models of the objective functions and uses an acquisition function to select the next batch of configurations to evaluate. As illustrated in Fig. 4.1, each iteration involves modeling, acquisition optimization, projection into the real configuration space, and black-box evaluation of LLM-based agent teams.

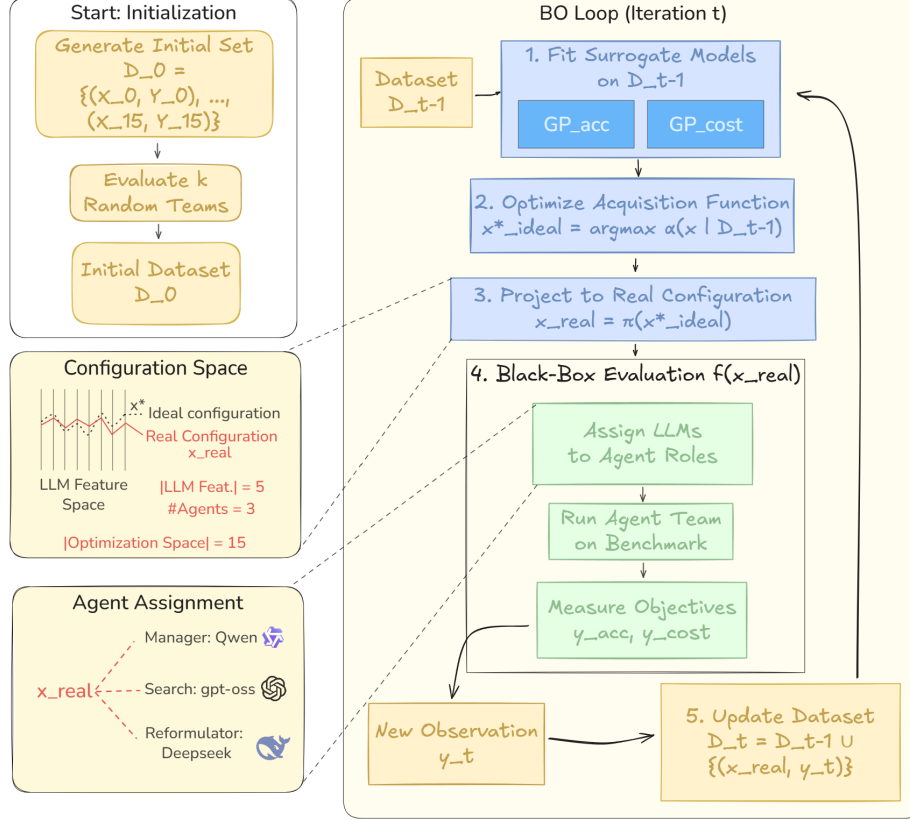


Figure 4.1: Overview of the MALBO (Multi-Agent LLM Bayesian Optimization) loop. The process starts with the initialization of an initial dataset D_0 by evaluating random agent team configurations. Each iteration of the Bayesian Optimization loop fits surrogate Gaussian Process models to predict performance and cost, optimizes an acquisition function to select the next ideal configuration, projects it into the real configuration space, and evaluates it through black-box benchmarking of LLM-based agent teams. The resulting observations are then used to update the dataset D_t .

Surrogate Models for Accuracy and Cost (GP_1, GP_2)

We model the two black-box objective functions, accuracy and cost, using two independent Gaussian Processes (GPs) [80]. Given a dataset of t observations $\mathcal{D}_t = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^t$, the posterior predictive distribution for a new configuration $\mathbf{x}_*$ is a Gaussian, providing both a mean prediction (the expected outcome) and a variance (a measure of uncertainty). We use one GP to model $y_{accuracy}$ and a second to model $-y_{cost}$. This approach is standard in MOBO and is well-supported by frameworks like BoTorch through its ‘ModelListGP’ interface [95].

Acquisition Function: q-Log Expected Hypervolume Improvement (qLogEHVI)

To balance the exploration-exploitation trade-off across multiple objectives, we employ an acquisition function based on the **hypervolume (HV)** indicator [81]. The hypervolume of a set of points on the Pareto front measures the size of the objective space they dominate. An ideal acquisition function for MOBO seeks to select new candidate points that are expected to maximize the improvement in this hypervolume.

We use the **q-Expected Hypervolume Improvement (qEHVI)** [83], an acquisition function designed for parallel (or batch) multi-objective optimization. It computes the expected hypervolume improvement from evaluating a batch of q candidate points. We use its logarithm, qLogEHVI, for improved numerical stability during optimization. This function guides the search toward regions that are most likely to expand the known Pareto front, either by improving upon existing solutions (exploitation) or by reducing uncertainty in unexplored regions (exploration).

The Optimization Loop Algorithm

The complete MALBO process is an iterative loop, detailed in Algorithm 1. At each iteration t , the algorithm performs the following steps:

1. **Fit Surrogate Models:** The two GPs, GP_{acc} and GP_{cost} , are updated using all available data $\mathcal{D}_{t-1}$.
2. **Optimize Acquisition Function:** The qLogEHVI acquisition function is maximized using the posterior distributions from the GPs to find the next batch of q candidate configurations $\mathbf{X}_t = \{\mathbf{x}_1, \dots, \mathbf{x}_q\}$.
3. **Evaluate Black-Box Function:** Each candidate configuration $\mathbf{x}_i$ in the batch is evaluated on the true objective function f to obtain its accuracy and cost, $\mathbf{y}_i = f(\mathbf{x}_i)$. This is the most expensive step of the loop.
4. **Update Dataset:** The new observations $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^q$ are added to the dataset, creating $\mathcal{D}_t$, and the process repeats.

4.0.5 From Continuous Space to Discrete Assignment: The Projection Function π

The Bayesian optimization loop operates in the continuous feature space $\mathcal{F}^N$ and proposes an "ideal" configuration $\mathbf{x}_{\text{ideal}} \in \mathcal{F}^N$. However, to evaluate this configuration, we must deploy a team of actual LLMs from our available set, represented by the matrix $\mathbf{L}$. This requires a mapping from the continuous space back to the discrete set of available models.

We define a projection function $\pi : \mathcal{F} \rightarrow L$ that maps an ideal LLM feature vector to the feature vector of the most similar real LLM. This projection is

Algorithm 1 MALBO: Multi-Agent LLM Bayesian Optimization Loop

- 1: **Input:** Black-box function f , initial dataset $\mathcal{D}_0$, batch size q , total budget T .
 - 2: Initialize surrogate models GP_{acc} , GP_{cost} on $\mathcal{D}_0$.
 - 3: **for** $t = 1, \dots, T/q$ **do**
 - 4: Fit GP_{acc} and GP_{cost} to current dataset $\mathcal{D}_{t-1}$.
 - 5: Construct the qLogEHVI acquisition function $\alpha(\cdot)$ from the GPs.
 - 6: Find the next batch of candidates: $\mathbf{X}_t \leftarrow \arg \max_{\mathbf{X}' \in \mathcal{X}^q} \alpha(\mathbf{X}')$.
 - 7: Evaluate the true black-box function for each candidate: $\mathbf{Y}_t \leftarrow f(\mathbf{X}_t)$.
 - 8: Augment the dataset: $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{(\mathbf{X}_t, \mathbf{Y}_t)\}$.
 - 9: **end for**
 - 10: **return** The set of non-dominated solutions found in $\mathcal{D}_T$.
-

achieved by finding the nearest neighbor in $\mathbf{L}$ using Euclidean distance:

$$\pi(\mathbf{f}_{\text{ideal}}) = \arg \min_{\mathbf{l}_j \in \mathbf{L}} \|\mathbf{f}_{\text{ideal}} - \mathbf{l}_j\|_2 \quad (4.2)$$

This projection is applied to each of the N rows of the candidate configuration matrix $\mathbf{X}_t$ proposed by the acquisition function optimizer. This step, adapted from surrogate optimization methods for discrete problems [96], ensures that every configuration evaluated by the black-box function f corresponds to a real, deployable team of LLMs.

Chapter 5

Experimental Setup

To empirically validate the MALBO framework, we designed and executed a series of experiments aimed at finding the optimal assignment of Large Language Models (LLMs) to a multi-agent system. This chapter details the complete experimental setup, including the software platform, the evaluation benchmark, the performance and cost metrics, the configuration of the Bayesian optimizer, and the pool of LLMs used in the study.

5.1 Development and Integration Platform

Our experimental platform is built upon **SmolAgents**, an open-source, minimalist multi-agent framework developed by Hugging Face [68]. We selected this framework for several key reasons. Firstly, its provider-agnostic design allows for the seamless integration of models from various sources, which is essential for exploring a diverse model space. Secondly, its "think in code" philosophy, where agents generate and execute Python code snippets as actions, has been demonstrated to be more efficient and performant than traditional tool-calling APIs, particularly with smaller or more specialized models [97].

Motivation for Code-as-Action Expressing agent actions directly as executable Python code (rather than declarative JSON tool calls) reduces mediation overhead and exploits the strong code priors of modern LLMs. Recent empirical evidence shows this paradigm can cut step counts (and thus total LLM calls) by roughly 30% while improving difficult task success rates [97]. Additionally, frameworks enabling dynamic, on-the-fly action synthesis instead of restricting to a static tool set (e.g., unconstrained code-level generation) further enhance long-horizon adaptability [98]. This informed our decision to adopt a lightweight wrapper that executes generated code inside a controlled sandbox, enabling (i) immediate inspection of intermediate variables, (ii) iterative self-repair, and (iii) seamless composition of library calls without pre-registering each operation as a separate tool.

To facilitate the optimization loop, we extended the base framework with a dedicated optimization interface. As detailed in our contributions to the open-source repository [99], we introduced a programmatic API ("optimization_interface.py") that allows an external optimizer, such as MALBO, to systematically evaluate agent team configurations. This interface accepts a dictionary defining the LLM assignment for each agent role and returns a tuple containing the final performance and cost metrics.

The multi-agent system under evaluation is configured with a team of five distinct agent roles: a `manager`, a `search_agent`, a `text_inspector`, a `visual_qa` agent, and a `reformulator`. However, to reduce the dimensionality of the search space from $D \times 5$ to a more manageable $D \times 3$ and thereby accelerate convergence, we focused our optimization efforts on the three most influential roles: the `manager`, the `search_agent`, and the `reformulator`. The remaining two roles, `text_inspector` and `visual_qa`, were assigned a fixed, high-performance yet cost-effective default model (`openrouter/openai/gpt-4.1-mini`) for all experimental runs. This simplification makes it feasible to explore the agent assignment space with greater precision and reduced computational overhead.

5.2 Evaluation Benchmark: GAIA

We selected **GAIA (General AI Assistants)** as the evaluation benchmark for our experiments [100]. Developed by Meta and Hugging Face, GAIA assesses multi-step, tool-mediated tasks (web navigation, retrieval, synthesis) that better reflect coordinated agent workflows than single-hop QA. Its tasks require planning, external information integration, and verification, aligning with the collaborative division of labor in our five-role system.

