AIU. ALAMEIN INTERNATIONAL UNIVERSITY
جامعة العلمين الدولية

SERVIO
smart service discovery

SERVIO:

A RAG-Enabled Smart Service Discovery Tool

- **Project Supervisor**: Dr. Islam Elgedawy
- **Project Leader**: Youssef Badreldin (20100294)
- **Project Members**:
  Omar Elhamrawy (20100357)
  Aly Farrag (20100297)
  Mohamed Ibrahim (21100837)
  Shahenda Adel (21100796)
  Ahmed Mahdi (21100822)

Outline

- Introduction and Motivation
- Background
- Problem
- Available Solutions and Proposed Solution
- User Analysis and Design Requirements
- Literature Review
- Methodology and Design
- Implementation
- Evaluation and Results
- Future Work
- References
- Demo

Introduction and Motivation

In today's digital ecosystem, businesses rely on thousands of interconnected software services.

Service Reuse solve the need for building new softwares for developers

Developers waste 50% of their time searching for services. What if AI could cut this to?

Background

Service Discovery:

Software Service Discovery refers to the process of identifying, connecting, and managing software services across service registries.

Problem

Pain Points:

❌ Excessive results (keyword-based tools).

❌ High manual effort (Consul's latency).

❌ Lack of contextual understanding.

Impact:

- 80% of developers report wasted time.
- Enterprise projects delayed by 15% due to poor discovery.

Available Solutions

Research-Based Tools

Semantic Matching Tools

- There are **many research studies**, such as **Jamejam** and **Usta**, that have been conducted but have not yet been implemented.

Ready Tools

Syntactic Base Matching Tools

- **Eureka**
- **Consul**
- Zookeeper
- Kubernetes DNS
- Etcd
- Nacos

Available Solutions

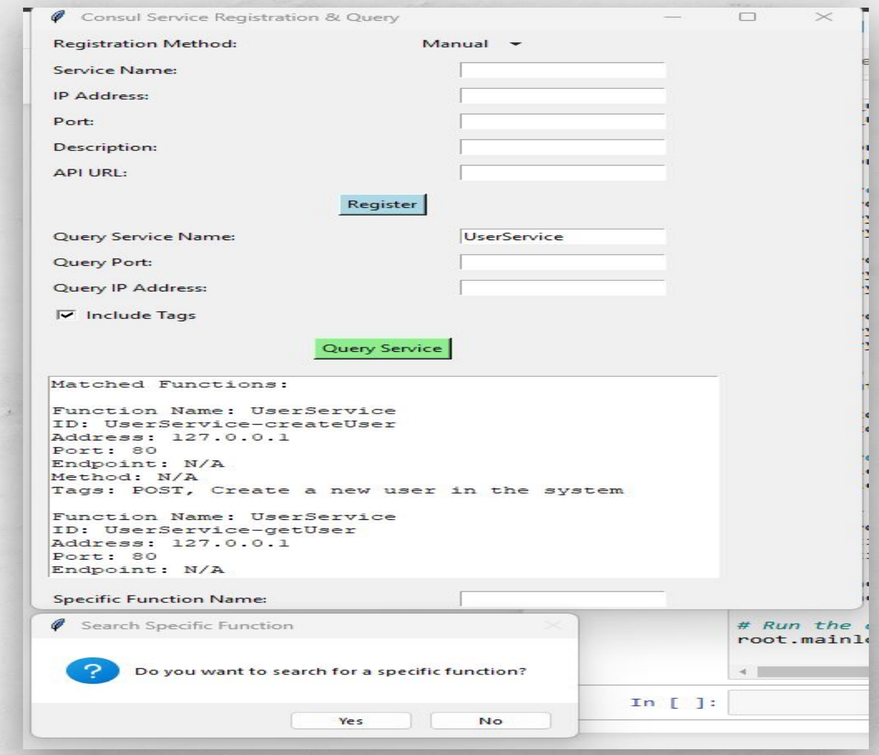
Feature/Capability	Eureka (Netflix)	Consul (HashiCorp)	Research Tools (Jamejam/Usta)
Discovery Method	Keyword matching	Key-value lookup	Semantic matching (prototype)
Context Awareness	❌	❌	⚠️ Limited
Dynamic Filtering	❌	❌	✅
Query Refinement	❌	❌	❌
Management UI	❌	✅	❌
Scalability	✅ High	⚠️ Latency issues	❌ Untested
Implementation	Production-ready	Production-ready	Research-only

