Smart Prompt Recommender for LLM Workloads

Optimizing Quality, Cost, and Safety with Data-Driven Prompt Selection

Team: Brocode

Course: CMPE-256 Sec 49 - Recommender Systems

Professor: Chandrasekar Vuppalapati

University: San Jose State University

Date: 04 December 2025

- **Project Overview:** An intelligent system that automatically selects optimal prompts for LLM tasks
- **Why Prompts Matter:** Prompt quality directly impacts output accuracy, token consumption, and safety compliance
- **System Capability:** Leverages historical data and machine learning to recommend the best prompt for each specific task

Motivation, Problem Statement & Objectives

The Challenge of Manual Prompt Engineering

Current approaches to prompt engineering rely heavily on manual trial-and-error, creating significant operational challenges. Engineers spend hours testing different prompt variations, leading to **high token costs** from repeated API calls. Each iteration introduces **latency delays** that slow down development cycles. Without systematic evaluation, teams face **inconsistent quality** across different use cases. Perhaps most critically, ad-hoc prompting increases **safety risks** as harmful outputs may go undetected until production deployment.

Problem Statement

> Organizations need an **automated, safety-aware, cost-optimized system** for selecting the most appropriate prompt template for each LLM task, eliminating manual experimentation while maintaining quality standards.

Improve Quality

Maximize output accuracy and task completion rates through data-driven prompt selection

Reduce Cost per Task

Lower token consumption and API expenses while maintaining successful outcomes

Enforce Safety

Apply safety constraints proactively to prevent harmful or biased outputs

Enable Interpretability

Provide transparent metrics explaining why specific prompts are recommended

Requirements & KDD Pipeline Overview

System Requirements

Functional Requirements

- **Data Ingestion:** Capture historical prompt-response logs with quality metrics
- **Prompt Library:** Maintain versioned repository of validated templates
- **Retrieval & Ranking APIs:** Fast candidate selection and intelligent ordering
- **Off-Policy Evaluation (OPE):** Estimate new policy performance from logged data
- **Online Experiments:** A/B testing framework for prompt strategies
- **User Interface:** Task runner, prompt explorer, and analytics dashboard
- **Monitoring:** Real-time tracking of quality, cost, and safety metrics

Non-Functional Requirements

- **Latency:** Sub-200ms recommendation response time
- **Scalability:** Handle 10,000+ requests per second
- **Reliability:** 99.9% uptime with graceful degradation
- **Security:** End-to-end encryption and role-based access control

Knowledge Discovery Pipeline

Data Sources

Historical interaction logs containing task descriptions, prompt templates, model configurations, input/output pairs, quality scores, safety violation flags, and precise token consumption counts across multiple application domains.

Data Sources & Feature Engineering

Comprehensive Log Capture

Each logged interaction provides a rich dataset for model training. Records include the **task description** (user intent and requirements), **prompt template** (the specific instruction structure used), **model configuration** (which LLM and parameters), **input/output pairs** (actual request and response content), **quality scores** (human ratings or automated metrics), **safety labels** (violation flags and risk assessments), and **precise token counts** (for accurate cost accounting).

Template Features

- Character and token length
- Number of few-shot examples
- Presence of reasoning directives (chain-of-thought, step-by-step)
- Safety clause inclusion
- Output format constraints (JSON, XML, structured)
- Task type indicators

Context Features

- Application domain (legal, medical, creative, technical)
- Input text length and complexity
- Target language and locale
- Model family and version
- User segment and permissions
- Time of day and seasonality

Target Labels

- **Quality Score:** Aggregated rating (0-100) from automated metrics and human feedback
- **Cost:** Total tokens (input + output) with pricing model applied
- **Safety Flag:** Binary violation indicator with severity level

Utility Function

Overall prompt utility combines multiple objectives: **Score = Quality – λ_1(Cost) – λ_2(Risk)**, where λ weights balance competing priorities based on application requirements. This multi-objective formulation enables flexible optimization across different deployment scenarios.