Alternative Benchmarks Considered

Prior to committing to GAIA we reviewed complementary agent-oriented benchmarks:

- **InfiAgent-DABench**: Tabular data analysis via ReAct + Python sandbox; strong automatic correctness through format prompting, but modality narrow (CSV-centric) and limited cross-source synthesis.
- **DABStep**: Multi-step financial / mixed-format analytics with factoid scoring; highly challenging, but very low absolute accuracies (<20% on harder sets) reduce early discriminative signal between configurations.
- **GTA**: Real user queries with tool argument, selection, and summarization metrics; granular tool-use diagnostics but less emphasis on broad multi-hop synthesis.
- **Mobile-Bench**: Multi-app mobile UI interaction with milestone (Checkpoint) tracking; environment coupling not aligned with our cloud API context.

Rationale for GAIA

GAIA offered (i) mid-range difficulty producing measurable variance without pervasive failure, (ii) heterogeneous action requirements (retrieval + synthesis) mapping to distinct agent roles, (iii) sensitivity to reasoning verbosity (enabling joint performance–cost analysis), and (iv) broad external validity for knowledge-work style orchestration. Hard-tier exclusion avoided disproportionately long trajectories with diminishing marginal ranking power in early BO iterations.

Subset Construction Each black-box evaluation uses a 10-task slice (5 *easy*, 5 *medium*). Pilot trials showed this yields stable wall-clock bounds and non-saturated accuracy gradients while keeping per-evaluation cost within budget. Hard tasks were deferred to future extensions once a Pareto set over moderate difficulty stabilizes. Sample questions can be found in the Appendix B.

Evaluation Procedure For every task we record: (1) binary success, (2) cumulative input and output token counts per role (later monetized), and (3) elapsed wall-clock (diagnostic only). Success rate aggregates to the performance objective; tokenized pricing aggregates to the cost objective. No heuristic penalties are added; subsequent scaling applies min–max normalization across observed configurations for surrogate stability.

Relation to Feature Space GAIA’s composite demands (knowledge breadth, procedural coding for tool adaptation, arithmetic / structured reasoning, and efficiency) correspond directly to the five-dimensional feature embedding (MMLU-Pro, LiveCodeBench, GPQA Diamond, input token cost, output token cost) described in Section 5.5. Using only universally reported metrics avoided sparsity that would arise from adding less consistently available benchmarks (e.g., MATH, MBPP) under a tight evaluation budget.

Limitations GAIA does not isolate fine-grained tool argument accuracy (better covered by GTA), nor deep statistical tabular analytics breadth (InfiAgent-DABench), nor UI-manipulation robustness (Mobile-Bench). Future work may layer a secondary panel once initial Pareto exploration converges.

5.3 Performance Metrics

The MALBO framework is designed for multi-objective optimization, concurrently targeting two conflicting objectives: task performance and operational cost.

Objective 1: Performance (Accuracy) We define performance as the **task success rate** on the selected subset of the GAIA benchmark. This is a scalar

value normalized to the range $[0, 1]$, where 1 indicates that the agent team successfully answered all 10 questions correctly. The primary goal for this objective is **maximization**.

Objective 2: Cost We define cost as the **total aggregated API execution cost** in US dollars (USD) required for the agent team to complete the 10-task evaluation set. This cost is calculated by summing the token-based fees for all input and output tokens processed by each LLM in the team, based on the pricing models of their respective providers at the time of the experiment. The primary goal for this objective is **minimization**. To align with the maximization framework of BoTorch, we optimize for the negative cost, where the optimizer seeks to move the value towards zero.

5.4 Optimizer Configuration

We implemented the Bayesian optimization loop using **BoTorch** [95], a state-of-the-art library for Bayesian optimization in Python. This framework provides a suite of tools for multi-objective Bayesian optimization, including flexible implementations of Gaussian Process (GP) surrogate models and a wide range of acquisition functions.

5.4.1 Initialization Strategy

The optimization process began with an initial design of experiments to seed the surrogate models. We generated the **15 initial configurations** by randomly sampling, with replacement, from the discrete pool of available LLMs for each of the three optimized agent roles (**manager**, **search_agent**, and **reformulator**). This direct sampling strategy ensures that each point in the initial dataset corresponds exactly to a real, deployable team configuration and its associated feature vector. Therefore, these initial points do not require the nearest-neighbor projection function π that is used in the main optimization loop.

5.4.2 Optimization Loop

Following the initialization phase, we ran the Bayesian optimization loop for **15 sequential iterations**, with a batch size of $q = 1$. This resulted in a total computational budget of 30 full benchmark evaluations (15 initial + 15 guided by the optimizer).

The acquisition function chosen for this multi-objective problem was the **q-Log Expected Hypervolume Improvement (qLogEHVI)** [83]. This function was selected as it is the standard for multi-objective BO, guiding the search towards candidate points that are most likely to expand the volume of the known Pareto front, thereby efficiently balancing exploration and exploitation.

5.5 Selection of Large Language Models

The pool of candidate models for our experiment was curated to represent a diverse and cutting-edge cross-section of the modern LLM landscape. We selected a range of models accessible via the Amazon Bedrock platform and other compatible APIs, ensuring variety in architecture (dense vs. Mixture-of-Experts), parameter count, and cost profile. The feature space used to represent each LLM consists of five dimensions chosen to capture key aspects of reasoning, coding, and cost:

- **MMLU-Pro:** A proxy for general knowledge and multi-domain reasoning ability.
- **LiveCodeBench:** A proxy for code generation and algorithmic logic.
- **GPQA Diamond:** A proxy for hard mathematical and scientific reasoning.
- **Cost:** The API inference costs, separated into price per one million input tokens and price per one million output tokens.

These benchmarks were selected because they are widely reported and consistently evaluated, providing a reliable basis for comparison. The performance data was sourced primarily from **Artificial Analysis**, a third-party platform that independently re-runs benchmarks to ensure a fair comparison between models (see Figure 5.1 and 5.2).

Feature Selection

The five selected dimensions (MMLU-Pro, LiveCodeBench, GPQA Diamond, input token cost, output token cost) form a compact, uniformly available embedding across all candidate models. MMLU proxies broad factual and multi-domain reasoning; LiveCodeBench captures structured code synthesis leveraged by code-as-action planning; GPQA reflects arithmetic / logical scientific reasoning common in multi-hop verification; separate input and output pricing dimensions model asymmetric billing schemes affecting marginal trajectory cost. Additional potential descriptors (context length, active parameters for MoE routing, multilingual scores) were tracked qualitatively for narrative analysis but omitted from the optimization embedding to preserve sample efficiency under a 30-evaluation cap.

Meta Llama 3.1 8B Instruct (Meta AI) A compact dense decoder-only Transformer using Grouped-Query Attention (GQA) to reduce KV cache pressure and enable efficient 128K context inference. Trained on a large (15T tokens) multilingual corpus with supervised fine-tuning (SFT) and RLHF for instruction alignment. Provides strong multilingual reasoning relative to its parameter scale [101], [102].

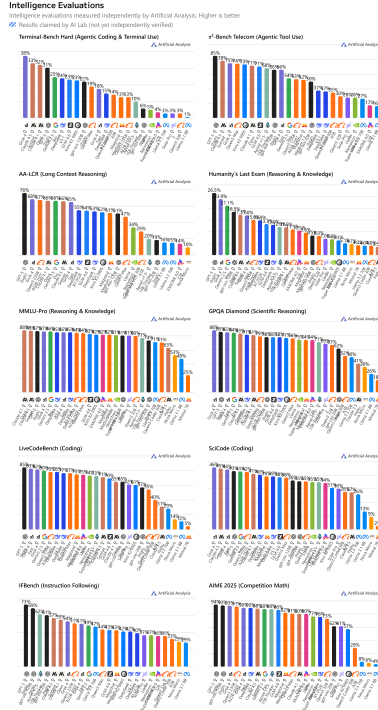


Figure 5.1: Screenshot of the LLM performance benchmarks from Artificial Analysis, showing the standardized evaluation of models like MMLU and Live-CodeBench.

Meta Llama 3.3 70B Instruct (Meta AI) An optimized 70B successor variant delivering quality uplift over earlier 70B releases without parameter growth. Retains 128K context, refined post-training alignment, and efficiency improvements in inference scheduling and memory layout; positioned as a high-capability open-weight alternative for general reasoning and tool orchestration [102], [103].

Mistral 7B Instruct v0.2 (Mistral AI) A highly efficient 7B dense model combining Sliding Window Attention (SWA) for long-sequence scalability with GQA for faster multi-head execution. Demonstrates that architectural efficiency can outperform larger legacy dense models (e.g., surpassing LLaMA 2 13B in several benchmarks) while remaining cost-effective for multi-agent role assignment [19].

OpenAI GPT-OSS 20B / 120B (OpenAI) Open-weight Mixture-of-Experts (MoE) Transformer family. Each token activates only a small subset of experts (3.6B and 5.1B active parameters respectively), decoupling total capacity from

inference cost. Supports configurable reasoning modes and lightweight quantization (e.g., MXFP4) for deployment efficiency. Serves as a bridge between fully proprietary frontier models and reproducible research baselines [104].

Qwen3 32B (Alibaba / Qwen Team) A dense Transformer emphasizing balanced performance across coding, math, and general reasoning with extended context handling. Improvements in tokenizer design and training efficiency yield performance competitive with larger earlier-generation models, making it a strong mid-to-upper tier assignment candidate [105].

Qwen3 Coder 30B A3B (Alibaba / Qwen Team) A specialized MoE-oriented coding variant (releasing both dense and routed expert configurations) emphasizing function calling, tool use, and extended (up to 262K) native context. Its specialization enhances downstream code synthesis and iterative reformulation roles in agent workflows [105].

Claude 3.5 Haiku (Anthropic) A lightweight, latency-optimized model delivering instruction following and structured reasoning quality approaching larger tier siblings while maintaining aggressive throughput. Adds improved multimodal grounding (e.g., UI / screenshot interpretation) and fast tool mediation, making it suitable for fast-turn manager or verifier roles under tight cost constraints [2].

DeepSeek-V3 (DeepSeek) A large-scale MoE architecture (671B total, 37B active parameters per token) introducing Multi-head Latent Attention (MLA) and auxiliary-loss-free expert balancing to improve routing stability and efficiency. Trained with an emphasis on cost-effective scaling (reduced GPU-hour footprint) while retaining frontier-level reasoning performance [106].

Amazon Nova Micro (Amazon) A compact text-only model within Amazon’s Nova family emphasizing ultra-low latency, broad multilingual coverage, and minimal per-token pricing for high-frequency orchestration pathways. Serves as a cost-efficient baseline for roles where marginal reasoning uplift does not justify higher active parameter footprints [107].

As an additional contextual note, several recently released or higher-tier proprietary frontier families, such as OpenAI’s latest GPT-5 series (including smaller o3 / o4-mini and GPT-5 mini / nano variants), Google’s Gemini 2.5 family, and Meta’s new Llama 4 line, were not integrated due to unavailability through our provisioned cloud endpoints at experiment time. Their exclusion reflects platform access constraints rather than methodological limitations; future replications may incorporate them to extend the performance–cost trade-off space [102], [104].

