Mid-Term Progress Report

Optimizing the Utilization of Large Language Models via Schedule Optimization

---

Course: MCA 101
Program: Master of Computer Applications (Final Semester)
Academic Year: 2025-26
Institution: National Institute of Technology, Warangal

Student Name: [Your Name]
Roll Number: [Your Roll Number]
Branch: Computer Science and Engineering

Project Guide: [Guide Name]
Submission Date: January 2026

Abstract

Large Language Models have demonstrated remarkable capabilities in various natural language processing tasks, including log analysis and anomaly detection. However, their deployment in production systems faces a critical challenge: the trade-off between prediction accuracy and computational cost. Different LLM prompting strategies (ranging from simple instructions to few-shot examples) yield varying levels of accuracy at different token costs. This project explores an approach to optimize LLM utilization through intelligent schedule optimization, where different prompting strategies are dynamically assigned to different tasks based on predicted success rates. We implement a machine learning-based predictor combined with multi-objective optimization algorithms to find optimal cost-accuracy trade-offs. The mid-term implementation includes a basic prediction framework and preliminary optimization algorithms, with initial experiments showing potential for cost reduction while maintaining acceptable accuracy levels.

Chapter 1: Introduction

1.1 Background

The widespread adoption of Large Language Models in enterprise applications has brought both opportunities and challenges. While these models offer unprecedented capabilities in understanding and processing natural language, their operational costs can be substantial, particularly in high-volume scenarios such as log analysis systems that process thousands of entries per day.

Modern systems generate vast amounts of log data that require continuous monitoring and analysis. Traditional rule-based approaches struggle with the variety and complexity of log patterns, while manual inspection is impractical at scale. LLMs provide an attractive solution by offering flexible, context-aware analysis capabilities. However, deploying powerful models for every task regardless of complexity leads to unnecessary expenditure.

1.2 Problem Statement

The central problem addressed in this project is: **How can we minimize the cost of utilizing Large Language Models for classification tasks while maintaining high accuracy?**

This involves several sub-problems:

- Different prompting strategies (simple vs. detailed instructions) have different costs and accuracy rates
- Not all tasks require the most expensive prompting strategy
- Predicting which strategy will succeed for a given task is non-trivial
- There exists a multi-objective optimization problem: maximizing accuracy while minimizing cost

1.3 Motivation

Consider a system processing log entries for anomaly detection. Using the most detailed prompting strategy for all entries might achieve high accuracy but at significant cost. Conversely, using only simple prompts reduces costs but may miss critical anomalies. An intelligent system that assigns appropriate prompting strategies to different tasks based on their characteristics could achieve near-optimal accuracy at a fraction of the cost.

This research is motivated by:

- **Economic factors**: Reducing operational costs in production deployments
- **Efficiency**: Optimal resource utilization in large-scale systems
- **Scalability**: Making LLM-based solutions viable for cost-sensitive applications
- **Scientific interest**: Exploring search-based software engineering approaches for LLM optimization

1.4 Objectives

The primary objectives of this project are:

1. To study and understand the cost-accuracy trade-offs in LLM prompting strategies
2. To develop a machine learning-based predictor that estimates the success probability of different prompting strategies for given tasks
3. To implement multi-objective optimization algorithms that find optimal assignments of prompting strategies to tasks
4. To evaluate the approach on real-world datasets and compare against baseline strategies
5. To demonstrate measurable cost savings while maintaining acceptable accuracy levels

1.5 Report Organization

This mid-term report is organized as follows: Chapter 2 reviews relevant literature and background concepts. Chapter 3 summarizes the reference research paper. Chapter 4 describes our methodology. Chapter 5 presents implementation details. Chapter 6 discusses experimental setup. Chapter 7 shows preliminary results. Chapter 8 acknowledges current challenges, and Chapter 9 outlines future work.

Chapter 2: Literature Review

2.1 Large Language Models in Log Analysis

Log analysis and anomaly detection have traditionally relied on pattern matching, statistical methods, and rule-based systems. Recent research has demonstrated that Large Language Models can effectively understand log semantics and detect anomalies without extensive feature engineering. However, the computational cost of LLM inference remains a practical concern for deployment.

