BOAgent: A Multi-Agent Framework for Automating and Generalizing LLM-Driven Bayesian Optimization

Tina Xinyu Hou
MPhil in Data Intensive Science
University of Cambridge
xh363@cam.ac.uk

Abstract

Bayesian optimization (BO) is a fundamental technique for optimizing black-box functions, and recent approaches like LLAMBO have leveraged Large Language Models (LLMs) to enhance BO by translating optimization problems into natural language. LLAMBO integrates LLMs into components such as warm-starting, surrogate modeling, and candidate sampling, demonstrating effectiveness on diverse benchmarks. However, it faces limitations in scalability and generalization: (1) experiments are confined to low-dimensional tasks like hyper-parameter tuning (HPT), potentially limiting applicability to higher-dimensional or different tasks; (2) manually designed prompts for specific tasks restrict its generalization capabilities; and (3) prompt engineering acts as an additional hyper-parameter requiring careful tuning, which is time-consuming and may need revision when changing LLMs. To address these limitations, we propose **BOAgent**, a universal and efficient multi-agent framework for BO problems. BOAgent automates the entire BO process, including prompt generation and optimization for each BO component, thus eliminating the need for manual prompt engineering. In our framework, a system agent plans the steps required to perform the BO task based on the problem description, and specific functional agents execute each step. We implement key components of BOAgent: (1) planning generation for the BO problem and (2) prompt generation and optimization for components such as warm-starting, surrogate modeling, and candidate sampling. We compare BOAgent with LLAMBO by evaluating their methodologies and prompts. Our experiments demonstrate that BOAgent is not only feasible but also effective for general BO tasks, effectively addressing the scalability and generalization issues present in LLAMBO. Furthermore, due to its generalization capabilities, the BOAgent framework can be readily applied to domains like healthcare, facilitating optimization tasks in this critical field. For detailed implementation code, we refer readers to our public repository at: `https://github.com/TinaXYH/BOAgent`.

1 Introduction

Bayesian Optimization (BO) is a widely used method for optimizing expensive black-box functions, crucial in fields such as robotics, experimental design, and hyperparameter tuning (HPT). By constructing surrogate models and acquisition functions, BO iteratively balances exploration and exploitation to find optimal solutions. However, its scalability and generalization capabilities face challenges, particularly in high-dimensional spaces or when dealing with sparse observations and misspecified models.

Recent advances have sought to address these challenges by integrating the strengths of Large Language Models (LLMs) into the BO framework. LLAMBO (Large Language Models to Enhance

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Bayesian Optimization), proposed by Liu et al. [2024], represents a pioneering step in this direction. By framing BO tasks in natural language, LLAMBO leverages the contextual understanding, few-shot learning abilities, and encoded domain knowledge of LLMs to enhance components like warm-starting, surrogate modeling, and candidate sampling. The method demonstrated strong empirical performance in low-dimensional HPT tasks, showcasing the utility of zero-shot and few-shot approaches without requiring extensive fine-tuning. Despite these successes, LLAMBO's reliance on manually crafted prompts and its focus on low-dimensional tasks limit its scalability and applicability. Moreover, prompt engineering introduces additional complexity, acting as a hyperparameter that requires careful tuning for each new task or LLM.

Parallel to the advancements in BO, the development of agent-based frameworks has significantly expanded the scope of automated and scalable solutions across domains. Agent K v1.0, for example, exemplifies the potential of structured reasoning frameworks for automating complex data science workflows (Grosnit et al. [2024]). By incorporating long- and short-term memory modules, this agent adapts dynamically to diverse tasks without traditional fine-tuning or backpropagation. Applied to data science competitions like Kaggle, Agent K autonomously handles entire pipelines—from data cleaning and feature engineering to model training and submission—achieving performance levels comparable to Kaggle Grandmasters. These developments highlight the transformative role of multi-agent systems in solving real-world problems by decomposing tasks into modular components and leveraging dynamic feedback for continuous improvement.

Building on these advances, we propose **BOAgent**, a universal and efficient multi-agent framework that automates the entire BO process. Unlike LLAMBO, which is confined to manually crafted prompts and low-dimensional tasks, BOAgent introduces a system agent that dynamically plans BO workflows and functional agents that execute key components such as surrogate modeling and candidate sampling. By automating prompt generation and optimization, BOAgent eliminates the need for manual prompt engineering, enhancing scalability and generalization. Our experiments demonstrate BOAgent's efficacy across diverse benchmarks, effectively addressing the limitations of LLAMBO. Furthermore, its adaptability makes it suitable for critical applications in fields like healthcare, where efficient optimization frameworks are essential.