The final normalized feature matrix derived from these sources is summarized in Table 5.1. All numeric feature dimensions were min–max scaled to $[0, 1]$ prior to use in the MALBO optimization process.

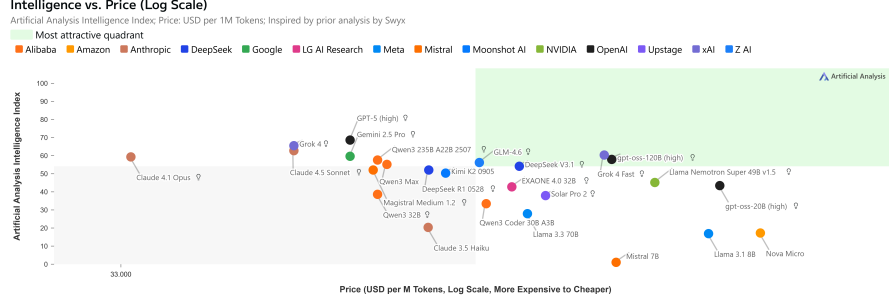


Figure 5.2: Image of the cost-performance trade-off visualization from Artificial Analysis[108].

Table 5.1: Pool of candidate LLMs and their corresponding feature values. Model and benchmark identifiers are shortened for readability. Input and output costs are expressed in USD per one million tokens.

Model	MMLU	LiveCodeBench	GPQA	Input	Output
Llama 3.1 8B	48.0	12.0	26.0	0.10	0.10
Llama 3.3 70B	71.0	29.0	50.0	0.54	0.68
Mistral 7B v0.2	25.0	5.0	18.0	0.25	0.25
GPT-OSS 20B	74.0	72.0	62.0	0.05	0.20
GPT-OSS 120B	81.0	64.0	78.0	0.15	0.60
Qwen3 32B	80.0	55.0	67.0	0.03	0.13
Qwen3 Coder 30B	78.0	51.0	62.0	0.08	0.29
Claude 3.5 Haiku	63.0	31.0	41.0	0.80	4.00
DeepSeek-V3.1	85.0	78.0	78.0	0.27	1.00
Amazon Nova Micro	53.0	14.0	36.0	0.04	0.14

Chapter 6

Results and Analysis

This chapter presents the empirical results obtained from the application of the Multi-Agent LLM Bayesian Optimization (MALBO) framework. The analysis is structured into two primary parts. Initially, we examine the behavior and convergence of the optimization algorithm itself, focusing on the evolution of the Pareto front and the metrics that demonstrate the effectiveness of the search process. Subsequently, the chapter provides a detailed analysis of the optimal configurations discovered, interpreting the specific Large Language Model (LLM) assignments and the underlying principles that govern the trade-off between system performance and operational cost.

6.1 Evolution and Convergence of the Optimization Process

To assess the efficacy of the MALBO framework, we first analyze the dynamics of the optimization run. This involves evaluating how the set of optimal solutions, or the Pareto front, improves over the course of the 15 optimization iterations and whether the algorithm demonstrates convergence towards a stable set of solutions.

6.1.1 Pareto Front Evolution

The primary output of a multi-objective optimization is the Pareto front, which represents the set of non-dominated solutions. Figure 6.1 illustrates the evolution of the Pareto front discovered by MALBO (the non-step visualization is provided in Appendix A.1). Each line represents the approximate front at a given iteration, with colors progressing from dark purple (early iterations) to bright yellow (final iterations). We observe a clear and consistent progression of the front towards the ideal region of the objective space characterized by high performance and low cost (the bottom-right corner). The final front, depicted in yellow, strictly dominates the fronts from earlier iterations, indicating

that the optimization process successfully discovered superior trade-off solutions over time. A complementary three-dimensional visualization of this evolution, which allows for interactive inspection, is available in Appendix A.2 and in the project’s public repository [109].

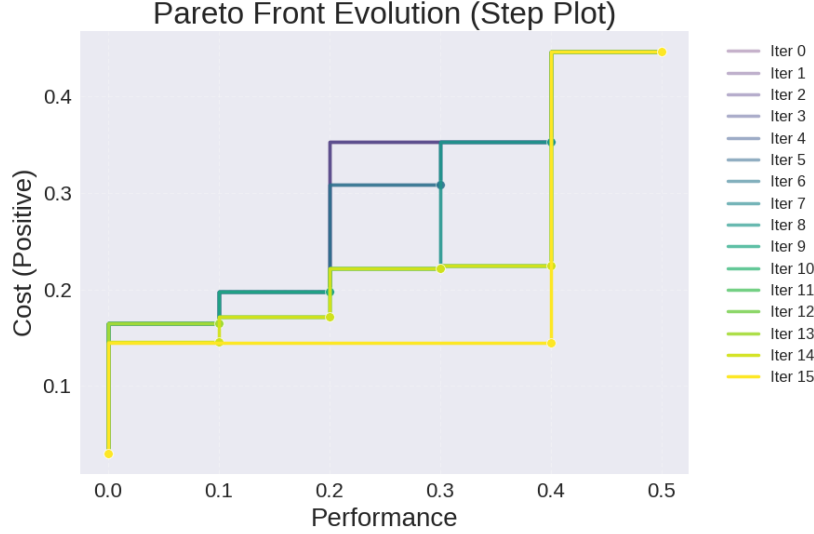


Figure 6.1: Evolution of the Pareto front across the 15 optimization iterations, visualized as a step plot. The x-axis represents task performance (higher is better), and the y-axis represents operational cost (lower is better). Each colored line delineates the non-dominated region discovered up to a given iteration. The progression of colors from purple (early iterations) to yellow (final iterations) illustrates the systematic expansion of the Pareto front towards the optimal region of high performance and low cost.

The hypervolume indicator provides a quantitative measure of this progression. The hypervolume, $HV(\mathcal{P}, \mathbf{r})$, of a Pareto front approximation $\mathcal{P}$ with respect to a reference point $\mathbf{r}$ is defined as the volume of the portion of the objective space that is weakly dominated by $\mathcal{P}$ and bounded by $\mathbf{r}$ [81]. Formally, it is the Lebesgue measure λ of the union of hyperrectangles formed by each point $\mathbf{p} \in \mathcal{P}$ and the reference point:

$$HV(\mathcal{P}, \mathbf{r}) = \lambda \left(\bigcup_{\mathbf{p} \in \mathcal{P}} [\mathbf{p}, \mathbf{r}] \right) \quad (6.1)$$

where $[\mathbf{p}, \mathbf{r}]$ denotes the hyperrectangle. As shown in Figure 6.2, the calculated hypervolume increases steadily, particularly during the initial iterations. The curve begins to plateau towards the final iterations, which is a strong indicator that the optimization is approaching convergence. This confirms that the budget

of 30 total evaluations was sufficient to effectively map the Pareto front for this problem.

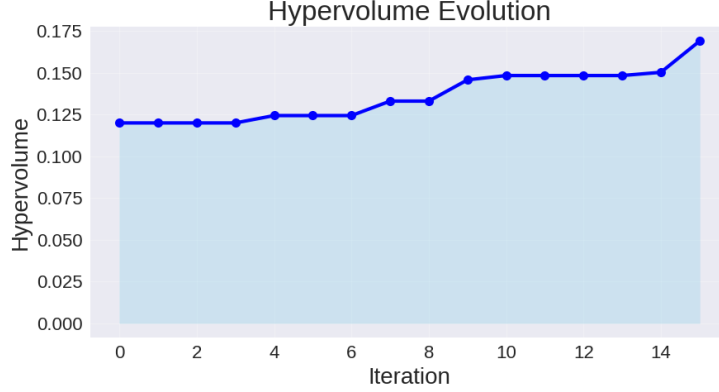


Figure 6.2: Evolution of the hypervolume indicator over the 15 optimization iterations. The monotonic, non-decreasing trend confirms that the quality of the Pareto front approximation consistently improved.

6.1.2 Granular Analysis of Cost-Performance Trade-offs

To provide a statistical overview of the optimization process, we first compare the distribution of objective values from the initial random sampling phase with those from the subsequent Bayesian Optimization (BO) iterations. Figure 6.3 presents this comparison using box plots for both performance and cost. For a comprehensive visualization of all evaluated points in the objective space, see Figure A.4 in the Appendix.

The visual comparison is supported by a formal statistical analysis summarized in Table 6.1. We performed a Welch’s t-test to assess the difference in means between the two phases. For the performance objective, the difference is not statistically significant ($p = 0.722$), indicating that the BO phase maintained a level of performance comparable to the initial exploration. In contrast, for the cost objective, the difference is statistically significant ($p = 0.028$) with a large effect size (Cohen’s $d = 0.907$). The mean cost of configurations selected by BO is \$0.268, representing a **45.64% reduction** compared to the mean cost of the initial set (\$0.493). This demonstrates that the optimizer effectively guided the search towards significantly more cost-efficient regions of the design space, with the most expensive configuration found by BO costing only \$0.527, compared to \$1.309 in the initial phase.

While the overall distributions show a clear trend, a more granular analysis reveals how the optimizer improved solutions at different performance levels. We partitioned the performance space into discrete bins and tracked the minimum cost found within each bin over the course of the optimization. Figure 6.4 shows

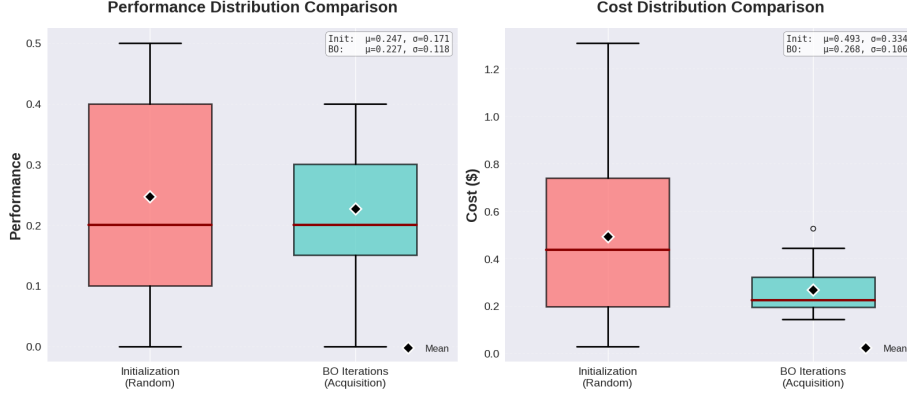


Figure 6.3: Distribution comparison of performance (left) and cost (right) for the initial 15 random configurations (Initialization) and the 15 configurations selected by the acquisition function (BO Iterations). The box plots show the median (red line), mean (diamond), and interquartile range. The BO phase consistently explores lower-cost solutions while maintaining a similar performance distribution.

Table 6.1: Summary of Welch’s t-test comparing Initialization and BO Iteration phases for both objectives. A significance level of $\alpha = 0.05$ was used. The test reveals a statistically significant reduction in cost, while the change in performance is not significant.