System Architecture & Data Flow

Data Lake & Logs
Historical interaction data, prompt templates, and user feedback

Feature Store
Preprocessed features, embeddings, and training datasets

Retrieval Service
Fast candidate prompt selection using vector similarity

Ranking Service
ML model scoring and ordering candidate prompts

Policy Engine
Experimentation framework and prompt selection logic

LLM API
Selected prompt execution and response generation

Logging & Monitoring
Performance tracking and feedback collection

End-to-End Request Workflow

1. **Task Submission:** User submits task description and requirements through API or UI
2. **Candidate Retrieval:** System retrieves top-K similar prompt templates from library
3. **Intelligent Ranking:** ML model scores candidates on quality, cost, and safety dimensions
4. **Policy Selection:** Experiment engine chooses prompt based on current A/B test allocation
5. **LLM Execution:** Selected prompt sent to appropriate language model for generation
6. **Outcome Logging:** Response quality, token count, and safety assessment recorded
7. **Feedback Loop:** Logged data fed back into training pipeline for continuous improvement

Core Algorithms & Safety Mechanisms

1

Retrieval: Candidate Generation

The system employs a two-stage retrieval strategy. **BM25 keyword matching** provides fast lexical similarity for structured queries, while **dense embeddings** capture semantic similarity using transformer-based encoders. Candidate prompts are retrieved via **vector search** (FAISS or approximate nearest neighbors) with sub-millisecond latency. This hybrid approach balances precision and recall across diverse task types.

2

Ranking: Multi-Objective Optimization

A **learning-to-rank model** scores retrieved candidates using gradient boosted trees or neural rankers. The model predicts either a composite utility score or separate estimates for quality, cost, and safety risk. Training uses pairwise or listwise loss functions to learn optimal orderings. Feature importance analysis reveals which template and context attributes drive recommendations.

3

Off-Policy Evaluation

Before deploying new prompt policies, we estimate their performance using historical logs through **inverse propensity scoring (IPS)** and **doubly robust estimators**. These techniques correct for the bias introduced by past selection policies, providing unbiased estimates with confidence intervals. This enables safe policy updates without costly online experiments.

4

Safety Classification

A dedicated **safety classifier** predicts violation probability for each prompt-task pair, trained on labeled examples of harmful outputs. The model considers prompt content, task context, and historical violation patterns. Predictions serve as either hard filters (rejecting high-risk prompts) or soft penalties (downweighting in the ranking function). Threshold tuning balances safety and utility based on application risk tolerance.

Model Selection: Gradient boosted trees (XGBoost, LightGBM) provide strong baseline performance with excellent interpretability, while neural rankers (transformer-based) excel on complex semantic patterns. Production systems often ensemble both approaches for robust performance.

API Interfaces & System Design

REST API Endpoints

Core Services

- **POST /recommend** – Submit task, receive ranked prompt recommendations with predicted metrics
- **POST /log_interaction** – Record execution outcome, quality rating, and user feedback
- **GET /templates** – Browse prompt library with filtering and search capabilities
- **POST /experiments** – Create and manage A/B tests for prompt policies

Example: /recommend Request

```
{  "task": "Summarize legal document",  "domain": "legal",  "input_length": 5000,  "model": "gpt-4",  "constraints": {    "max_cost": 0.10,    "safety_required": true  }}
```

Example: /recommend Response

```
{  "recommended_prompt_id": "p_847",  "template": "You are a legal expert...",  "predictions": {  "quality_score": 87,    "estimated_cost": 0.07,    "safety_probability": 0.98  },  "alternatives": [    {"prompt_id": "p_923", "score": 84},    {"prompt_id": "p_612", "score": 81}  ],  "explanation": {    "factors": ["domain match", "low cost",                "high safety compliance"]  }}
```

Server-Side Architecture

The backend follows a **microservices architecture** with containerized services for retrieval, ranking, and policy management. Alternatively, **serverless functions** provide auto-scaling for variable workloads. The **repository pattern** abstracts data access layers, enabling flexible storage backends. The **strategy pattern** allows runtime switching between prompt selection policies (greedy, Thompson sampling, epsilon-greedy) for experimentation.