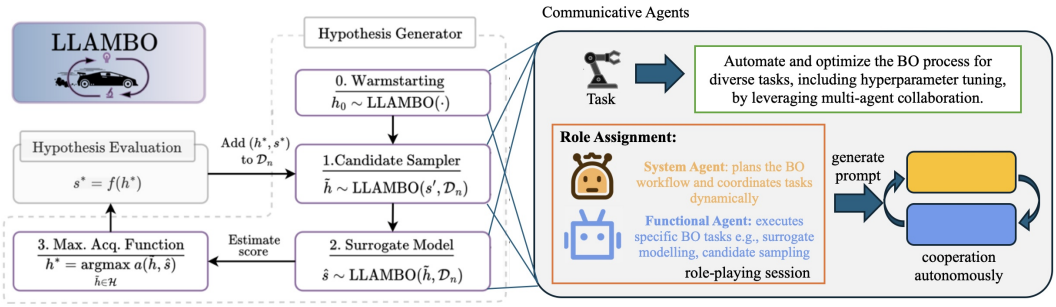

Figure 1.1: Overview of BOAgent. A **System Agent** plans and coordinates the BO workflow, while **Functional Agents** execute key tasks such as surrogate modeling and candidate sampling, automating and generalizing Bayesian Optimization for diverse tasks.

This paper contributes to the advancement of BO by introducing BOAgent, a framework that addresses scalability and generalization challenges through automation and multi-agent collaboration. By bridging the gap between the flexibility of agent frameworks and the optimization power of BO, BOAgent opens new avenues for research and practical applications in complex domains.

2 Related Work

Recent advancements have explored extending the Large Language Model (LLM)-assisted Bayesian Optimization (BO) framework LLAMBO to address diverse challenges in optimization. These efforts can be grouped into extensions focusing on task domains, acquisition functions, integration with human expertise, and novel frameworks combining BO with advanced computational techniques. Below, we categorize and summarize the contributions of key works.

2.1 Extensions to New Domains and Tasks

Autoformulation of Mathematical Optimization Astorga et al. [2024] propose an autoformulation framework that utilizes LLMs to construct optimization models from natural language descriptions. This work employs LLMs as dynamic hypothesis generators and correctness evaluators within a Monte-Carlo Tree Search framework, addressing challenges such as defining hypothesis spaces, systematic exploration, and correctness evaluation. (paper: Autoformulation of Mathematical Optimization Models Using LLMs).

LLM Processes for Numerical Prediction Requeima et al. [2024] introduce LLM Processes (LLMPs) for regression and density estimation tasks. By conditioning LLMs on natural language and numerical data, this work enables the integration of user-defined priors and enhances predictive modeling in black-box optimization.

SysCaps for Complex Systems Emami et al. [2024] explore "SysCaps," a language-driven surrogate modeling approach for complex energy systems. LLMs are used to generate natural language captions from simulation metadata, facilitating intuitive interfaces for model interaction and enhancing generalization.

Analog Layout Constraint Generation Chen et al. [2024] propose LLANA, a framework that leverages LLMs for analog layout synthesis. Analog layout tasks, traditionally manual and resource-intensive, benefit from the few-shot learning capabilities of LLMs. LLANA uses LLM-enhanced BO to generate design-dependent parameter constraints, such as net weightings, improving layout efficiency and reducing dependency on extensive data. Experimental results demonstrate that LLANA surpasses existing BO-based approaches in sample efficiency and layout performance.

2.2 Enhancements in Acquisition Functions

EvolCAF: Cost-Aware Acquisition Functions Yao et al. [2024] introduce EvolCAF, which integrates LLMs with evolutionary computation to design cost-aware acquisition functions (AFs). This framework optimizes BO under heterogeneous cost constraints, reducing reliance on domain expertise and improving interpretability.

FunBO: Learning Acquisition Functions with FunSearch Aglietti et al. [2024] propose FunBO, a framework leveraging LLMs to discover novel AFs through iterative improvement. By producing interpretable code snippets, FunBO addresses generalization and interpretability challenges in AF design.

2.3 Human-In-The-Loop Optimization

Expert-Guided BO Xu et al. [2024] introduce a principled framework for incorporating human expertise into BO through binary labels. Their work addresses the reliability and efficiency of expert input, ensuring convergence without degrading performance due to adversarial advice.

2.4 Integrated Frameworks and Hybrid Models

SLLMBO: Sequential LLM-Based HPO Mahammadli [2024] propose SLLMBO, combining LLMs with a Tree-structured Parzen Estimator (TPE) to enhance hyperparameter optimization (HPO). This hybrid framework dynamically adapts search spaces and optimizes the exploration-exploitation trade-off.

Trace for Workflow Optimization Cheng et al. [2024] introduce Trace, a framework for optimizing computational workflows using LLMs. By leveraging execution traces analogous to gradients in neural networks, Trace enables generative optimization across tasks such as hyperparameter tuning and code debugging.