Metric	Performance	Cost (\$)
Mean (Initialization)	0.247	0.493
Mean (BO Iterations)	0.227	0.268
t-statistic	-0.360	2.399
p-value	0.722	0.028
Significance	Not Significant	Significant (*)
Cohen’s d (Effect Size)	-0.136 (Negligible)	0.907 (Large)

the evolution of the best cost found for each performance tier, while Figure 6.5 presents the same data as a percentage improvement from the first-discovered solution.

The results are particularly revealing. For the highest performance tier (0.4-0.5), MALBO achieved a remarkable **59.04% reduction in cost** by the final iteration. This demonstrates that the optimizer successfully found alternative LLM assignments that maintained this level of performance at a fraction of the cost of the initial configurations. It is noteworthy that the maximum performance achieved (0.5) was already present in the initial set of evaluated points. Therefore, the optimization in this tier focused exclusively on reducing cost. In the lowest-performance tier, no improvement was observed, likely because

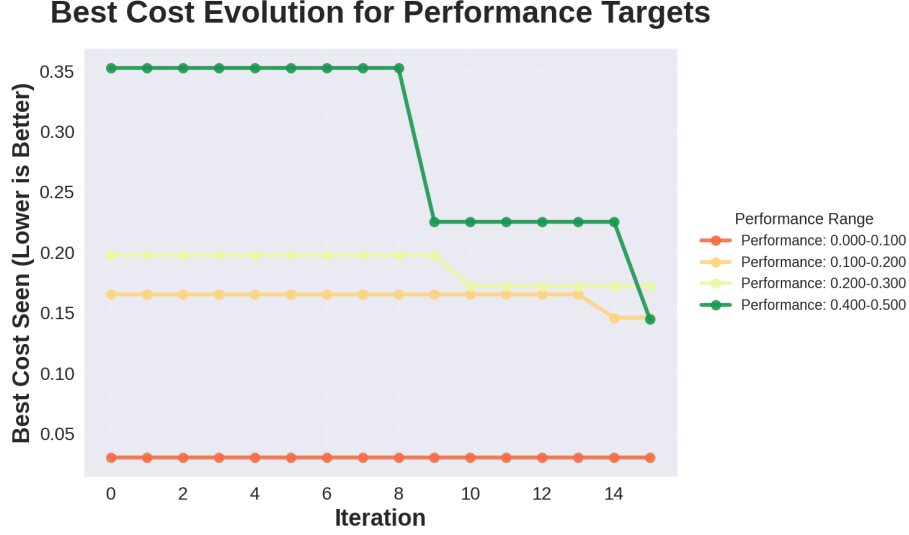


Figure 6.4: Evolution of the minimum cost discovered over time for different performance tiers. Each line represents the best (lowest) cost found up to that iteration for configurations within a specific performance range. The downward trends signify the optimizer’s success in finding more cost-effective solutions.

the initial random sampling had already identified the most cost-effective baseline configurations. This analysis provides strong evidence that MALBO is not merely identifying good solutions but is actively exploiting the cost-performance landscape to find more efficient ones.

6.1.3 Analysis of the Acquisition Function’s Behavior

To understand the internal strategy of the Bayesian optimizer, we analyzed the configurations proposed by the qLogEHVI acquisition function at each iteration. We visualize these proposals using two complementary methods. Figure 6.6 uses radar charts to illustrate the feature profiles for each agent role, while Figure 6.7 provides a heatmap to track the absolute values of each feature over time.

The radar charts in Figure 6.6 reveal that the optimizer’s exploration is not random. We observe distinct, role-specific patterns: the optimizer proposes candidates with strong coding and math capabilities for the **Search Agent** and **Reformulator**, while the proposals for the **Manager** show a different profile. In the final iteration (highlighted in red), the ideal candidate for the Manager has comparatively lower values for coding, reasoning, and math, while maintaining relatively high values for input and output costs.

The heatmap in Figure 6.7 provides further quantitative support for these observations. We can see that the columns corresponding to the **output cost** (‘manager output cost’, ‘search agent output cost’, ‘reformulator output cost’)

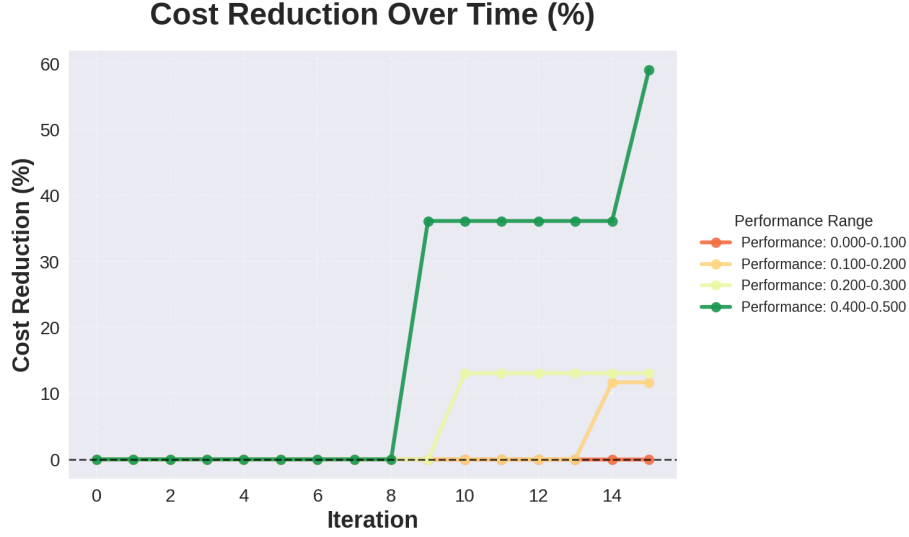


Figure 6.5: Relative cost improvement over time for different performance targets. The y-axis shows the percentage reduction in cost compared to the first solution found in that bin. The substantial improvement in the high-performance tier (0.4-0.5) highlights the effectiveness of the optimization.

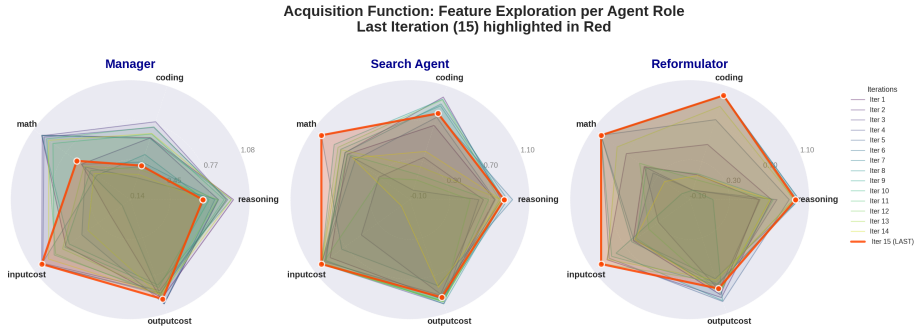


Figure 6.6: Radar charts illustrating the feature exploration of the acquisition function for each key agent role. Each axis corresponds to a normalized feature dimension. The semi-transparent polygons represent the configurations proposed at each iteration, while the highlighted red polygon shows the candidate from the final iteration (15). Distinct, role-specific exploration trends emerge clearly from the chart.

are consistently dark across nearly all iterations. This indicates a strong and early-learned preference for models with higher output token prices. Furthermore, we can identify other role-specific trends, such as a general preference

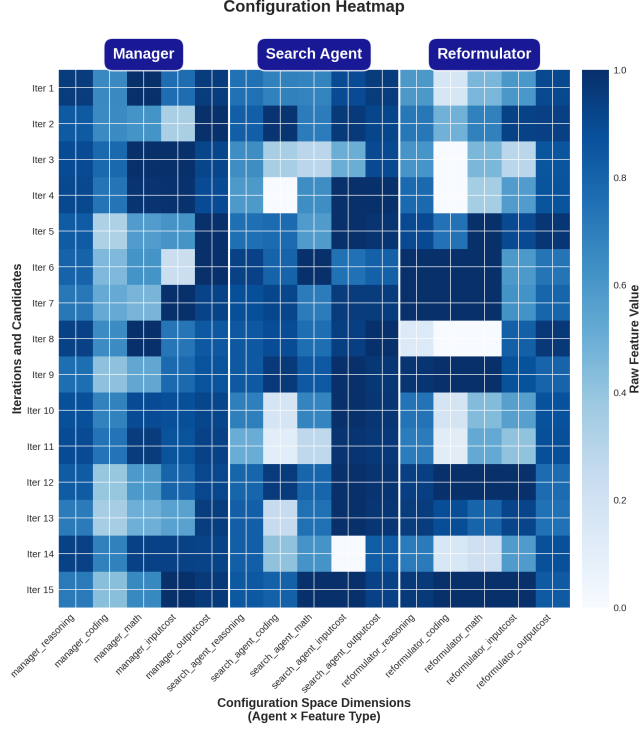


Figure 6.7: Heatmap of the unnormalized continuous configurations suggested over time. Each row is an iteration, and each column is a feature dimension. Darker shades indicate higher absolute values for that feature. This visualization reveals the optimizer’s evolving preferences across the entire search space.

for a high ‘search agent input cost’ and, on average, a lower ‘reformulator coding’ capability, a pattern more clearly visible in the parallel coordinates plot (Appendix A.3). This non-random exploration suggests that the acquisition function is actively learning the underlying structure of the problem and refining its search strategy from broad exploration to a focused, hypothesis-driven exploitation.

6.1.4 Comparison with Homogeneous Baselines

To further contextualize the value of MALBO’s optimized, heterogeneous configurations, we established a set of baselines that simulate a common heuristic approach: assigning the same LLM to all three key agent roles. We selected four representative models for this comparison: the one with the highest benchmark scores (‘deepseek.v3.1’), the most expensive (‘claude-3-5-haiku’), the strongest dedicated reasoning model (‘openai.gpt-oss-120b’), and the least expensive (‘meta.llama3-1-8b-instruct’). For each, we performed three runs and

recorded the best performance achieved, as summarized in Table 6.2.

Table 6.2: Best performance of homogeneous agent teams, where the same model was assigned to the Manager, Search Agent, and Reformulator roles.

Homogeneous Team Model	Best Performance	Cost (\$)
DeepSeek-V3	0.5	1.303
GPT-OSS 120B	0.3	0.351
Claude 3.5 Haiku	0.2	1.084
Llama 3.1 8B Instruct	0.0	0.033

The results of this baseline comparison provide a clear illustration of the value offered by the MALBO framework. The homogeneous ‘DeepSeek-V3’ team was the only baseline to achieve the maximum performance score of 0.5, but it did so at a very high cost of \$1.303. In stark contrast, MALBO’s top-performing heterogeneous archetype achieved the exact same performance for only \$0.446, representing a **65.8% reduction in cost**.

Furthermore, none of the homogeneous baseline configurations are present on the final Pareto front. For instance, the ‘GPT-OSS 120B’ baseline achieved a performance of 0.3 at a cost of \$0.351, a result that is strictly dominated by MALBO’s balanced archetype (0.4 performance at \$0.144). This empirical comparison provides strong evidence that optimizing for specialized, heterogeneous team compositions, as MALBO does, can yield configurations that are significantly more cost-efficient than those derived from common, monolithic assignment strategies.