2.2 Prompt Engineering

Prompt engineering refers to the practice of designing input instructions to guide LLM behavior. Research has shown that prompt complexity significantly affects both accuracy and cost:

- **Simple prompts**: Direct instructions with minimal context, lower token count, faster inference
- **Standard prompts**: More detailed instructions with task clarification
- **Few-shot prompts**: Include example inputs and outputs to guide the model

The choice of prompting strategy impacts both the quality of results and the operational cost, creating a fundamental trade-off.

2.3 Multi-Objective Optimization

Multi-objective optimization addresses problems where multiple conflicting objectives must be simultaneously optimized. In our context, we aim to:

- **Maximize**: Overall classification accuracy
- **Minimize**: Total computational cost (token usage)

Classical algorithms for multi-objective optimization include:

- **NSGA-II (Non-dominated Sorting Genetic Algorithm)**: Uses evolutionary computation with non-dominated sorting to maintain population diversity
- **SPEA2 (Strength Pareto Evolutionary Algorithm)**: Employs an archive of non-dominated solutions with fitness assignment based on Pareto dominance
- **Random Search**: Serves as a baseline by randomly generating solution candidates

These algorithms produce a Pareto front—a set of non-dominated solutions representing different trade-offs between objectives.

2.4 Machine Learning for Prediction

Supervised learning techniques can predict task outcomes based on input features. In our case, we train a classifier to predict whether a given prompting strategy will successfully classify a particular task. Features extracted from tasks (such as complexity indicators, length, specific keywords) serve as inputs, while historical success/failure data provides training labels.

2.5 Search-Based Software Engineering

Search-Based Software Engineering (SBSE) applies metaheuristic search techniques to software engineering problems. Our work falls within this paradigm by treating prompting strategy selection as a search problem in a large solution space, where we seek optimal configurations according to defined objectives.

Chapter 3: Reference Paper Summary

3.1 Core Concept

The reference paper proposes a framework for optimizing LLM utilization through a two-phase approach:

Phase 1 - Prediction: Train a machine learning model to predict the probability that a specific LLM configuration (model choice or prompting strategy) will successfully complete a given task.

Phase 2 - Optimization: Use multi-objective optimization algorithms to find optimal assignments of LLM configurations to tasks, balancing accuracy and cost.

3.2 ML-Based Predictor (MLBP)

The ML-Based Predictor works as follows:

1. **Training Data Collection**: Sample a subset of tasks and obtain ground truth by querying LLMs with various prompting strategies
2. **Feature Extraction**: Extract relevant features from tasks and prompting strategies
3. **Model Training**: Train a classifier (Random Forest or XGBoost) to predict success probability
4. **Prediction**: For new tasks, predict success probabilities for all available prompting strategies

The predictor outputs a probability matrix P where P[i][j] represents the probability that prompting strategy j will successfully classify task i.

3.3 Schedule Optimization

Given the probability matrix and cost information, the optimization phase seeks to assign one prompting strategy to each task such that:

- The sum of success probabilities is maximized (high accuracy)
- The sum of costs is minimized (low expenditure)

This is formulated as a multi-objective optimization problem solved using evolutionary algorithms.

3.4 Evaluation Metrics

The paper uses standard multi-objective optimization metrics:

- **IGD (Inverted Generational Distance)**: Measures how close the obtained Pareto front is to the true reference Pareto front
- **Δ (Delta/Spread)**: Measures the diversity and spread of solutions along the Pareto front
- **M_n (Number of solutions)**: Counts non-dominated solutions in the Pareto front

---

Chapter 4: Methodology

4.1 System Architecture

Our implementation follows a pipeline architecture with distinct phases:

```
[Data Loading] → [Prediction Phase] → [Optimization Phase] → [Evaluation]
      ↓                   ↓                       ↓                    ↓
```

```
    Raw logs         Probability matrix    Pareto front       Metrics & plots
```

4.2 Data Preprocessing

We work with structured log datasets where each entry contains:

- Timestamp and severity level
- Component identifier
- Message text
- Ground truth label (normal/anomaly)

Feature extraction includes:

- Token count (log length)
- Unique token count (vocabulary richness)
- Presence of error keywords
- Prompt complexity level

4.3 Prompting Strategies

We define multiple prompting strategies with varying complexity:

1. **Simple**: Minimal instruction, lowest token cost
2. **Standard**: Moderate detail with task context
3. **Few-shot (1 example)**: Includes one example
4. **Few-shot (3 examples)**: Includes multiple examples for better guidance

Each strategy has an associated token cost proportional to its complexity.