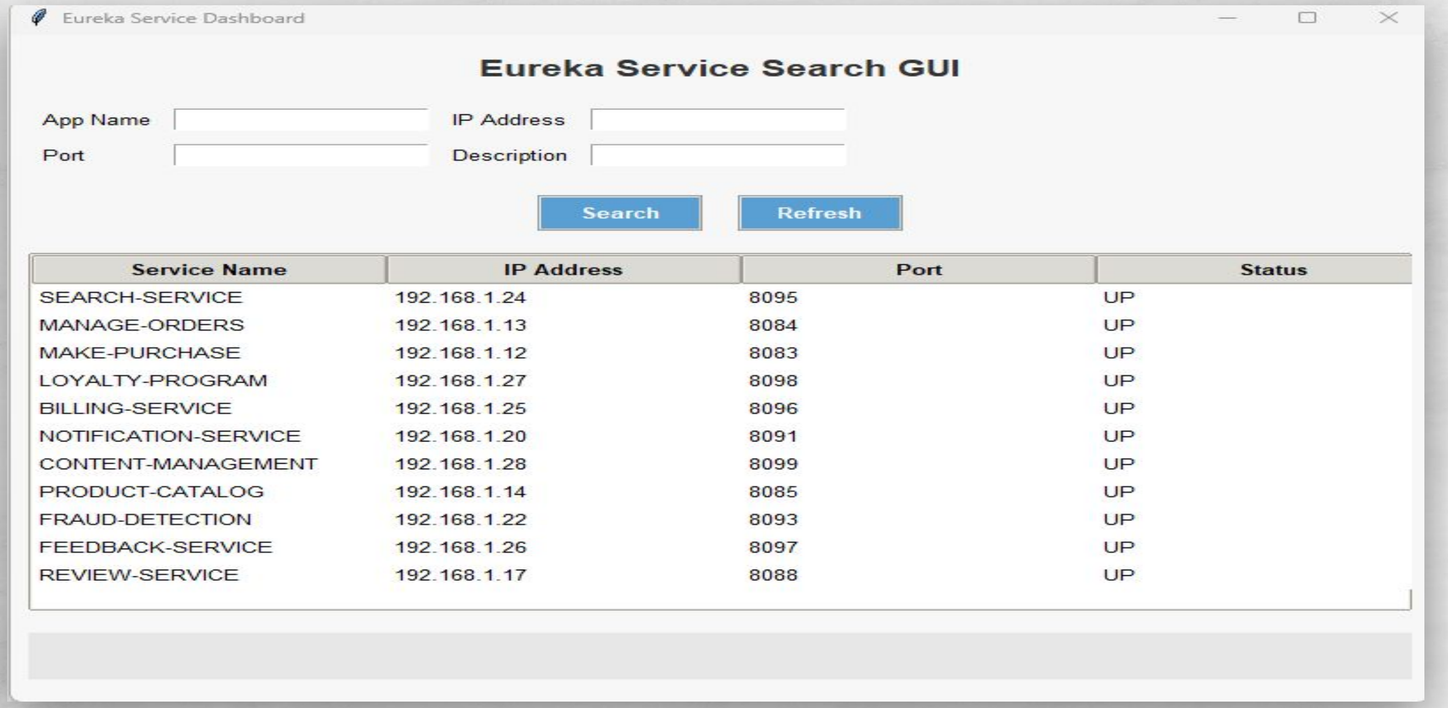
Trials to improve available solutions

Consul

Developed search method with filtering

Eureka





Service Name	IP Address	Port	Status
SEARCH-SERVICE	192.168.1.24	8095	UP
MANAGE-ORDERS	192.168.1.13	8084	UP
MAKE-PURCHASE	192.168.1.12	8083	UP
LOYALTY-PROGRAM	192.168.1.27	8098	UP
BILLING-SERVICE	192.168.1.25	8096	UP
NOTIFICATION-SERVICE	192.168.1.20	8091	UP
CONTENT-MANAGEMENT	192.168.1.28	8099	UP
PRODUCT-CATALOG	192.168.1.14	8085	UP
FRAUD-DETECTION	192.168.1.22	8093	UP
FEEDBACK-SERVICE	192.168.1.26	8097	UP
REVIEW-SERVICE	192.168.1.17	8088	UP

Still rely on key-word matching! Not efficient!!