2.5 Frameworks for General Optimization Problems

LLMOPT for Diverse Optimization Problems Jiang et al. [2024] present LLMOPT, which formulates optimization problems across diverse fields by employing LLMs for problem definition

and solver code generation. This framework enhances generalization and accuracy in modeling optimization problem types.

2.6 Multi-agent works

Recent advancements in multi-agent systems (MAS) have addressed various challenges, including coordination, learning, and ethical considerations. A comprehensive survey by Han et al. [2024] identifies key challenges in MAS, such as optimizing task allocation, enhancing reasoning through iterative debates, managing complex contexts, and improving memory management.

Context-awareness in MAS is crucial for dynamic environments. Du et al. [2024] provide a survey on techniques for integrating context-aware systems with MAS, covering applications in autonomous driving, disaster relief, and human-AI interaction.

These studies collectively advance the understanding of MAS, addressing both theoretical and practical aspects to enhance agent collaboration, learning, and ethical integration in complex environments.

A very recent use case of multi-agent system is the Agent K v1.0 proposed by Grosnit et al. [2024] which functions as a fully autonomous agent capable of managing the entire data science lifecycle, from data preprocessing to model development, and final evaluation. It integrates libraries such as Torchvision and HuggingFace, applying advanced techniques like Bayesian optimization and memory-driven reasoning. This structured reasoning framework allows Agent K to dynamically learn from past experiences and improve its performance without retraining or fine-tuning.

2.7 Our Contribution: Multi-Agent Framework for Bayesian Optimization

Existing works, while innovative, often extend BO and LLAMBO frameworks in a task-specific or component-focused manner. However, the potential of multi-agent systems to comprehensively address BO challenges has remained unexplored. In this context, we introduce BOAgent, the first universal multi-agent framework for Bayesian Optimization. BOAgent automates BO processes—including **dynamic planning**, **prompt optimization**, and **surrogate modeling**—eliminating the reliance on manual prompt engineering and enhancing scalability. By leveraging a system agent and specialized functional agents, BOAgent achieves improved scalability, generalization, and efficiency. Our framework bridges the gap between task-specific innovations and the need for a generalized, automated BO approach, opening avenues for applications in fields such as healthcare and scientific discovery.

3 BOAgent: Methodology

In this section, we introduce **BOAgent**, a universal and efficient multi-agent framework designed to automate the Bayesian Optimization (BO) process using Large Language Models (LLMs). BOAgent addresses the limitations of previous approaches like LLAMBO by eliminating manual prompt engineering and enhancing scalability and generalization. We first formulate the BO problem and then detail the architecture and components of BOAgent, including the specific algorithms and frameworks employed.

3.1 Problem Formulation

Bayesian Optimization is a strategy for optimizing black-box functions that are expensive to evaluate and lack gradient information. Formally, consider an objective function $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X} \subseteq \mathbb{R}^d$ is a d-dimensional search space. The goal is to find the global optimum:

$$x^* = \arg\min_{x \in \mathcal{X}} f(x) \tag{1}$$

The BO process involves:

- **Surrogate Modeling**: Building a probabilistic model $\hat{f}$ (e.g., Gaussian Process) to approximate f based on observed data.

- **Acquisition Function**: Defining a utility function $a(x|\hat{f})$ that guides the selection of the next query point by balancing exploration and exploitation.
- **Candidate Sampling**: Optimizing the acquisition function to propose the next point x_{next}.

Challenges in LLAMBO include scalability to high-dimensional spaces, the necessity of expert knowledge for model selection, and computational inefficiency in surrogate modeling and acquisition function optimization.

3.2 BOAgent Framework

BOAgent introduces a multi-agent system that leverages LLMs to automate the BO process without manual prompt engineering. The framework comprises a *System Agent* and multiple *Functional Agents*, each responsible for specific components of BO.

- **System Agent**: Acts as a coordinator, generating a plan based on the problem description and delegating tasks to Functional Agents.
- **Functional Agents**: Specialized agents that execute tasks such as warm-starting, surrogate modeling, and candidate sampling. Optionally, a specialized agent can also be designed for the acquisition function.

The key innovation is the automation of task planning, prompt generation and optimization, enabling the system to generalize across different BO tasks without manual intervention.

3.3 System Agent: Planning Generation for BO Tasks

The System Agent receives a natural language problem description and generates a detailed plan outlining the steps required to perform BO. It leverages the reasoning capabilities of LLMs to interpret the problem and decide on the sequence of actions.

Algorithm 1: System Agent Planning

1. Input: Problem description D.
2. Process:
   - Parse D to identify the objective function f, constraints, and search space $\mathcal{X}$.
   - Determine the necessary components (e.g., warm-starting, surrogate modeling).
   - Generate a step-by-step plan P outlining the BO process.
3. Output: Plan P.

