AI-Based Client Clearance Recommendation System

A Comprehensive AI-Powered Software Consulting Framework

Rayonix Solutions
Zaid Ahmad

September 2025

Abstract

This document presents a detailed technical and business report on the AI-Based Client Clearance Recommendation System, an advanced AI-powered engine designed to analyze client inputs and deliver tailored software development recommendations. The solution leverages a hybrid AI architecture combining rule-based logic and semantic similarity matching to provide context-aware suggestions for platforms, features, technology stacks, timelines, and cost estimations. Developed using state-of-the-art natural language processing techniques, this framework is built for scalability, precision, and professional-grade deployment, making it a pivotal tool for software project planning and consultation.

Contents

1 Overview

The AI-Based Client Clearance Recommendation System is an intelligent engine designed to analyze client business requirements expressed in natural language and generate specific, actionable software development recommendations. By integrating rule-based keyword matching with advanced semantic similarity models, the system guides users from vague queries to precise, tailored solutions, ensuring optimal technology selection, feature inclusion, timeline framing, and budgetary projection.

2 Key Features

- **Hybrid AI Architecture**: Combines deterministic rule-based keyword matching with transformer-based semantic similarity analysis to improve intent recognition accuracy and contextual understanding.
- **Context-Aware Clarification**: Detects vagueness in client inputs and dynamically generates intelligent follow-up questions to gather additional essential details.
- **Comprehensive Recommendations**: Outputs detailed suggestions including recommended platforms (e.g., Web, Mobile), critical features aligned with the business domain, technology stack choices, estimated development timelines, and approximate software costs.
- **Scalable Design**: Employs a modular knowledge base architecture easily expandable with new business domains and scenario profiles without retraining the core model.
- **Professional-Grade Implementation**: Structured as modular Python components intended for production, with a maintainable codebase, documented interfaces, and a lightweight runtime environment.

3 System Architecture

3.1 Core Components

The system is architected into three primary modules:

1. **NLP Processing Module** (`nlp_processor.py`):
    - Utilizes the spaCy library with the `en_core_web_md` model for intent classification and custom entity recognition targeting business domains and relevant features.

- Implements heuristic vagueness detection algorithms to flag underspecified user inputs.
- Generates context-sensitive follow-up questions to progressively refine client requirements.

2. **Recommendation Engine** (`recommendation_engine.py`):
   - Employs Sentence Transformers (using the `all-MiniLM-L6-v2` model) to embed client inputs and knowledge base scenarios.
   - Applies cosine similarity to find the closest matching business scenario entries.
   - Assigns confidence scores based on similarity magnitudes for transparent recommendation certainty.
   - Interfaces seamlessly with the NLP module to incorporate refined user context.
3. **Knowledge Base** (`client_scenarios.json`):
   - Contains a curated and extensible dataset of business scenarios representing diverse industries, business types, and software needs.
   - Each scenario includes detailed profiles covering recommended platforms, feature sets, tech stack compositions, development timelines, and cost estimates.

3.2 Technical Stack

- **Natural Language Processing:** spaCy 3.7.2 with `en_core_web_md`
- **Semantic Similarity:** Sentence Transformers 2.2.2, all-MiniLM-L6-v2
- **Similarity Computation:** scikit-learn 1.2.2 cosine similarity metric
- **Data Storage:** JSON format for knowledge base scenarios
- **Environment:** Python 3.10+ on Google Colab or local virtual environment

4 Installation & Setup

4.1 Google Colab Implementation

1. Create a new Google Colab notebook.
2. Execute the dependency installation cell:

```
!pip install spacy==3.7.2 sentence-transformers==2.2.2 scikit-learn==1.2.2 numpy=
!python -m spacy download en_core_web_md
```

3. Run cells sequentially for:
   - Client scenario dataset creation

- NLP processor module
- Recommendation engine module
- Main interactive application

4.2 Local Installation (Alternative)

1. Clone the project repository.
2. Install required packages:

   ```
   pip install -r requirements.txt
   python -m spacy download en_core_web_md
   ```

3. Run the main application via:

   ```
   python main.py
   ```

5 Usage Instructions

5.1 Interactive Mode

1. Start the main application (Colab or local).
2. Enter business requirements in natural language at the prompt (”You:”).
3. The system analyzes input and:
   - Requests clarifying questions if input lacks specificity.
   - Provides comprehensive recommendation profiles for specific queries.
   - Displays confidence scores to indicate recommendation strength.