Proposed Solution

GOAL: Enhance service discovery by integrating AI-driven technologies with conventional search tools.

Objective: To allow the developer to search for a **Atomic** service in fast, efficient and **guided** process.

SERVIO
smart service discovery

User Analysis & Design Requirements

Direct Service Discovery

users enter query, which is executed and used to find the best services.

Guided Service Discovery

chatbot asks structured questions to gather information, then refines the query using LLMs and RAG to ensure accurate service discovery

Literature Review

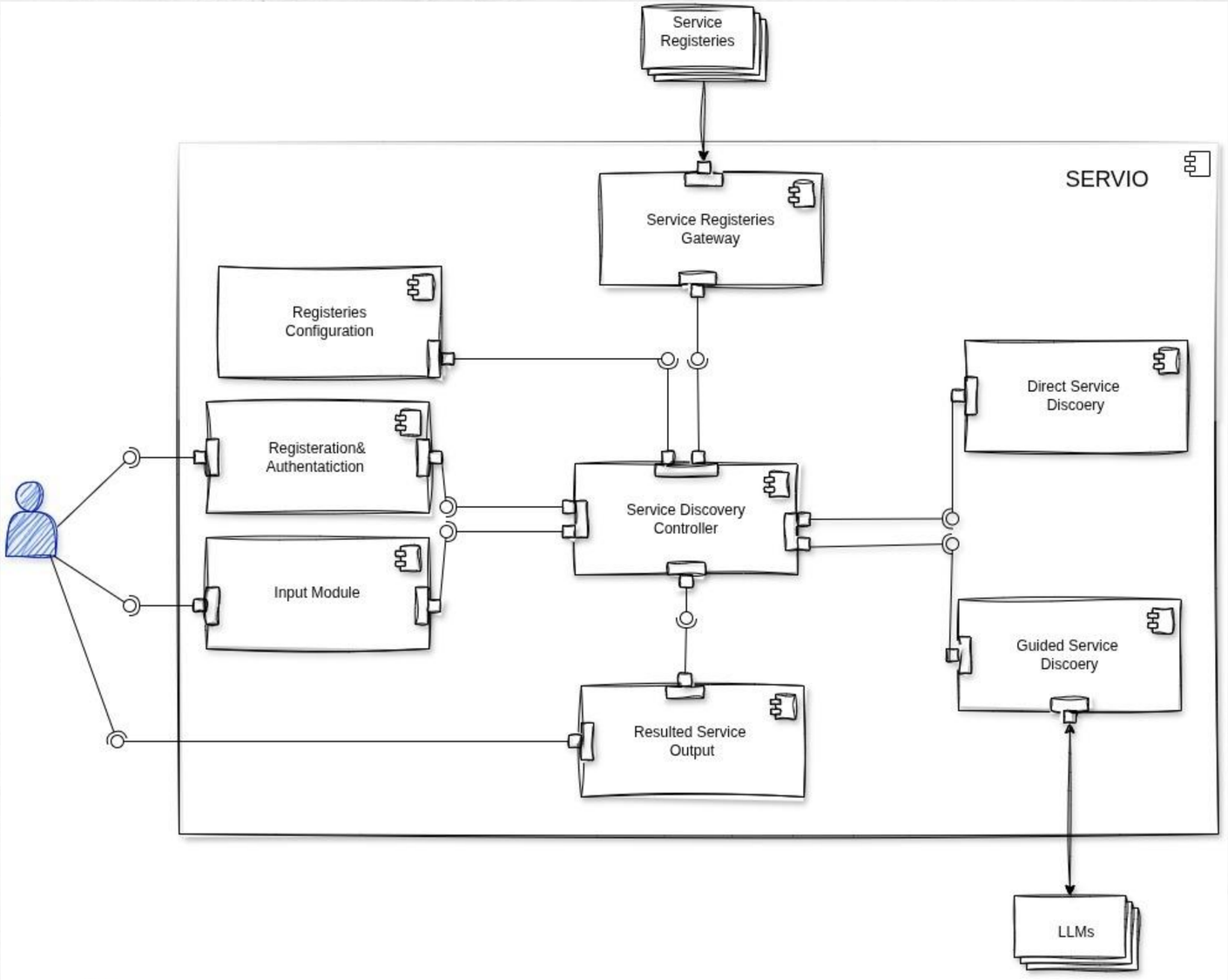
- **Traditional Syntactic Tools (e.g., Netflix Eureka, HashiCorp Consul)**
  - **Method:** Rely on basic keyword matching.
  - **Limitation:** Lack contextual understanding, leading to inaccurate results and high manual effort.
- **Academic Semantic Frameworks (e.g., USTA, JAMEJAM)**
  - **Method:** Propose advanced semantic and aspect-oriented models.
  - **Limitation:** Primarily theoretical; not implemented as production-ready, scalable tools.