Client-Side User Interfaces

Task Runner

Primary interface for submitting tasks and viewing recommended prompts. Displays predicted quality, cost, and safety scores with confidence intervals. Users can override recommendations or provide feedback on results.

Prompt Explorer

Browse alternative prompt options with side-by-side comparison. Filter by domain, task type, or performance metrics. View historical success rates and example outputs for each template.

Admin Dashboard

Manage prompt library, create new templates, and monitor system performance. Visualize quality trends, cost breakdowns, and safety violation rates. Configure experiment allocations and policy parameters.

Testing, Deployment & HPC Optimization

Comprehensive Testing Strategy

- **Data Validation:** Schema conformance, type checking, and constraint verification for all ingested data
- **Model Validation:** N-fold cross-validation for ranking and safety models with stratified sampling
- **Off-Policy Evaluation:** Estimate new policy performance on held-out historical data before deployment
- **Integration Tests:** End-to-end workflow validation from task submission through logging
- **Load Testing:** Simulate production traffic patterns to verify latency and throughput requirements

Production Deployment Pipeline

Containerization

Docker images for ranking service, API gateway, and feature computation. Kubernetes orchestration for auto-scaling and resource management.

Feature Flags

Gradual rollout controls with instant rollback capability. Per-user or per-segment activation for targeted testing.

Data Engineering & Continuous Improvement

Robust Data Pipeline Architecture

Our data engineering infrastructure ensures reliable model training and system operation. **ETL processes** extract raw logs from application databases, transform them into standardized feature schemas, and load them into the feature store. **Scheduled pipelines** run daily for data ingestion and weekly for model retraining, with automatic failure recovery and alerting. **Data quality monitoring** tracks schema drift, feature distributions, and label consistency, triggering investigations when anomalies are detected. All datasets are **versioned with lineage tracking** to enable reproducible experiments and model rollback if needed.

AutoML & Hyperparameter Optimization

Finding optimal model configurations manually is time-consuming and suboptimal. Our system employs **automated hyperparameter search** using Bayesian optimization to identify the best ranking model architecture and parameters under latency constraints. The search space includes tree depth, learning rate, feature sampling ratios, and regularization strengths. Models are evaluated on validation sets with multi-objective optimization balancing accuracy and inference speed.

For deployment flexibility, we support **serverless AI platforms** that automatically provision compute resources based on demand, reducing infrastructure costs for variable workloads while maintaining performance guarantees.

Active Learning Loop

The system continuously improves through structured feedback collection. Users provide **quality ratings** and **issue tags** (accuracy problems, formatting errors, safety concerns) on generated outputs. An **uncertainty-based sampling strategy** identifies prompt-task pairs where the model is least confident, prioritizing them for human review. Expert annotators provide ground-truth labels, which are added to the training set. **Periodic retraining** (weekly or triggered by performance degradation) incorporates this feedback, yielding improved prompt selection policies deployed through automated CI/CD pipelines.

Comprehensive Documentation

1

System Architecture Overview

High-level design documents, component diagrams, and data flow specifications for onboarding engineers and stakeholders.

2

API Documentation

OpenAPI specifications with request/response schemas, authentication details, rate limits, and code examples in multiple languages.

3

Model Cards

Detailed model documentation including training data, performance metrics, limitations, bias assessments, and ethical considerations.

4

Operations Runbook

Deployment procedures, monitoring dashboards, common failure scenarios, troubleshooting guides, and incident response protocols.