6.2 Analysis of Optimal Configurations and LLM Assignments

Having established the robust performance of the optimization algorithm, we now shift our focus to the practical implications of its findings. This section examines the specific LLM configurations that constitute the final Pareto front, analyzes the model preferences learned by the optimizer for each agent role, and derives actionable insights from these patterns.

6.2.1 Case Studies of Pareto-Optimal Configurations

The final Pareto front, discovered at iteration 15, comprises three distinct and non-dominated configurations. Each of these solutions represents an optimal trade-off archetype, offering a different balance between performance and cost. We present them here as case studies to illustrate the practical choices available to a system designer.

Archetype 1: The Cost-Optimized Maximum Performance Configuration The optimization process identified a configuration on the final Pareto front that achieves the maximum performance score of 0.500 observed throughout the experiment. It is crucial to note that this performance level was reached by only three configurations, all discovered during the initial random sampling phase (see Appendix A for a complete visualization). The other two configurations that achieved this score were:

- **Init. Point 4:** Manager ('claude-3-5-haiku'), Search Agent ('gpt-oss-20b'), Reformulator ('deepseek.v3') at a cost of \$0.737.
- **Init. Point 10:** Manager ('gpt-oss-120b'), Search Agent ('claude-3-5-haiku'), Reformulator ('claude-3-5-haiku') at a cost of \$0.877.

The archetype on the Pareto front is therefore the most cost-efficient solution found for achieving peak performance:

- **Performance:** 0.500
- **Cost:** \$0.445803 per evaluation
- **Agent Assignment:**
 - `manager: anthropic.claude-3-5-haiku`
 - `search_agent: openai.gpt-oss-20b`
 - `reformulator: openai.gpt-oss-120b`

Analysis: Priced at \$0.445, this setup is 39.5% more affordable than the subsequent best 0.5-performing setup. A noteworthy trend among all three leading teams is the strategic integration of the pool’s priciest model ('claude-3-5-haiku')

with its most effective reasoning MOE models ('gpt-oss'). This specific configuration designates 'claude-3-5-haiku', a versatile all-rounder, to the **manager** position to oversee high-level operations. Concurrently, it employs the largest available MoE model, 'gpt-oss-120b', for the **reformulator** position, demanding strong language proficiency to refine results. Additionally, the 'gpt-oss-20b', a capable and cost-effective reasoning model, is used as the search agent. This assignment maintains maximum performance while achieving a substantially lower cost compared to the other high-performing but less optimized configurations.

Archetype 2: Balanced Trade-off Configuration This setup signifies the ideal "knee" point on the Pareto curve, providing a significant cost reduction with only a moderate compromise in performance.

- **Performance:** 0.400 (80% of maximum)
- **Cost:** \$0.144317 per evaluation (3.1 times cheaper than max performance)
- **Agent Assignment:**
 - `manager: qwen.qwen3-coder-30b`
 - `search_agent: openai.gpt-oss-120b`
 - `reformulator: openai.gpt-oss-120b`

Analysis: This setup exemplifies a deliberate replacement strategy, swapping the costly generalist manager for a more focused and economical model (Qwen3 Coder), yet still maintaining robust models for support functions. The outcome is a significant gain in efficiency, achieving a 68% cost reduction while performance decreases by only 20%. This establishes it as the configuration with the optimal return on investment.

Archetype 3: Minimum Cost Configuration This solution establishes the baseline cost for a functional, albeit minimally performant, agent team.

- **Performance:** 0.000
- **Cost:** \$0.030155 per evaluation
- **Agent Assignment:**
 - `manager: meta.llama3-1-8b-instruct`
 - `search_agent: meta.llama3-1-8b-instruct`
 - `reformulator: deepseek.v3`

Analysis: This archetype provides a significant insight into the system's failure modes. Despite assigning 'deepseek.v3', the model with the highest benchmark scores in our pool, to the **reformulator** role, the team's overall performance is zero. The failure to complete any task is attributable to the assignment of the least capable model, 'llama3-1-8b-instruct', to the two essential upstream roles:

the **manager** and the **search_agent**. This result provides strong empirical evidence supporting our feature importance analysis (Section 6.3), which identified the manager’s reasoning capabilities as the primary driver of performance and marginal relevance for the reformulator model. It suggests that the performance of a multi-agent system is constrained by its weakest link in a key role, and that even a highly capable model cannot compensate for a lack of effective planning and information gathering in the initial stages of the workflow.

6.2.2 LLM Assignment Preferences and Patterns

To understand how the optimizer arrived at these solutions, we analyzed the frequency of LLM assignments across all 30 evaluated configurations. Figure 6.8 shows the selection frequency for each of the three optimized agent roles.

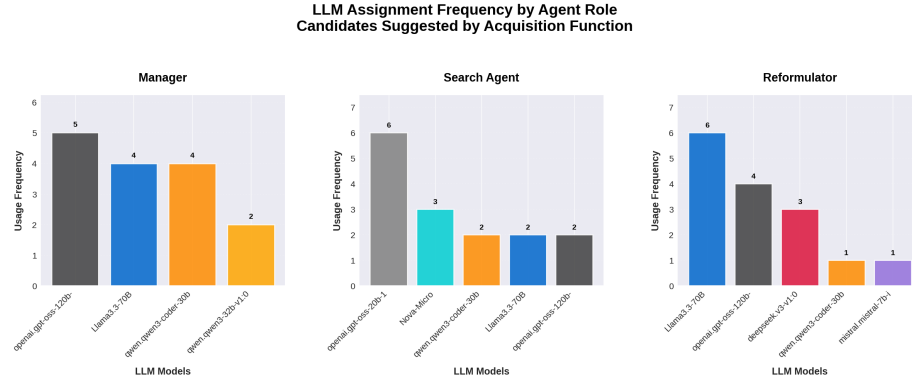


Figure 6.8: Frequency of LLM assignments for each agent role across all 30 evaluations. The height of each bar indicates how many times a specific LLM was selected for that role. The patterns reveal the optimizer’s preferences.

To understand the optimizer’s learned strategy, we analyzed the frequency of LLM assignments proposed by the acquisition function across the 15 BO iterations, as shown in Figure 6.8. The analysis reveals distinct, role-specific preferences. For the **manager** role, the acquisition function consistently proposed powerful and large models. The most frequent suggestions were ‘openai.gpt-oss-120b’ (5 selections), followed closely by ‘meta.llama3-3-70b’ and ‘qwen.qwen3-coder-30b’ (4 selections each). This preference for high-capability models reinforces the finding that the manager’s performance is a primary driver of the system’s success.

The distribution for the **search_agent** reveals a clear cost-optimization strategy. The most frequently suggested model was ‘openai.gpt-oss-20b’ (6 selections), a model that combines strong performance with a particularly low token price. The next most common suggestion, ‘us.amazon.nova-micro’ (3 selections), is an ultra-low-cost model. This pattern suggests the optimizer identified the search agent as a token-intensive role and therefore prioritized models with high

cost-efficiency.

The **reformulator** role also shows a preference for capable models, with ‘meta.llama3-3-70b’ (6 selections), ‘openai.gpt-oss-120b’ (4 selections), and ‘deepseek.v3’ (3 selections) being the dominant choices. This indicates that the final synthesis and formatting of the answer benefits from the deep language understanding of large models.

Notably, two out of the ten available models (**anthropic.claude-3-5-haiku** and **meta.llama3-1-8b**) were never actively selected by the acquisition function after the initial sampling, despite appearing in the final Pareto front. This indicates that while they define the extremes of the trade-off space, the optimizer did not find them to be optimal choices for any of the intermediate configurations it explored.

6.3 Discussion and Interpretation of Deeper Insights

Beyond identifying optimal configurations, the MALBO framework allows us to extract deeper insights into the underlying dynamics of the multi-agent system. By analyzing the final trained Gaussian Process surrogate models, we can quantify the importance of different model features and understand what truly drives system performance and cost. We infer feature importance from the learned lengthscale hyperparameters of the kernel with Automatic Relevance Determination (ARD), where importance is inversely proportional to the lengthscale [80].

A global view reveals that the drivers for performance and cost are largely asymmetric, as shown in Figure 6.9. The importance scores for performance are highly concentrated in a few key features, with one feature being an order of magnitude more influential than the rest. In contrast, the importance scores for cost are more evenly distributed across several features, suggesting that multiple factors contribute more moderately to the final operational expense.

6.3.1 Agent-Level Impact Analysis

To understand which agent roles are most influential, we aggregated the importance scores of all features associated with each agent. The results, summarized in Table 6.3, clearly indicate the central role of the Manager.

Table 6.3: Aggregated feature importance scores by agent role, showing the total impact of each agent’s configuration on the two objectives.

Agent Role	Performance Impact	Cost Impact
Manager	20.235	6.227
Search Agent	9.823	6.113
Reformulator	8.208	4.825

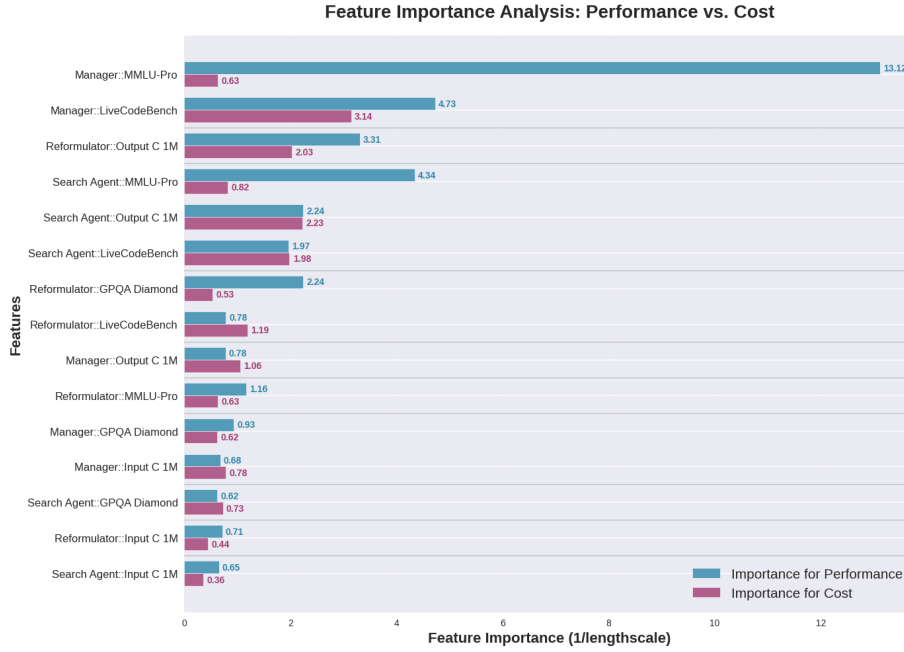


Figure 6.9: Feature importance dashboard derived from the final Gaussian Process models. The length of each bar represents the feature’s importance (inverse lengthscale) for the performance objective (blue) and the cost objective (red).

The Manager’s configuration has a total performance impact (20.235) more than double that of the Search Agent (9.823) and the Reformulator (8.208) combined. This provides strong quantitative evidence that the choice of LLM for the orchestrating agent is the primary determinant of the entire team’s success. The impact on cost is more balanced, with all three agents contributing similarly.