4.4 Machine Learning Predictor

We employ XGBoost (Extreme Gradient Boosting) as our prediction model due to its:

- Strong performance on structured data
- Ability to capture non-linear relationships
- Reasonable training time
- Built-in feature importance

The model takes task features and prompting strategy features as input and outputs a probability of success.

4.5 Optimization Algorithms

We implement three multi-objective optimization algorithms:

Random Search: Generates random assignments as a baseline for comparison.

NSGA-II: Uses genetic algorithm principles with:

- Non-dominated sorting for fitness evaluation
- Crowding distance to maintain diversity
- Tournament selection and crossover/mutation operators

SPEA2: Maintains an external archive of best solutions with:

- Fitness assignment based on domination count
- Density estimation for diversity preservation

4.6 Evaluation Framework

We evaluate using:

- **Baseline strategies**: Assigning the same prompting strategy to all tasks
- **Multi-objective metrics**: IGD, Delta, and solution count
- **Visual comparison**: Pareto front plots showing cost-accuracy trade-offs

---

Chapter 5: Implementation Details

5.1 Technology Stack

- **Programming Language**: Python 3.14
- **ML Library**: scikit-learn, XGBoost
- **Optimization**: Custom implementations of NSGA-II and SPEA2
- **LLM Interface**: Ollama (local deployment) with Gemma3 model
- **Visualization**: Matplotlib, NumPy
- **Data Handling**: Pandas

5.2 System Components

Data Loader (`jobs/loader.py`): Parses log files, extracts features, associates ground truth labels.

Prompt Templates (`prompts/templates.py`): Defines multiple prompting strategies with associated costs.

LLM Client (`llm/gemini_client.py`): Interfaces with local LLM deployment, implements response caching for efficiency.

Predictor (`predictors/mlbp.py`): Implements training data creation, model training, and probability prediction.

Optimizers (`optimizers/`): Contains implementations of Random Search, NSGA-II, and SPEA2.

Evaluation (`evaluation/`): Baseline computation, Pareto front extraction, metric calculation.

Visualization (`visualization/plots.py`): Generates plots comparing optimization algorithms and baselines.

5.3 Caching Mechanism

To reduce repeated LLM API calls during experimentation, we implement a file-based cache:

- Each (task, prompting strategy) pair has a unique hash
- LLM responses are stored as JSON files
- Subsequent requests check cache before making API calls
- This enables rapid iteration during development

5.4 Current Implementation Status

Completed:

- Dataset loading and preprocessing
- Feature extraction pipeline
- Multiple prompting strategy definitions
- XGBoost predictor training
- Basic implementations of three optimization algorithms
- Baseline evaluation framework
- Probability matrix generation and storage
- Visualization of Pareto fronts

In Progress:

- Complete LLM response caching (partial cache exists)
- Model accuracy improvement through hyperparameter tuning

Not Yet Implemented:

- Multiple LLM model deployment
- Additional dataset testing
- Real-time deployment interface
- Comprehensive ablation studies

Chapter 6: Experimental Setup

6.1 Dataset

We conduct preliminary experiments using a publicly available log dataset containing several thousand log entries with binary labels (normal/anomaly). The dataset is split into training and evaluation sets.

6.2 Experimental Configuration

- **Number of tasks**: 5,000 log entries
- **Prompting strategies**: 4 variants with costs ranging from 10 to 90 tokens
- **Optimization runs**: 3 independent runs per algorithm for statistical validity
- **Population size**: 100 (for evolutionary algorithms)
- **Generations**: 50 (for evolutionary algorithms)
- **Random search iterations**: 1,000

6.3 Baseline Strategies

We compare against four single-strategy baselines:

1. Use simplest prompting for all tasks (lowest cost, potentially lower accuracy)
2. Use standard prompting for all tasks
3. Use moderate few-shot prompting for all tasks
4. Use most detailed prompting for all tasks (highest cost, potentially highest accuracy)

Chapter 7: Preliminary Results

7.1 Predictor Performance

Our XGBoost-based predictor achieves approximately 79% accuracy in predicting whether a prompting strategy will successfully classify a given task. While this leaves room for improvement, it demonstrates that task features contain useful signals for prediction.