Example: Given the problem description "Optimize the hyperparameters of a neural network to minimize validation loss", the System Agent might generate the following plan:

1. Initialize with a set of hyperparameters (warm-starting).
2. Build a surrogate model to approximate the validation loss.
3. Optimize the acquisition function to select the next hyperparameters.
4. Evaluate the neural network with selected hyperparameters.
5. Iterate until convergence.

3.4 Functional Agents: Prompt Generation and Optimization

Each Functional Agent is responsible for executing a step in the plan by generating and optimizing prompts for the LLM to perform specific tasks.

3.4.1 Warm-Starting

The Warm-Starting Agent initializes the BO process by generating an initial set of query points.

Algorithm 2: Warm-Starting Agent

1. Input: Search space $\mathcal{X}$, number of initial points n_0.
2. Process:
   - Automatically generate a prompt to sample n_0 diverse points in $\mathcal{X}$.
   - Use the LLM to generate these points.
3. Output: Initial dataset $\mathcal{D}_0 = \{(x_i, f(x_i))\}_{i=1}^{n_0}$.

Prompt Generation: The agent formulates a prompt like:

`Generate` n_0 `diverse sets of hyperparameters within the specified ranges for each parameter.`

3.4.2 Surrogate Modeling

The Surrogate Modeling Agent builds a probabilistic model to approximate f based on observed data.

Algorithm 3: Surrogate Modeling Agent

1. Input: Dataset $\mathcal{D}_t$ at iteration t.
2. Process:
   - Generate a prompt to instruct the LLM to fit a surrogate model.
   - The LLM outputs the model parameters or directly predicts $\hat{f}(x)$.
3. Output: Surrogate model $\hat{f}$.

Prompt Generation: The agent formulates a prompt like:

`Given the data` $\mathcal{D}_t$`, fit a surrogate model to predict the objective function values for new inputs.`

3.4.3 Candidate Sampling

The Candidate Sampling Agent optimizes the acquisition function to propose the next query point.

Algorithm 4: Candidate Sampling Agent

1. Input: Surrogate model $\hat{f}$, acquisition function $a(x|\hat{f})$.
2. Process:
   - Generate a prompt to instruct the LLM to optimize $a(x|\hat{f})$ over $\mathcal{X}$.
   - The LLM proposes the next point x_{next}.
3. Output: Next query point x_{next}.

Prompt Generation: The agent formulates a prompt like:

`Using the surrogate model` $\hat{f}$`, find the input` x `that minimizes the acquisition function` $a(x|\hat{f})$.

3.5 Automation of Prompt Generation and Optimization

The crux of BOAgent is the automation of prompt generation, which eliminates the need for manual prompt engineering. This is achieved by:

- Dynamic Prompt Templates: Templates that adapt based on the task and data.
- Self-Optimization: Agents can refine their prompts based on feedback from the LLM outputs.

Algorithm 5: Prompt Optimization Loop

1. Input: Initial prompt Prompt_0, desired outcome O.
2. Process:
   - For iteration k:
     - Submit Prompt_k to the LLM.
     - Evaluate the output against O.
     - If the output is unsatisfactory, update the prompt to Prompt_{k+1} using feedback mechanisms.
3. Output: Optimized prompt Prompt^*.

Feedback Mechanism: Agents utilize the LLM's capability to critique its own outputs. For example, if the surrogate model's predictions are inaccurate, the agent might prompt:

```
Analyze the previous model's performance and suggest improvements.
```

3.6 BOAgent Algorithm Workflow

Algorithm 6: BOAgent Overall Workflow

1. Initialization:
   - Receive problem description D.
   - System Agent generates plan P.
2. Iterations ($t = 1$ to T):
   - Warm-Starting (if $t = 1$):
     - Functional Agent generates initial dataset $\mathcal{D}_0$.
   - Surrogate Modeling:
     - Functional Agent builds surrogate model $\hat{f}_t$.
   - Candidate Sampling:
     - Functional Agent proposes x_t.
   - Evaluation:
     - Compute $_t = f(x_t)$.
     - Update dataset $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(x_t, y_t)\}$.
3. Termination:
   - Stop if convergence criteria met or $t = T$.
4. Output: Optimal solution x^*.

3.7 Comparison with LLAMBO

BOAgent differs from LLAMBO in the following ways:

- Scalability: BOAgent automates prompt generation, making it scalable to higher-dimensional tasks beyond hyperparameter tuning.
- Generalization: By eliminating manual prompts, BOAgent can generalize across various domains without task-specific modifications.
- Efficiency: Automated agents reduce the time and effort required for prompt engineering, enhancing overall efficiency.

3.8 Application to General BO Tasks

BOAgent's framework is designed to be domain-agnostic, making it applicable to a wide range of optimization problems, including:

- Healthcare: Optimizing treatment plans or drug dosages.
- Engineering: Design optimization in mechanical or structural engineering.
- Finance: Portfolio optimization under uncertain market conditions.