5.2 Example Inputs

- **Specific (direct recommendations):**
  - “I want to create an online store for handmade products”
  - “I need a delivery tracking app for my courier business”
  - “I run a restaurant and want online ordering”
- **Vague (trigger follow-up queries):**
  - “I need an app for my business”
  - “I want to build something online”
  - “How much would software cost?”

6 Output Format

For each qualified input, the system outputs:

- Recommended platforms (e.g., Web App, Mobile App, Desktop)
- Key domain-specific features
- Suggested technology stack broken down by frontend, backend, and database
- Estimated development timeline
- Cost range estimation
- Confidence score quantifying recommendation reliability

7 Dataset Structure

The knowledge base is structured as JSON objects with the following fields:

```
{
  "id": 1,
  "business_domain": "Retail",
  "business_type": "Clothing Store",
  "client_input": "Example input text",
  "client_input_vagueness": "low/medium/high",
  "required_follow_up": "Clarification question or null",
  "recommended_platform": ["Web App", "Mobile App"],
  "recommended_features": ["Feature 1", "Feature 2"],
  "recommended_tech_stack": {
    "frontend": "React.js",
    "backend": "Node.js",
    "database": "PostgreSQL"
  },
  "estimated_dev_time": "3-5 months",
  "estimated_cost_range": "$20,000 - $40,000"
}
```

8 Customization & Expansion

8.1 Adding New Business Domains

To extend the system's applicability:

1. Update `domain_keywords` within `NLPProcessor`:

   ```
   self.domain_keywords = {
     "new_domain": ["keyword1", "keyword2", "keyword3"],
     ... existing domains ...
   }
   ```

2. Append new scenario entries to `client_scenarios.json` documenting full recommendation profiles.
3. Modify `generate_follow_up()` method to include clarifying questions pertinent to new domains.

8.2 Tuning Performance

- Adjust semantic similarity threshold in `find_most_similar_scenario()` (default typically 0.3) to balance precision and recall.
- Refine vagueness detection heuristic parameters in `analyze_input()`.
- Broaden keyword lexicons to improve intent detection rates.

9 Testing & Validation

- Execute included test cells to confirm core functionalities.
- Test across diverse business scenarios for domain coverage.
- Verify clarification mechanism engages correctly upon vague inputs.
- Cross-check recommendation accuracy against known expected outputs.

10 Performance Considerations

- Precompute and cache embeddings of scenarios for rapid similarity computation.
- Employ lightweight sentence-transformer models to optimize inference speed and resource usage.
- Maintain modular architecture enabling independent scaling and upgrades.

11 Limitations & Future Enhancements

11.1 Current Limitations

- Coverage restricted to predefined business domains in knowledge base.
- Initial intent classification heavily dependent on keyword matching.
- Support limited to English language input only.

11.2 Potential Enhancements

- Integrate large language models (LLMs) for advanced, context-aware conversation capabilities.
- Incorporate real-time data sources to dynamically update technology stack recommendations.
- Expand multilingual capabilities using translation services or multilingual models.
- Develop frontend UI/UX to improve accessibility and client interaction.
- Integrate with project management platforms to facilitate end-to-end software delivery workflows.

12 Ethical Considerations

- Use of synthetic and anonymized datasets to prevent client privacy exposure.
- Transparent communication emphasizing that recommendations are informational estimates, not contractual guarantees.
- Confidence scoring included to allow users to judge trustworthiness of suggestions.

13 Support Resources

- spaCy documentation: `https://spacy.io/`
- Sentence Transformers documentation: `https://www.sbert.net/`
- Google Colab documentation: `https://colab.research.google.com/`

14 License

This project is provided as part of the Rayonix Solutions internship program. Usage is subject to program-specific terms and guidelines.

Prepared by: Zaid Ahmad