7.2 Baseline Results

Single-strategy baselines show expected patterns:

- Simpler prompting: Lower cost, lower accuracy
- More complex prompting: Higher cost, higher accuracy
- Clear monotonic relationship between prompt complexity and performance

7.3 Optimization Results

Preliminary runs of optimization algorithms show:

- **NSGA-II** produces the best Pareto fronts with IGD scores approaching zero (indicating closeness to reference front)
- **SPEA2** generates fewer solutions but with reasonable quality
- **Random Search** serves as an effective baseline, though outperformed by evolutionary algorithms

Visual inspection of Pareto fronts indicates that optimized solutions offer better cost-accuracy trade-offs compared to single-strategy baselines.

7.4 Cost Savings Potential

Early results suggest that intelligent assignment can achieve accuracy levels close to the most expensive baseline while significantly reducing overall token usage, demonstrating the viability of the approach.

Chapter 8: Challenges and Limitations

8.1 Current Challenges

LLM Response Caching: Building a comprehensive cache of LLM responses for all task-strategy pairs is time-intensive, requiring substantial API calls to the local model.

Model Accuracy: The current predictor accuracy of 79% indicates room for improvement through better feature engineering or model selection.

Computational Cost: Running multiple optimization algorithms with multiple independent runs requires considerable computation time.

8.2 Limitations

Dataset Scope: Current experiments use a single dataset type; generalization to other domains requires validation.

Prompting Strategy Design: Our prompting strategies are manually crafted; automated prompt generation could be explored.

Static Assignment: The system produces static schedules; dynamic re-optimization based on runtime feedback is not yet implemented.

Real-World Deployment: The system has not been tested in production environments with real-time constraints.

Chapter 9: Future Work

9.1 Immediate Next Steps

1. **Complete Cache Generation**: Finish collecting LLM responses for all task-strategy combinations to enable 100% grounding in real data rather than synthetic heuristics

2. **Improve Predictor Accuracy**: Conduct hyperparameter tuning, feature engineering, and possibly try neural network-based predictors

3. **Extended Evaluation**: Test on additional datasets from different domains to validate generalization

9.2 Medium-Term Goals

4. **Multiple LLM Models**: Extend the framework to support choosing among different LLM models (not just prompting strategies), representing a more realistic cost-accuracy spectrum

5. **Online Learning**: Implement mechanisms to update the predictor based on runtime feedback, adapting to changing patterns

6. **Visualization Dashboard**: Develop an interactive interface for exploring Pareto fronts and selecting deployment configurations

9.3 Long-Term Vision

7. **Production Deployment**: Deploy the system in a real log analysis pipeline and measure actual cost savings

8. **Automated Prompt Engineering**: Integrate techniques to automatically generate and optimize prompting strategies

9. **Transfer Learning**: Investigate whether predictors trained on one task type can transfer to others with minimal retraining

---

Chapter 10: Conclusion

This mid-term report documents our progress in implementing a framework for optimizing Large Language Model utilization through schedule optimization. We have successfully established the foundational components: data preprocessing pipelines, multiple prompting strategies, a machine learning-based predictor, and implementations of multi-objective optimization algorithms.

Preliminary results demonstrate the feasibility of the approach, with optimization algorithms producing Pareto fronts that offer superior cost-accuracy trade-offs compared to naive single-strategy baselines. The ML predictor, while showing room for improvement, successfully captures patterns that distinguish tasks requiring different prompting strategies.

Significant work remains to complete the implementation, particularly in building comprehensive training data, improving prediction accuracy, and conducting extensive evaluations across diverse datasets. However, the current progress provides a solid foundation for the remaining development phases.

The project aligns with current research directions in search-based software engineering and addresses a practically important problem as LLM deployment costs become increasingly significant in production systems. Upon completion, this work aims to demonstrate that intelligent, data-driven scheduling can substantially reduce operational costs while maintaining quality of service.