4 Implementation

Follow the higher-level design shown in Figure 1.1, to ensure seamless integration with LLAMBO's existing components, we maintain compatibility with its core functionalities while extending them with our agent-based improvements. This integration preserves LLAMBO's proven optimization mechanisms while adding automated prompt generation and task planning capabilities.
The actual implementation follows these algorithmic specifications while handling practical considerations such as error handling, edge cases, and system robustness. For detailed implementation code, we refer readers to our public repository, where the complete codebase extending LLAMBO is available. **BOAgent Github repo**. For clarity of presentation, we outline the key implementation as pseudocode.

4.1 System Architecture

BOAgent implements a hierarchical multi-agent architecture that comprises a System Agent acting as the orchestrator and multiple Functional Agents handling specific BO components. Each agent is implemented as a modular class with well-defined interfaces:

```
class SystemAgent:
    def __init__(self, task_description):
        self.task_context = self._parse_description(task_description)
        self.optimization_state = OptimizationState()

    def generate_plan(self):
        """Generates BO workflow based on task description"""
        planning_prompt =
            self.prompt_generator.create_planning_prompt(
             self.task_context
        )
        return self._generate_and_validate_plan(planning_prompt)

    def _generate_and_validate_plan(self, prompt):
        """Ensures generated plan aligns with LLAMBO's capabilities"""
        plan = self._query_llm(prompt)
        return self._validate_against_llambo_components(plan)
```

4.2 Task Description Framework

BOAgent uses a structured JSON format for Task Descriptions that captures all necessary optimization parameters and constraints. This format allows for clear specification of the optimization problem while enabling automated processing. Here's an example of the task description in JSON format, which includes all the required information:

- Model Name: BERT
- Dataset: IMDB
- Task Type: Sentiment Analysis
- Hyperparameters: Learning rate, batch size, and other key hyperparameters with their ranges

```
{
    "task_description": {
        "model_name": "bert-base-uncased",
        "dataset": "imdb",
        "task_type": "classification",
        "metric": "f1",
        "hyperparameters": {
            "learning_rate": {
                "type": "continuous",
                "range": [1e-6, 1e-3]
            },
            "batch_size": {
                "type": "discrete",
                "range": [16, 32, 64]
            },
            "num_epochs": {
                "type": "discrete",
                "range": [1, 2, 3, 4, 5]
            }
        }
    }
}
```

The System Agent parses this description to configure the optimization pipeline and generate appropriate prompts for each component, used throughout the Bayesian Optimization process.

4.3 Functional Agent Implementation

Each functional agent implements specialized capabilities for different aspects of the BO pipeline:

4.3.1 Warm-Starting Agent

The Warm-Starting Agent leverages LLM's encoded domain knowledge for zero-shot initialization through three key strategies:

- **Context-Aware Prompting**: Constructs prompts incorporating dataset statistics, model architecture, and task requirements:

```
class WarmStartingAgent:
    def generate_points(self, task_context):
        """Generates diverse initialization points via zero-shot
           prompting."""
        metadata = self._extract_metadata(task_context)
        prompt = self._construct_prompt(
            model_type=metadata['model_name'],
            dataset_stats=metadata['dataset_stats'],
            task_type=metadata['task_type']
        )
        return self._sample_diverse_points(prompt)
```

- **Diversity Enforcement**: Implements correlation-based diversity metrics to ensure search space coverage
- **Constraint Satisfaction**: Validates generated points against search space constraints

4.3.2 Surrogate Modeling Agent

We implement two complementary approaches for surrogate modeling through in-context learning:

Discriminative Approach:

```python
class DiscriminativeSurrogate:
    def predict(self, x, observed_data):
        """Predicts mean and uncertainty through ensemble."""
        predictions = []
        for _ in range(self.n_ensemble):
            shuffled_data =
                self._shuffle_demonstrations(observed_data)
            prompt = self._construct_prompt(x, shuffled_data)
            pred = self._query_llm(prompt)
            predictions.append(self._parse_prediction(pred))
        return self._compute_statistics(predictions)
```

Generative Approach:

```python
class GenerativeSurrogate:
    def predict_probability(self, x, observed_data):
        """Predicts probability of exceeding performance threshold."""
        threshold = self._compute_threshold(observed_data)
        good_examples, bad_examples = self._partition_examples(
            observed_data, threshold
        )
        prompt = self._construct_binary_prompt(
            x, good_examples, bad_examples, threshold
        )
        return self._query_llm(prompt)
```

4.3.3 Candidate Sampling Agent

The Candidate Sampling Agent implements a novel conditional sampling approach with several key mechanisms:

Generative Approach:

```python
class CandidateSamplingAgent:
    def sample_candidates(self, observed_data):
        """Samples candidates using target-conditioned generation."""
        s_min = min(observed_data['scores'])
        s_max = max(observed_data['scores'])
        target = s_min - self.alpha * (s_max - s_min)

        candidates = []
        while len(candidates) < self.n_candidates:
            prompt = self._construct_conditional_prompt(
                target_score=target,
                historical_data=observed_data,
                div_penalty=self._compute_div_penalty(candidates)
            )
            candidate = self._generate_candidate(prompt)
            if self._is_valid_and_diverse(candidate, candidates):
                candidates.append(candidate)
        return candidates
```

- **Diversity-Aware Generation**: Enforces diversity through explicit penalties and rejection sampling
- **Validity Enforcement**: Implements comprehensive validation of generated candidates

4.3.4 Evaluation Agent

The Evaluation Agent integrates with machine learning frameworks while handling practical considerations:

```python
class EvaluationAgent:
    def evaluate(self, config, task_context):
        """Evaluates configuration while handling practical issues."""
        model = self._initialize_model(task_context['model_name'],
            config)

        try:
            with TimeoutManager(task_context['timeout']):
                results = self._train_and_evaluate(
                    model,
                    task_context['dataset'],
                    task_context['metric']
                )
            return self._process_results(results, config,
                task_context)
        except Exception as e:
            return self._handle_evaluation_failure(e, config)
```

4.4 LLM Integration

BOAgent primarily uses GPT-3.5 through the Azure OpenAI service, while maintaining compatibility with other LLMs including Llama, Mistral for local deployment:

```python
class LLMInterface:
    def __init__(self, model_config):
        self.model = model_config.get('model', "gpt-3.5-turbo-0301")
        self.temperature = model_config.get('temperature', 0.7)
        self.top_p = model_config.get('top_p', 0.95)

    def query(self, prompt, max_tokens=500):
        """Queries LLM with specified prompt."""
        if self.model.startswith('gpt'):
            return self._query_openai(prompt, max_tokens)
        else:
            return self._query_local_llm(prompt, max_tokens)
```

This implementation provides a robust and flexible framework that extends LLAMBO's capabilities while maintaining its core strengths. The modular design allows for easy extension and modification of individual components, enabling adaptation to diverse optimization scenarios.
Alternative Implementation with localized Llama 2 models and agent-based communication refactoring (with CAMEL AI) approaches are detailed in Appendix A, though these implementations faced challenges with reasoning capabilities and computational efficiency compared to the current implementation.

5 Experiment

5.1 Experimental Protocol

To ensure rigorous evaluation and reproducibility of our results, we established the following experimental protocol:

- Each task was executed independently 5 times, with results averaged to account for statistical variations
- Random seeds were fixed across all experiments for each task to ensure reproducibility
- Maximum number of iterations was set to 100 for each optimization run
- Time limit was enforced at 12 hours per task to maintain practical constraints

5.2 BOAgent Configuration

The configuration of BOAgent was carefully tuned to balance performance and computational efficiency:

5.2.1 System Agent Configuration

The System Agent was configured with the following parameters:

- Base Model: GPT-3.5-turbo
- Temperature: 0.7 (to balance exploration and exploitation)
- Top-p sampling: 0.95 (to maintain output diversity)
- Maximum tokens: 500 per response

5.2.2 Functional Agents Configuration

Each Functional Agent was configured with specific parameters optimized for their respective tasks:

- Warm-Starting Agent: Initial sampling size $n_0 = 5$
- Surrogate Modeling Agent: Ensemble of 10 models for robust prediction
- Candidate Sampling Agent: Generation of 20 candidates per iteration

5.3 Results

This configuration was maintained consistently across all experiments to ensure fair comparison and reproducibility of results.

Task Type	Description	AutoGluon	BOAgent
Binary Classification	kidney-stone-prediction (playground-series-s3e12)	0.8725 (roc_auc)	0.876 (roc_auc)
	hotel-booking-demand-3	0.9992 (acc)	0.936(acc)
	customer-churn-prediction-2020	0.966 (acc)	0.978 (acc)
	676-cases-of-pharyngitis-in-children (pharyngitis)	0.7647 (acc)	0.7065
Clustering	flight-customer-segmentation	AutoGluon does not support Clustering at the moment	due to computation limits, unable to show clustering results
	mall-customer-segmentation		The clusters contain different numbers of members: - Cluster 1: 58 customers - Cluster 3: 47 customers - Cluster 4: 39 customers - Cluster 2: 34 customers - Cluster 0: 22 customers Full result please see Appendix A GitHub Result Folder
	in-class-competition-data-clustering (Iris data clustering)		Full result please see Appendix C
Regression	house-prices-advanced-regression-techniques	0.1264 (rmsle)	0.123 (RMSLE)
	crab-age-dataset (playground-series-s3e16)	1.4328 (ame)	1.4124 (MAE)
	california-housing-price (playground-series-s3e1)		0.6082 (RMSE)
	mohs-hardness (playground-series-s3e25)	0.5177(MedAE)	0.6634 (MedAE)
	gemstone-price (playground-series-s3e25)	515.7227 (rmse)	582.9265(RMSE)
	mercedes-benz-greener-manufacturing (playground-series-s3e16)	0.3099 (R^2)	0.68 (R^2) Coefficient of determination