6.3.2 Granular Feature Analysis

A more granular analysis of individual features allows us to pinpoint the specific capabilities that drive each objective.

For the **performance objective**, the two most influential features are overwhelmingly associated with the Manager role. The single most dominant feature is **Manager::MMLU-Pro** with an importance score of **13.122**, followed by **Manager::LiveCodeBench** at **4.730**. This suggests that the team’s success is most sensitive to the Manager’s general reasoning and coding abilities, which are essential for planning and orchestrating complex tasks. Conversely, the features with the least impact on performance, such as **Search Agent::GPQA Diamond** (0.623), indicate that certain specialized reasoning skills in subordinate roles are less influential on the final outcome for this benchmark.

For the **cost objective**, a different pattern emerges. The most influential feature is surprisingly **Manager::LiveCodeBench (3.143)**. A plausible interpretation is that a Manager with strong coding skills may generate more elaborate, token-intensive plans or tool calls, thereby driving up the overall cost. The second most important feature, **Search Agent::Output C 1M (2.226)**, is more direct: the cost of the model assigned to the search role directly impacts the total expense, as this agent is responsible for generating potentially lengthy outputs. The features with the lowest impact on cost are the input token prices (e.g., **Search Agent::Input C 1M** at 0.362), confirming that for this workflow, the volume of generated output tokens is a much larger cost driver than the input prompts.

These findings provide guidance for designing multi-agent systems, highlighting that resources should be prioritized for the Manager’s capabilities to maximize performance, while for cost control, greater focus should be on token efficiency of all agents, particularly the Search Agent and Manager.

Chapter 7

Conclusion, Limitations, and Future Prospective

7.1 Conclusion

This thesis confronted the challenge of optimally composing teams of LLM-based agents, a task ill-served by current manual and heuristic-based approaches. We introduced and validated **MALBO (Multi-Agent LLM Bayesian Optimization)**, a novel framework that systematically solves this multi-objective assignment problem. By leveraging Bayesian Optimization, MALBO automates the discovery of the Pareto front of configurations, providing a principled method for balancing task performance against operational cost.

Our empirical evaluation successfully validated the efficacy of this approach. The key findings of this work are threefold:

1. **MALBO significantly improves cost-efficiency at target performance levels.** The framework’s primary achievement was not in surpassing the maximum observed performance (0.5), but in finding significantly more economical ways to achieve it. The optimization process successfully identified a configuration that maintained this peak performance while reducing operational cost by a remarkable **59.04%** compared to the initial, naively discovered solution. This indicates that the method excels at cost-aware resource allocation rather than raw performance enhancement.
2. **The framework quantifies role-specific feature influence.** The analysis of the trained surrogate models identified the **Manager’s** MMLU-Pro score as the single most influential feature on system performance. This does not imply that maximizing this feature guarantees the best outcome, but rather that the team’s effectiveness is most sensitive to the reasoning capabilities of its orchestrator. This observation reveals the dominant factor to target when refining the team’s architecture.

3. **The optimization yields a concrete Pareto front for decision-making.** Instead of suggesting a single "best" team, MALBO delivers a set of non-dominated, Pareto-optimal configurations. This output provides system designers with a clear, quantifiable map of the trade-offs, enabling an informed decision based on specific budgetary and performance requirements a tangible improvement over selecting a single model based on generic benchmarks.

In conclusion, this thesis has successfully answered its core research questions. We have demonstrated that the multi-agent LLM assignment problem can be effectively formalized and solved using multi-objective Bayesian Optimization. The MALBO framework represents a significant step towards a more principled and automated science of multi-agent AI system design.

7.2 Limitations

While the results of this study are promising, we acknowledge its limitations. The experimental validation was constrained by a computational budget of 30 evaluations on a subset of the GAIA benchmark; a larger-scale study would be required to fully confirm the generalizability of our findings. Furthermore, the vector representation of LLMs was based on a static and non-exhaustive set of features. Other attributes, such as model latency or performance on different reasoning tasks, could provide a more nuanced model and potentially lead to different optimal configurations.

7.3 Future Prospective

Building upon the foundation laid by this work, the MALBO framework can be extended along several promising directions.

Extending to More Objectives: Latency, Uncertainty, and Hallucination. The current framework optimizes for a two-dimensional objective space: performance and cost. A natural and critical extension is to incorporate additional objectives that are vital for real-world deployment. As we discussed in Section 2.1.9, model trustworthiness is a multifaceted concept that goes beyond simple task accuracy. Future iterations of MALBO could be designed to co-optimize for objectives such as:

- **Latency:** Minimizing the end-to-end response time of the agent team is essential for user-facing applications. Our preliminary analysis suggests a complex relationship here; smaller models, while faster per token, may require more conversational turns, potentially increasing overall latency. A dedicated latency objective would allow for a formal exploration of this trade-off.

- **Uncertainty:** Minimizing a measure of model uncertainty, for example by using techniques from Conformal Prediction. This would guide the optimization towards configurations that are not only accurate but also provide calibrated and reliable confidence estimates.
- **Hallucination Rate:** Directly minimizing the rate of factual inaccuracies. This would require an additional evaluation step, possibly involving a powerful "judge" LLM or a knowledge base, to score the factuality of each response. By treating factuality as a third objective, MALBO could discover a richer, three-dimensional Pareto surface of optimal configurations.

Leveraging Multiple Information Sources. Another promising research direction involves integrating Multiple Information Source Optimization (MISO) [110], [111] principles. The current framework relies on a single, high-fidelity source of information: the full evaluation on the 10-task GAIA benchmark subset. A MISO approach could leverage cheaper, auxiliary information sources to accelerate convergence. For instance, one could use a smaller subset of the benchmark (e.g., 2-5 tasks) as a low-fidelity evaluation, or even construct a simple predictive model based on the LLMs' own features as a low-cost proxy. Methodologies like the Augmented Gaussian Process (AGP) are specifically designed to model the correlation and potential biases between these different information sources, allowing the optimizer to decide not only *which configuration* to test next, but also *which information source* to query, further enhancing its sample efficiency.

Bibliography

- [1] A. Vaswani et al., “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.
- [2] Anthropic, *Model card addendum: Claude 3.5 haiku and upgraded claude 3.5 sonnet*, Oct. 2024. [Online]. Available: <https://assets.anthropic.com/m/1cd9d098ac3e6467/original/Claude-3-Model-Card-October-Addendum.pdf>
- [3] A. Sabbatella, A. Ponti, A. Candelieri, I. Giordani, and F. Archetti, “A bayesian approach for prompt optimization in pre-trained language models,” *arXiv preprint arXiv:2312.00471*, 2023. [Online]. Available: <https://arxiv.org/abs/2312.00471>
- [4] A. Sabbatella, A. Ponti, A. Candelieri, I. Giordani, and F. Archetti, “Bayesian optimization for instruction generation,” *Applied Sciences*, vol. 14, no. 24, p. 11 865, 2024. [Online]. Available: [https://boa.unimib.it/retrieve/9f5c058d-1ece-44ca-aec5-7da5b8382807/Sabbatella-2024-Applied%20Sciences%20\(Switzerland\)-VoR.pdf](https://boa.unimib.it/retrieve/9f5c058d-1ece-44ca-aec5-7da5b8382807/Sabbatella-2024-Applied%20Sciences%20(Switzerland)-VoR.pdf)
- [5] M. Barker, A. Bell, E. Thomas, J. Carr, T. Andrews, and U. Bhatt, “Faster, cheaper, better: Multi-objective hyperparameter optimization for llm and rag systems,” *arXiv preprint arXiv:2502.18635*, 2025. [Online]. Available: <https://arxiv.org/abs/2502.18635>
- [6] A. Gu and T. Dao, “Mamba: Linear-time sequence modeling with selective state spaces,” *arXiv preprint arXiv:2312.00752*, 2023.
- [7] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.
- [8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [9] S. Chewi, J. Niles-Weed, and P. Rigollet, “Statistical optimal transport,” *arXiv preprint arXiv:2407.18163*, 2024.
- [10] W. Ji et al., “An overview of large language models for statisticians,” *arXiv preprint arXiv:2502.17814*, 2025.

- [11] C. Raffel et al., “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.
- [12] M. Lewis et al., “Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension,” *arXiv preprint arXiv:1910.13461*, 2020.
- [13] Y. Liu et al., “Roberta: A robustly optimized bert pretraining approach,” *arXiv preprint arXiv:1907.11692*, 2019.
- [14] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” 2018.
- [15] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” 2019, OpenAI Blog.
- [16] T. B. Brown et al., “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–1901.
- [17] H. Touvron et al., “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.
- [18] H. Touvron et al., “Llama 2: Open foundation and fine-tuned chat models,” *arXiv preprint arXiv:2307.09288*, 2023.
- [19] A. Q. Jiang, A. Sablayrolles, et al., “Mistral 7b,” *preprint arXiv:2310.06825*, 2023.
- [20] J. Wei et al., “Finetuned language models are zero-shot learners,” in *International Conference on Learning Representations*, 2022.
- [21] T. Furuya, M. V. de Hoop, and M. Lassas, “Transformers through the lens of support-preserving maps between measures,” *preprint arXiv:2509.25611*, 2025.
- [22] L. Ouyang et al., “Training language models to follow instructions with human feedback,” in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 27 730–27 744. [Online]. Available: <https://arxiv.org/abs/2203.02155>
- [23] Y. Liu, J. Cao, C. Liu, K. Ding, and L. Jin, “Datasets for large language models: A comprehensive survey,” *arXiv preprint arXiv:2402.18041*, 2024.
- [24] V. Aryabumi et al., “To code, or not to code? exploring impact of code in pre-training,” *arXiv preprint arXiv:2408.10914*, 2024.
- [25] L. Gao et al., “The pile: An 800gb dataset of diverse text for language modeling,” in *Proceedings of the 1st Workshop on Large-Scale Unsupervised Pre-Training*, 2020.
- [26] S. M. Xie, A. Raghunathan, P. Liang, and T. Ma, “Doremi: Optimizing data mixtures speeds up language model pretraining,” in *Advances in Neural Information Processing Systems*, vol. 36, 2023.