References

1. **Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael Jordan**. "Detecting Large-Scale System Problems by Mining Console Logs", Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP), 2009.

2. **Tianlin Li, et al**. "Optimizing the Utilization of Large Language Models via Schedule Optimization", Research Paper (Reference Document).

3. **Kalyanmoy Deb, et al**. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation, 2002.

4. **Eckart Zitzler, Marco Laumanns, Lothar Thiele**. "SPEA2: Improving the Strength Pareto Evolutionary Algorithm", Technical Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 2001.

5. **Tianqi Chen, Carlos Guestrin**. "XGBoost: A Scalable Tree Boosting System", Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

6. **Mark Harman, S. Afshin Mansouri, Yuanyuan Zhang**. "Search-based Software Engineering: Trends, Techniques and Applications", ACM Computing Surveys, 2012.

7. **Tom Brown, et al**. "Language Models are Few-Shot Learners", Advances in Neural Information Processing Systems, 2020.

8. **Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, Michael R. Lyu**. "Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics", IEEE International Symposium on Software Reliability Engineering (ISSRE), 2023.

Appendix A: System Architecture Diagram

```
┌─────────────────────────────────────────────────────────────────┐
│                    INPUT: Log Dataset                           │
│              (Logs + Ground Truth Labels)                       │
```

```
└──────────────────────────┬──────────────────────────┘
                           │
                           ▼
┌─────────────────────────────────────────────────────┐
│              PHASE 1: PREDICTION                     │
│                                                      │
│  ┌─────────────────┐      ┌──────────────────┐      │
│  │ Feature         │      │ Prompting        │      │
│  │ Extraction      │──────│ Strategies       │      │
│  └─────────────────┘      └──────────────────┘      │
│           │                         │               │
│           └────────────┬────────────┘               │
│                        ▼                             │
│         ┌────────────────────┐                      │
│         │  Training Data     │                      │
│         │  Creation          │                      │
│         │  (LLM Calls)       │                      │
│         └────────┬───────────┘                      │
│                  ▼                                   │
│         ┌────────────────────┐                      │
│         │  XGBoost Training  │                      │
│         └────────┬───────────┘                      │
│                  ▼                                   │
│         ┌────────────────────┐                      │
│         │ Probability Matrix │                      │
│         │   P[task][prompt]  │                      │
│         └────────────────────┘                      │
└──────────────────────────┬──────────────────────────┘
                           │
                           ▼
┌─────────────────────────────────────────────────────┐
│              PHASE 2: OPTIMIZATION                   │
│                                                      │
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐ │
│  │ Random      │  │   NSGA-II   │  │    SPEA2    │ │
│  │ Search      │  │             │  │             │ │
│  └─────────────┘  └──────┬──────┘  └──────┬──────┘ │
│         │                 │                 │       │
│         └─────────────────┼─────────────────┘       │
│                           ▼                          │
│                ┌──────────────────┐                 │
│                │  Pareto Front    │                 │
│                │  Extraction      │                 │
│                └────────┬─────────┘                 │
└─────────────────────────┼───────────────────────────┘
                          │
                          ▼
┌─────────────────────────────────────────────────────┐
│                 OUTPUT: Results                      │
│                                                      │
│  • Optimal Schedules (Task → Prompt assignments)    │
│  • Pareto Fronts (Cost vs Accuracy trade-offs)      │
│  • Metrics (IGD, Delta, Mₙ)                        │
│  • Visualizations                                    │
└──────────────────────────────────────────────────────┘
```

Appendix B: Sample Output Statistics

Dataset Statistics:

- Total log entries: 5,000
- Normal logs: ~96%
- Anomaly logs: ~4%

Prompting Strategy Costs:

- Simple: 10 tokens
- Standard: 25 tokens
- Few-shot (1 example): 50 tokens
- Few-shot (3 examples): 90 tokens

Model Configuration:

- Algorithm: XGBoost
- Estimators: 400
- Max Depth: 8
- Learning Rate: 0.05

Preliminary Metrics (Average over 3 runs):

- NSGA-II IGD: ~0.000 (excellent)
- SPEA2 IGD: ~0.231
- Random Search IGD: ~0.097

End of Mid-Term Progress Report