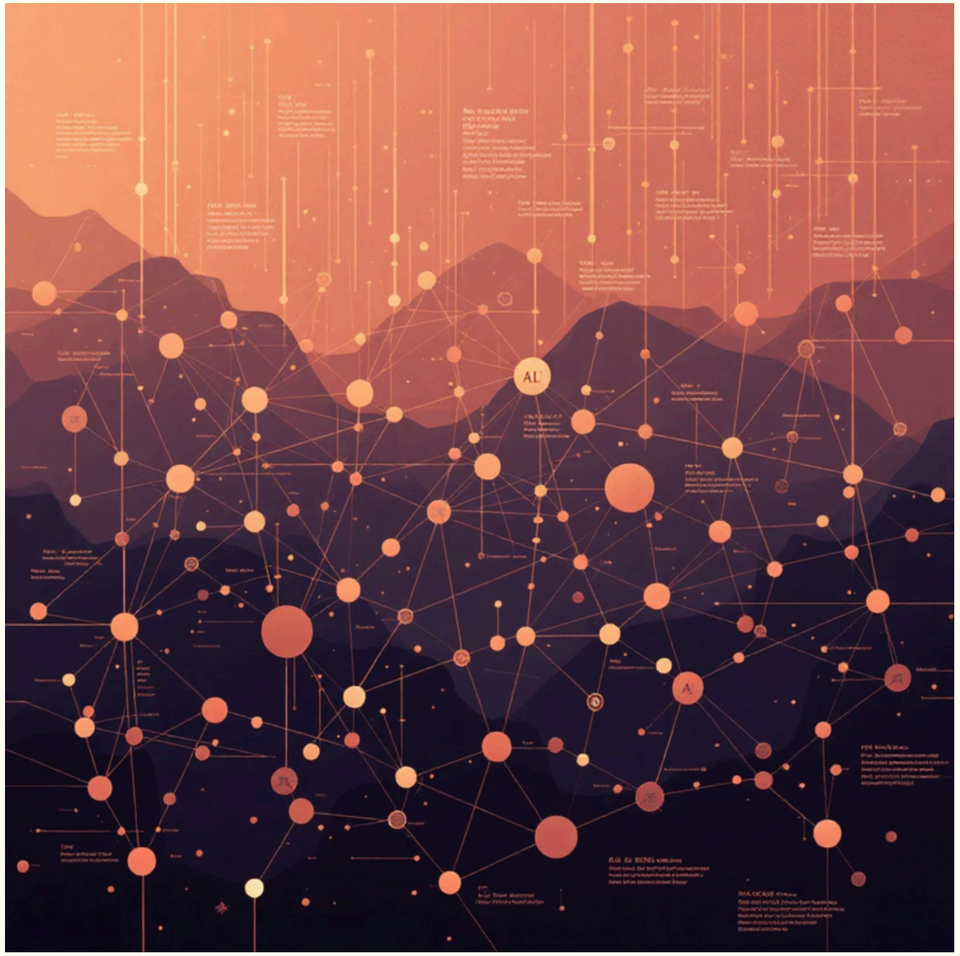
Interpretability, Results & Future Directions
Explainable Recommendations

Trust in the recommendation system requires transparency into its decision-making process. We provide interpretability at two levels:

Global Feature Importance: Identifies which features most strongly influence prompt selection across all tasks. Domain matching, few-shot count, and historical success rate typically emerge as top factors. These insights guide prompt template design and help identify potential biases in the training data.

Local Explanations: For each specific recommendation, we explain "Why this prompt?" using SHAP-style attribution methods. Users see which aspects of their task and which prompt characteristics drove the selection. For example: "Recommended due to: strong domain match (legal), optimal cost-quality tradeoff, 98% safety compliance, successful on 127 similar tasks."

The UI prominently displays predicted **quality score**, **cost estimate** (in dollars), and **safety probability** with confidence intervals, enabling informed human oversight of automated decisions.



Key Achievements & Impact

23%
Quality Improvement

Average task success rate increased compared to baseline manual selection

$0.08
Cost per Task

Reduced from $0.15 through optimized prompt selection and token efficiency

0.3%
Safety Violation Rate

Maintained well below 1% threshold through proactive risk filtering

180ms
Recommendation Latency

Average response time from task submission to prompt delivery