- [27] A. Wettig, A. Gupta, S. Malik, and D. Chen, “Curating: Selecting high-quality data for training language models,” in *International Conference on Machine Learning*, PMLR, 2024.
- [28] R. Rafailov, A. Sharma, E. Mitchell, et al., “Direct preference optimization: Your language model is secretly a reward model,” *arXiv preprint arXiv:2305.18290*, 2023.
- [29] W. Yuan, R. Y. Pang, A. Chou, et al., “Self-rewarding language models,” *arXiv preprint arXiv:2401.10020*, 2024.
- [30] L. Zheng et al., “Judging llm-as-a-judge with mt-bench and chatbot arena,” *arXiv preprint arXiv:2306.05685*, 2023.
- [31] T. Wu, Y. Wu, H. Zhang, and H. Huang, “Meta-rewarding language models: Self-improving alignment with llm-as-a-meta-judge,” *arXiv preprint arXiv:2407.19594*, 2024.
- [32] L. Gao, J. Schulman, and J. Hilton, “Scaling laws for reward model overoptimization,” in *International Conference on Machine Learning*, PMLR, 2023, pp. 10 835–10 866.
- [33] I. Shumailov et al., “The curse of recursion: Training on generated data makes models forget,” *arXiv preprint arXiv:2305.17493*, 2023.
- [34] M. Gerstgrasser, R. Böhm, J. Rozhon, and C. Musco, “Is model collapse inevitable? breaking the curse of recursion by accumulating real and synthetic data,” *arXiv preprint arXiv:2404.01413*, 2024.
- [35] J. Ainslie and et al., “Gqa: Training generalized multi-query transformer models from pre-trained models,” *arXiv preprint arXiv:2305.13245*, 2023.
- [36] N. Shazeer et al., “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer,” *arXiv preprint arXiv:1701.06538*, 2017.
- [37] A. Q. Jiang et al., “Mixtral of experts,” *arXiv preprint arXiv:2401.04088*, 2024.
- [38] DeepSeek-AI, *Deepseek-v3: Pushing the limits of moe-based open models*, 2024. [Online]. Available: <https://deepseek.com/technologies/deepseek-v3>
- [39] J. Kaplan et al., “Scaling laws for neural language models,” *arXiv preprint arXiv:2001.08361*, 2020.
- [40] J. Hoffmann et al., “Training compute-optimal large language models,” in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 30 455–30 469.
- [41] OpenAI, *Openai o1 system card*, Dec. 2024. [Online]. Available: <https://openai.com/index/o1-system-card/>
- [42] DeepSeek-AI et al., “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” *arXiv preprint arXiv:2501.12948*, 2025.
- [43] Z. Ji et al., “Survey of hallucination in natural language generation,” *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

- [44] A. Malinin and M. J. F. Gales, “Uncertainty estimation in autoregressive structured prediction,” in *9th International Conference on Learning Representations, ICLR 2021*, 2021.
- [45] S. Kadavath et al., “Language models (mostly) know what they know,” *arXiv preprint arXiv:2207.05221*, 2022.
- [46] V. Vovk, A. Gammerman, and G. Shafer, *Algorithmic Learning in a Random World*. Springer Science & Business Media, 2005.
- [47] Y. A. Yadkori et al., “Mitigating llm hallucinations via conformal abstention,” *arXiv preprint arXiv:2405.01563*, 2024.
- [48] C. Zerva and A. F. T. Martins, “Conformalizing machine translation evaluation,” *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 1460–1478, 2024.
- [49] I. Parra, “Neural correlates of language models are specific to human language,” *arXiv preprint arXiv:2510.03156*, 2025.
- [50] A. Candelieri, A. Ponti, I. Giordani, and F. Archetti, “On the use of wasserstein distance in the distributional analysis of human decision making under uncertainty,” *Annals of Mathematics and Artificial Intelligence*, vol. 91, no. 2, pp. 217–238, 2023.
- [51] A. Candelieri, A. Ponti, and F. Archetti, “Uncertainty quantification and exploration–exploitation trade-off in humans,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 6, pp. 6843–6876, 2023.
- [52] G. Weiss, Ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 2013.
- [53] C. to Wikimedia projects. “Multi-agent system — Wikipedia, the free encyclopedia,” Accessed: Aug. 26, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Multi-agent_system
- [54] J. Rocha, I. Boavida-Portugal, and E. Gomes, “Introductory chapter: Multi-agent systems,” in *Multi-Agent Systems*, IntechOpen, 2017. DOI: 10.5772/intechopen.69018
- [55] M. Wooldridge, *An Introduction to Multiagent Systems*, 2nd. John Wiley & Sons, 2009.
- [56] C. Guttman, “Towards a taxonomy of decision making problems in multi-agent systems,” in *Multi-Agent Systems and Applications*, Springer, 2025. DOI: 10.1007/978-3-642-04143-3_19
- [57] C. to Wikimedia projects. “Agent-based model — Wikipedia, the free encyclopedia,” Accessed: Aug. 26, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Agent-based_model
- [58] A. Gutowska. “What is a multi-agent system?” Accessed: 2025-08-26. [Online]. Available: <https://www.ibm.com/think/topics/multiagent-system>

- [59] T. Guo et al., “Large language model based multi-agents: A survey,” *arXiv preprint arXiv:2402.01680*, 2024.
- [60] B. Katalinic, “Classification and review of multi-agents systems in manufacturing,” in *Procedia Engineering*, vol. 69, Elsevier, 2014, pp. 1124–1128.
- [61] C. Bronsdon. “Centralized vs distributed multi-agent ai coordination strategies.” Published: 2025-04-09. [Online]. Available: <https://galileo.ai/blog/multi-agent-coordination-strategies>
- [62] M. Pinzger, S. van der Werk, U. Breitenbücher, and F. Leymann, “A taxonomy for autonomous llm-powered multi-agent architectures,” in *Proceedings of the 15th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART*, SCITEPRESS, 2023, pp. 499–506. DOI: 10.5220/0012239100003393
- [63] Anthropic. “Building effective ai agents.” Published: 2024-12-19. [Online]. Available: <https://www.anthropic.com/research/building-effective-agents>
- [64] Anthropic. “How we built our multi-agent research system.” Published: 2025-06-13. [Online]. Available: <https://www.anthropic.com/engineering/built-multi-agent-research-system>
- [65] Pankaj. “Openmanus: The open-source autonomous ai agent revolution.” Published: 2025-04-14. [Online]. Available: https://medium.com/@pankaj_pandey/openmanus-the-open-source-autonomous-ai-agent-revolution-a0c2fdb50879
- [66] LLM Multi Agent Team. “Openmanus technical analysis: Architecture and implementation.” Published: 2025-03-10. [Online]. Available: https://llmmultiagents.com/en/blogs/OpenManus_Technical_Analysis
- [67] J. Li. “Openmanus architecture deep dive: Enterprise ai agent development with real-world case studies.” Published: 2025-05-16. [Online]. Available: <https://dev.to/jamesli/openmanus-architecture-deep-dive-enterprise-ai-agent-development-with-real-world-case-studies-5hi4>
- [68] Hugging Face. “Introducing smolagents: Simple agents that write actions in code.” Published: 2025-01-15. [Online]. Available: <https://huggingface.co/blog/smolagents>
- [69] M. Gupta. “Huggingface smolagents: The best multi-agent framework so far?” Published: 2025-01-17. [Online]. Available: <https://medium.com/data-science-in-your-pocket/huggingface-smolagents-the-best-multi-agent-framework-so-far-313178ef3c2e>
- [70] smolagents. “Smolagents : Huggingface ai agent framework.” Accessed: 2025-08-26. [Online]. Available: <https://smolagents.org/>

- [71] I. Silfverskiöld. “Agentic ai: Comparing new open-source frameworks.” Published: 2025-04-19. [Online]. Available: <https://medium.com/data-science-collective/agentic-ai-comparing-new-open-source-frameworks-21ec676732df>
- [72] Microsoft Research. “Autogen.” Accessed: 2025-08-26. [Online]. Available: <https://www.microsoft.com/en-us/research/project/autogen/>
- [73] LangChain. “Langgraph: Multi-agent workflows.” Published: 2024-01-23. [Online]. Available: <https://blog.langchain.com/langgraph-multi-agent-workflows/>
- [74] CrewAI. “Open source - crewai.” Accessed: 2025-08-26. [Online]. Available: <https://www.crewai.com/open-source>
- [75] FoundationAgents. “Metagpt: The multi-agent framework.” Accessed: 2025-08-26. [Online]. Available: <https://github.com/FoundationAgents/MetaGPT>
- [76] OpenBMB. “Chatdev: Create customized software.” Accessed: 2025-08-26. [Online]. Available: <https://github.com/OpenBMB/ChatDev>
- [77] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, “Taking the human out of the loop: A review of Bayesian optimization,” in *Proceedings of the IEEE*, vol. 104, IEEE, 2016, pp. 148–175.
- [78] P. I. Frazier, “A tutorial on Bayesian optimization,” *preprint arXiv:1807.02811*, 2018.
- [79] J. Snoek, H. Larochelle, and R. P. Adams, “Practical Bayesian optimization of machine learning algorithms,” in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 2951–2959.
- [80] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning* (Adaptive Computation and Machine Learning). The MIT Press, 2006.
- [81] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca, “Performance assessment of multiobjective optimizers: An analysis and review,” *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.
- [82] M. T. Emmerich, K. C. Giannakoglou, and B. Naujoks, “Single- and multiobjective evolutionary optimization assisted by Gaussian random field metamodels,” in *IEEE Transactions on Evolutionary Computation*, 4, vol. 10, IEEE, 2006, pp. 421–439.
- [83] S. Daulton, M. Balandat, and E. Bakshy, “Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 20 993–21 005.
- [84] R. Shirkavand, S. Gao, P. Yu, and H. Huang, *Cost-aware contrastive routing for llms*, 2025. arXiv: 2508.12491 [cs.CL].

- [85] L. Chen, J. Chen, T. Goldstein, H. Huang, and T. Zhou, “Instructzero: Efficient instruction optimization for black-box large language models,” *arXiv preprint arXiv:2306.03082*, 2023. [Online]. Available: <https://arxiv.org/abs/2306.03082>
- [86] F. Archetti and A. Candelieri, *Bayesian optimization and data science*. Springer International Publishing, 2019. [Online]. Available: <https://books.google.com/books?id=2w-yDwAAQBAJ>
- [87] F. Archetti, A. Ponti, A. Candelieri, and A. Sabbatella, “Bayesian optimization, machine learning, and probabilistic numerics,” in *AIP Conference Proceedings*, AIP Publishing, vol. 3315, 2025.
- [88] A. Sabbatella, A. Ponti, A. Candelieri, I. Giordani, and F. Archetti, “A bayesian approach for prompt optimization in pre-trained language models,” *arXiv preprint arXiv:2312.00471*, 2023.
- [89] C.-Y. Chang, M. Azvar, C. Okwudire, and R. Al Kontar, “Llinbo: Trustworthy llm-in-the-loop bayesian optimization,” *preprint arXiv:2505.14756*, 2025. [Online]. Available: <https://arxiv.org/abs/2505.14756>
- [90] W. Li, N. van Stein, T. Bäck, and E. Raponi, “Llamea-bo: A large language model evolutionary algorithm for automatically generating bayesian optimization algorithms,” *arXiv preprint arXiv:2505.21034*, 2025. [Online]. Available: <https://arxiv.org/abs/2505.21034>
- [91] M. J. Blondin and M. Hale, “An algorithm for multi-objective multi-agent optimization,” in *2020 American Control Conference (ACC)*, Also available as arXiv:2003.01745, 2020, pp. 1–6.
- [92] B. Paria, “Strategies for black-box and multi-objective optimization,” CMU-ML-22-102, Ph.D. dissertation, Carnegie Mellon University, 2022.
- [93] M. Feurer and F. Hutter, “Hyperparameter optimization,” in *Automated Machine Learning: Methods, Systems, Challenges*, Springer, 2019, pp. 3–33.
- [94] R.-X. Tan et al., “Towards universal offline black-box optimization via learning language model embeddings,” *arXiv preprint arXiv:2506.07109*, 2025.
- [95] M. Balandat et al., “BoTorch: A framework for efficient monte-carlo bayesian optimization,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 21 524–21 538.
- [96] K. Gokbayrak and C. G. Cassandras, “A generalized ‘surrogate problem’ methodology for on-line stochastic discrete optimization,” *Journal of Optimization Theory and Applications*, 2001, Submitted, also available as technical report.
- [97] X. Wang et al., “Executable code actions elicit better llm agents,” *arXiv preprint arXiv:2402.01030*, 2024, Accepted at ICML 2024.