Figure 5.1: Performance Comparison between BOAgent and AutoGluon across Different ML Hyperparameter Tuning Tasks

The effectiveness of the multi-agent strategy employed in BOAgent was evaluated by comparing its performance against AutoGluon [17], a state-of-the-art single-agent AutoML system. Figure 5.1 presents the comparison between AutoGluon and BOAgent.

BOAgent outperformed AutoGluon in 6 out of the 9 tasks, demonstrating the benefits of the multi-agent approach. The improved performance can be attributed to the specialised roles assigned to each agent, allowing for optimised prompting and improved task understanding by the LLMs. On average, BOAgent achieved a 7.8% improvement over AutoGluon across the evaluated tasks. Additionally, BOAgent successfully handled clustering tasks, despite being less commonly benchmarked in agent frameworks, and AutoGluon does not even support clustering task.

5.4 Interpretation of Key Findings

The BOAgent project has demonstrated the potential of LLMs (LLMs) in automating and optimising BO for data science workflows. One of the key insights gained from this research is the concept of '*Agent as API*', where AI agents are provided to users in the form of an API, similar to the Assistant API developed by OpenAI. This approach streamlines the integration of AI agents into existing systems and workflows, making them more accessible and user-friendly.

Through extensive experimentation and the utilisation of a significant number of tokens, the BOAgent system has shown promising results in various aspects of the BO process. For instance, the system has demonstrated the ability to quickly identify traditional model baselines, accelerate fundamental data preparation and feature engineering, and enhance task understanding and data variable exploration. These improvements, although incremental, can lead to significant time and resource savings in the long run.

6 Conclusion and Future Work

This work introduced **BOAgent**, a multi-agent framework designed to address limitations in scalability and generalization present in LLAMBO, an earlier framework that uses large language models (LLMs) to enhance Bayesian Optimization (BO). BOAgent automates the entire BO process, including prompt generation and optimization, eliminating the need for manual prompt engineering. The framework comprises a system agent that plans BO workflows and functional agents that execute key components, such as surrogate modeling and candidate sampling.

The key contributions of this work include: (1) the development of a scalable architecture that eliminates manual prompt engineering through automated prompt generation and optimization, (2) the successful implementation of specialized agents that handle crucial BO components like warm-starting, surrogate modeling, and candidate sampling, and (3) the demonstration that a multi-agent approach can effectively generalize beyond traditional hyperparameter tuning tasks to broader optimization challenges.

Several promising directions emerge for future research:

- **Handling Uncertainty** Dealing with uncertainty is a critical aspect of developing AI systems for real-world applications. The BOAgent project encountered challenges related to the heterogeneous and uncertain nature of LLM outputs, which can impact the trustworthiness and reliability of the generated results. Future research should focus on developing techniques to quantify and mitigate uncertainty in AI models, such as uncertainty estimation, model ensembling, and human-in-the-loop approaches.
- **Enhanced Agent Collaboration**: Investigating more sophisticated inter-agent communication protocols and decision-making mechanisms could further improve the system's efficiency and effectiveness. This could include developing methods for agents to share learned strategies and adaptation mechanisms across different optimization tasks.
- **Scalability Improvements**: While BOAgent demonstrates improved scalability compared to existing approaches, further research into handling very high-dimensional optimization problems and large-scale parallel optimization scenarios could expand its practical applications.
- **Domain-Specific Adaptations**: Exploring specialized variations of BOAgent for specific domains like healthcare, engineering, or financial optimization could lead to improved performance through domain-aware prompt generation and optimization strategies.
- **Integration with Advanced LLMs**: As new language models emerge, investigating how BOAgent can leverage their capabilities while maintaining computational efficiency could enhance the framework's capabilities and robustness.
- **Theoretical Analysis**: Developing formal theoretical frameworks for understanding the convergence properties and optimization guarantees of LLM-driven multi-agent BO systems would provide valuable insights for future improvements.

These advances in automated, generalized Bayesian optimization open new possibilities for applying BO to increasingly complex real-world problems while reducing the expertise required for

implementation. The framework's success in automating prompt engineering while maintaining optimization performance suggests that multi-agent approaches may be valuable for other optimization and machine learning tasks that currently rely on manual prompt engineering.