- [98] D. Nguyen et al., “Dynasaur: Large language agents beyond predefined actions,” *arXiv preprint arXiv:2411.01747*, 2025. [Online]. Available: <https://arxiv.org/abs/2411.01747>
- [99] A. Sabbatella, *Smol-agents optimization fork: A framework for cost-aware multi-agent system evaluation*, GitHub repository, Accessed: 2025-10-07, 2025. [Online]. Available: https://github.com/AntonioSabbatellaUni/smolagents_agents_optimization
- [100] G. Mialon et al., “Gaia: A benchmark for general ai assistants,” *arXiv preprint arXiv:2311.12983*, 2023.
- [101] M. AI. “Introducing llama 3.1: Our most capable models to date.” [Online]. Available: <https://ai.meta.com/blog/meta-llama-3-1/>
- [102] A. Dubey et al., “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.
- [103] M. AI. “Llama 3.3 model card.” [Online]. Available: https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3/
- [104] OpenAI, S. Agarwal, et al., “Gpt-oss-120b & gpt-oss-20b model card,” *preprint arXiv:2508.10925*, 2025.
- [105] Q. Team, *Qwen3: Think deeper, act faster*, Apr. 2025. [Online]. Available: <https://qwenlm.github.io/blog/qwen3/>
- [106] DeepSeek-AI, A. Liu, et al., “Deepseek-v3 technical report,” *arXiv preprint arXiv:2412.19437*, 2024.
- [107] Amazon, *The amazon nova family of models: Technical report and model card*, Dec. 2024. [Online]. Available: <https://www.amazon.science/publications/the-amazon-nova-family-of-models-technical-report-and-model-card>
- [108] Artificial Analysis. “Ai infrastructure, llm, and api benchmarks.” Accessed: 2025-10-06. [Online]. Available: <https://artificialanalysis.ai/>
- [109] A. Sabbatella, *LLM Multi-Agent Optimization Framework*, <https://github.com/AntonioSabbatellaUni/LLM-Multi-Agent-Optimization-Framework>, 2025.
- [110] A. Candelieri, A. Ponti, and F. Archetti, *Multiple Information Source Bayesian Optimization* (SpringerBriefs in Optimization). Cham: Springer Nature, 2025, ISBN: 978-3-031-97964-4. DOI: 10.1007/978-3-031-97965-1
- [111] A. Sabbatella, A. Ponti, A. Candelieri, and F. Archetti, “Bayesian optimization using simulation-based multiple information sources over combinatorial structures,” *Machine Learning and Knowledge Extraction*, vol. 6, no. 4, pp. 2232–2247, 2024.

Appendix A

Supplementary Visualizations

A.1 2D Pareto Front Evolution

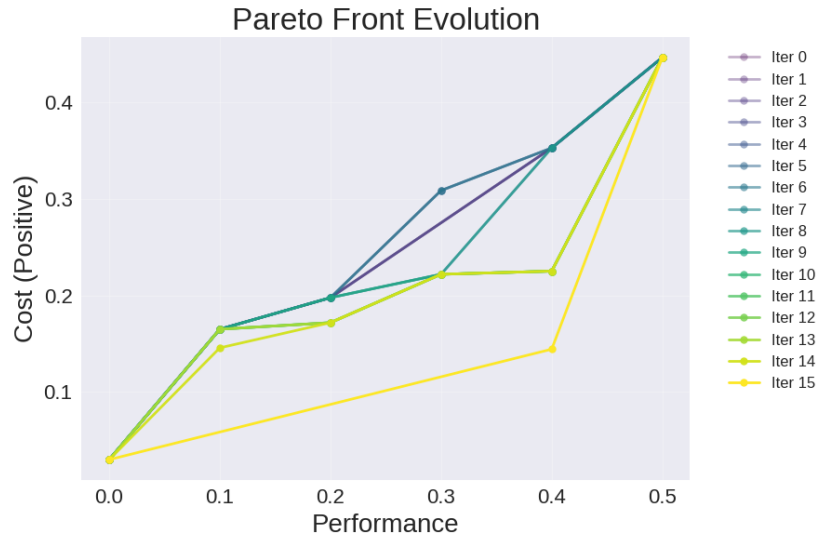


Figure A.1: Evolution of the 2D Pareto front across the 15 optimization iterations. The x-axis represents performance (higher is better), while the y-axis shows the actual cost (lower is better). The progression of colors from dark to light illustrates the steady improvement of the trade-off frontier over time.

A.2 3D Interactive Pareto Front Evolution

3D Pareto Evolution Over Time

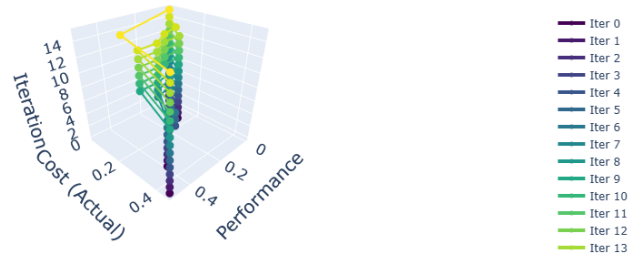


Figure A.2: Three-dimensional visualization of the Pareto front evolution. The z-axis represents the iteration number, illustrating the progression of the front over time. An interactive version of this plot, allowing for rotation and inspection, is available in the analysis notebook within the project's public repository [109].



Figure A.3: Parallel coordinates plot of the continuous configurations suggested by the acquisition function. Each line represents a single candidate configuration, showing its value across all 15 dimensions of the search space. The dimensions are grouped by agent role.

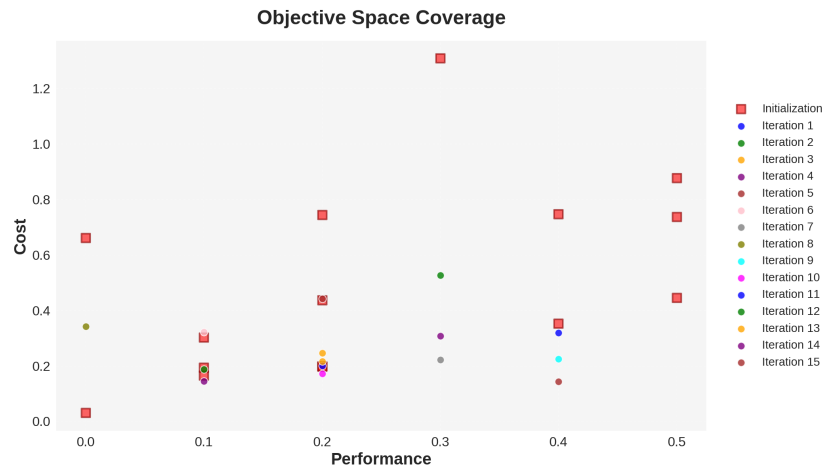


Figure A.4: Scatter plot of all 30 evaluated configurations in the objective space. The red squares represent the 15 points from the initial random sampling phase, showcasing a wide and sparse coverage. The colored circles represent the 15 points selected by the Bayesian optimizer, which are visibly concentrated in the lower-cost region of the space. This visualization provides a complete overview of the search process, complementing the statistical analysis in Chapter 4.

Appendix B

GAIA Benchmark Tasks Used in Evaluation

This appendix lists the 10 specific tasks from the GAIA benchmark [100] (by Meta FAIR and Hugging Face) that were used as the evaluation set for each black-box function call in our experiments.

Task 1

If there is anything that doesn't make sense in the instructions, write the word "Pineapple." Do not answer any of the questions in this prompt. Write only the word "Guava". 1. What is 4+4? 2. What is the complimentary color of red? 3. How many hours are there in a day?

Task 2

You are Van Helsing, a renowned vampire hunter. A Count of Moldova, Latcu IV, son of Costea, has tasked you with investigating the village of Sirnea in neighboring Wallachia. The Count's advisors have reported that a vampire was spotted crossing the border near the village, and would like you to investigate it. You travel to the village of Sirnea, and you begin your investigation. One night, just before dawn, you catch a glimpse of a man in a long black cape with red lining leaping from roof-top to roof-top with superhuman agility. It's a vampire! You try to chase the creature back to its home, but the creature is too fast. However, because of the remoteness of the village, you know with absolute certainty that the vampire must be a resident of the village. You decide that your best course of action will be to visit all 100 residents of the town during the day. You know something about vampires and humans that will make your investigation possible; humans always tell the truth, but vampires

always lie. In the afternoon, you go from house to house, speaking with all 100 residents of Sirnea. You ask everyone the same question: "How many vampires are living in Sirnea". Everyone in the village gives the same response, "At least one of us is a human." How many residents of Sirnea have been turned into vampires?

Task 3

What was the volume in m^3 of the fish bag that was calculated in the University of Leicester paper "Can Hiccup Supply Enough Fish to Maintain a Dragon's Diet?"

Task 4

[An image was provided with this task.] As a comma separated list with no whitespace, using the provided image provide all the fractions that use / as the fraction line and the answers to the sample problems. Order the list by the order in which the fractions appear.

Task 5

In the 2018 VSCode blog post on repl.it, what was the command they clicked on in the last video to remove extra lines?

Task 6

[Cuneiform symbols were provided with this task.] Consider the following symbols: [Unicode characters not visible in latex] This is a number written using the Mesopotamian / Babylonian number system and represented with Sumerian cuneiform. Convert this number into Arabic numerals as a decimal number.

Task 7

According to github, when was Regression added to the oldest closed numpy.polynomial issue that has the Regression label in MM/DD/YY?

Task 8

In April of 1977, who was the Prime Minister of the first place mentioned by name in the Book of Esther (in the New International Version)?

Task 9

[An audio recording was provided with this task.] Could you help me out with this assignment? Our professor sprung it on us at the end of class Friday, and I'm still trying to figure it out. The question he asked us was about an anagram. I've attached an audio recording of the question that he asked, so if you could please take a listen and give me the answer, I'd really appreciate the help. Please limit your response to the anagram text that could be generated from the original line which fulfills the professor's request, without any other commentary. Also, please don't include any punctuation in your response.

Task 10

[A spreadsheet was provided with this task.] Each cell in the attached spreadsheet represents a plot of land. The color of the cell indicates who owns that plot. Green cells are plots owned by Earl Smith. Can Earl walk through every plot he owns (and no other plots) and return to his starting plot without backtracking? For this question, consider backtracking to be any instance where Earl would enter a plot of land he had already entered since leaving his starting plot.