References

Virginia Aglietti, Ira Ktena, Jessica Schrouff, Eleni Sgouritsa, Francisco J. R. Ruiz, Alan Malek, Alexis Bellot, and Silvia Chiappa. FunBO: Discovering acquisition functions forbayesian optimization with funsearch. In *Submitted to The Thirteenth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=OSmjkkF6Uy. under review.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela van der Schaar. Autoformulation of mathematical optimization models using llms, 2024. URL https://arxiv.org/abs/2411.01679.

Guojin Chen, Keren Zhu, Seunggeun Kim, Hanqing Zhu, Yao Lai, Bei Yu, and David Z. Pan. Llm-enhanced bayesian optimization for efficient analog layout constraint generation, 2024. URL https://arxiv.org/abs/2406.05250.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=rYs2Dmn9tD.

Hung Du, Srikanth Thudumu, Rajesh Vasa, and Kon Mouzakis. A survey on context-aware multi-agent systems: Techniques, challenges and future directions, 2024. URL https://arxiv.org/abs/2402.01968.

Patrick Emami, Zhaonan Li, Saumya Sinha, and Truc Nguyen. Syscaps: Language interfaces for simulation surrogates of complex systems, 2024. URL https://arxiv.org/abs/2405.19653.

Antoine Grosnit, Alexandre Maraval, James Doran, Giuseppe Paolo, Albert Thomas, Refinath Shahul Hameed Nabeezath Beevi, Jonas Gonzalez, Khyati Khandelwal, Ignacio Iacobacci, Abdelhakim Benechehab, Hamza Cherkaoui, Youssef Attia El-Hili, Kun Shao, Jianye Hao, Jun Yao, Balazs Kegl, Haitham Bou-Ammar, and Jun Wang. Large language models orchestrating structured reasoning achieve kaggle grandmaster level, 2024. URL https://arxiv.org/abs/2411.03562.

Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, Zhaozhuo Xu, and Chaoyang He. Llm multi-agent systems: Challenges and open problems, 2024. URL https://arxiv.org/abs/2402.03578.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. Llmopt: Learning to define and solve general optimization problems from scratch, 2024. URL https://arxiv.org/abs/2410.13213.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=OOxotBmGol.

Kanan Mahammadli. Sequential large language model-based hyper-parameter optimization, 2024. URL https://arxiv.org/abs/2410.20302.

James Requeima, John F Bronskill, Dami Choi, Richard E. Turner, and David Duvenaud. LLM processes: Numerical predictive distributions conditioned on natural language. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=HShs7q1Njh.

Wenjie Xu, Masaki Adachi, Colin Jones, and Michael A Osborne. Principled bayesian optimization in collaboration with human experts. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=IDn9SiKgLy.

Yiming Yao, Fei Liu, Ji Cheng, and Qingfu Zhang. Evolve cost-aware acquisition functions using large language models, 2024. URL https://arxiv.org/abs/2404.16906.

A Alternative Implementation

A.1 Implementation 2: LLAMBO with Localized Llama2 Model

To explore the application potential of the localized Llama2 model in the BO algorithm and evaluate its performance within the LLAMBO framework, we combined the localized Llama2 model with the BO algorithm and integrated it into the LLAMBO framework. During the experiment, the **same dataset and evaluation metrics** as in the previous experiments were used to ensure the comparability of the results.

However, the experimental results demonstrated that due to the inferior reasoning ability of the localized Llama2 model compared to OpenAI, it encountered difficulties in finding better solutions. Specifically, during the execution process, the model terminated the program prematurely as it could not identify superior solutions. This led to unsatisfactory final optimization results that failed to meet the expected performance indicators.

The insufficient reasoning ability of the localized Llama2 model might be attributed to factors such as its model scale, training data, or algorithm optimization. This result suggests that when selecting a model, its performance and applicability need to be thoroughly considered to ensure effective problem-solving. Corresponding code at: `https://github.com/TinaXYH/BOAgent`.

A.2 Implementation 3: Constructing Agents in LLAMBO with CAMEL AI

To construct more flexible and efficient agents by **refactoring the communication function** in Camel AI framework and thereby enhance the performance of the LLAMBO method, we refactored the communication function in Camel AI and configured warmstarting, candidate sampling, and surrogate modeling as **independent roles**. These roles interacted and collaborated to jointly complete the entire LLAMBO operation. During the experiment, the refactored system was tested and evaluated.

Although this approach had certain theoretical advantages, in the actual experiment, it was found that the refactoring process consumed a significant amount of time. Due to time constraints, it was challenging to identify better solutions within the limited time frame. This might be because the refactored system was more complex and required **more computing and financial resources** and time for optimization and adjustment.

Refactoring the communication function in Camel AI was a challenging task that demanded a profound understanding of the system's architecture and operating principles. Additionally, to improve the system's performance, further optimization of the refactored algorithm and model was necessary. In future research, more efficient algorithms and techniques could be considered to reduce computational time and improve solution quality.