The Research Gap

A clear gap exists for a production-ready tool that combines the scalability of traditional systems with the contextual intelligence of modern, AI-driven semantic search.

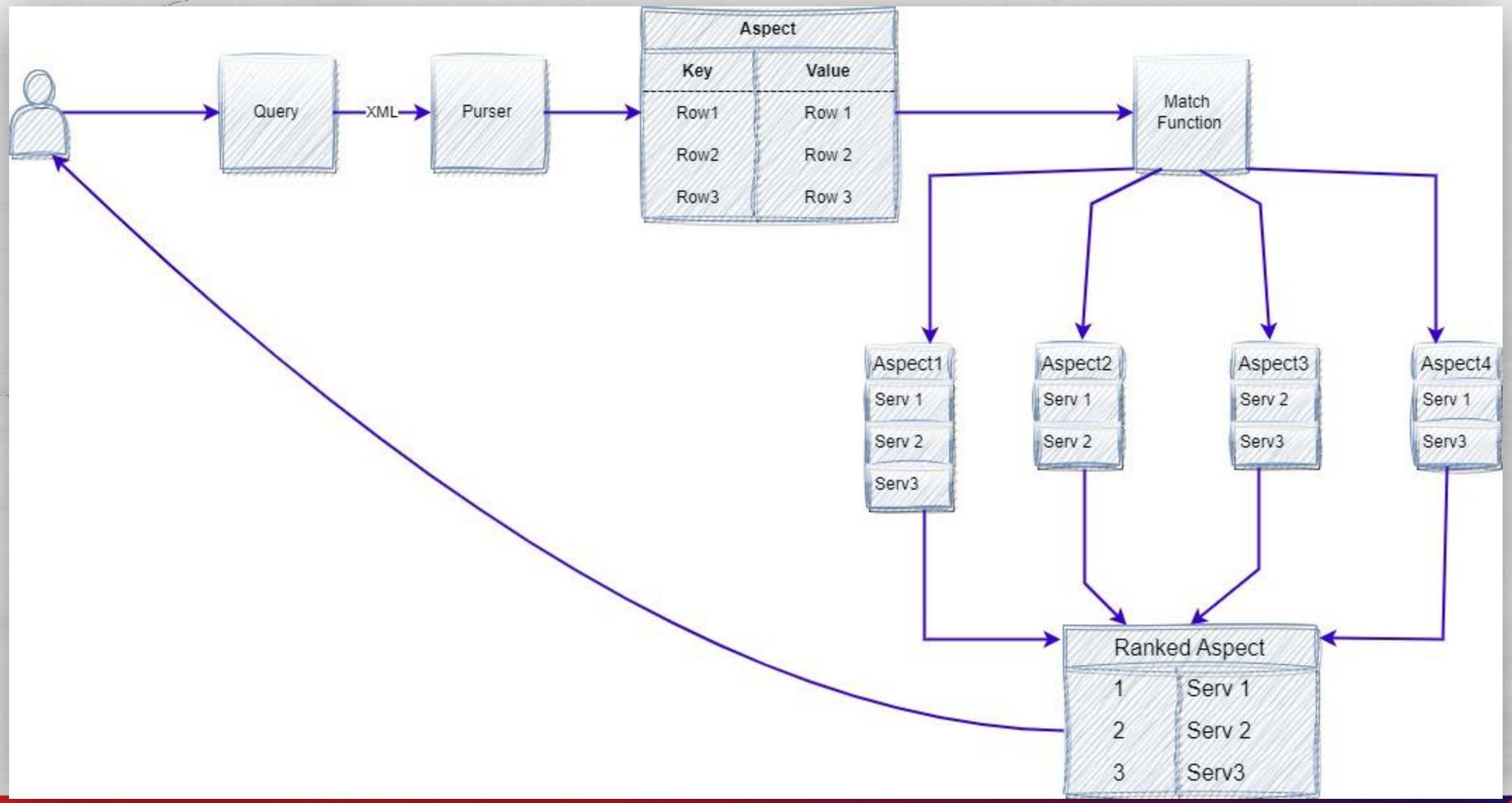
Design

Component Diagram



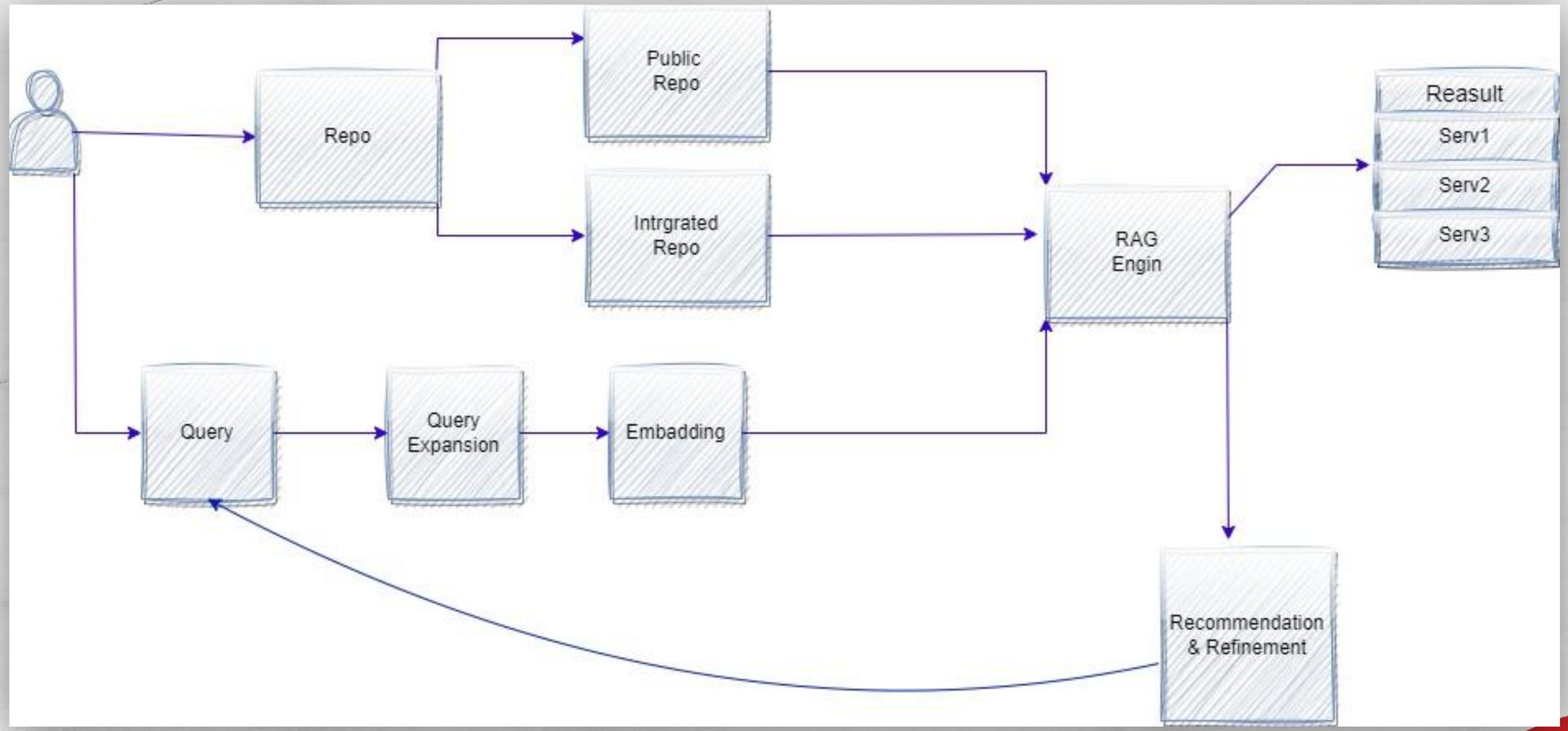
Methodology

Direct Service Discovery



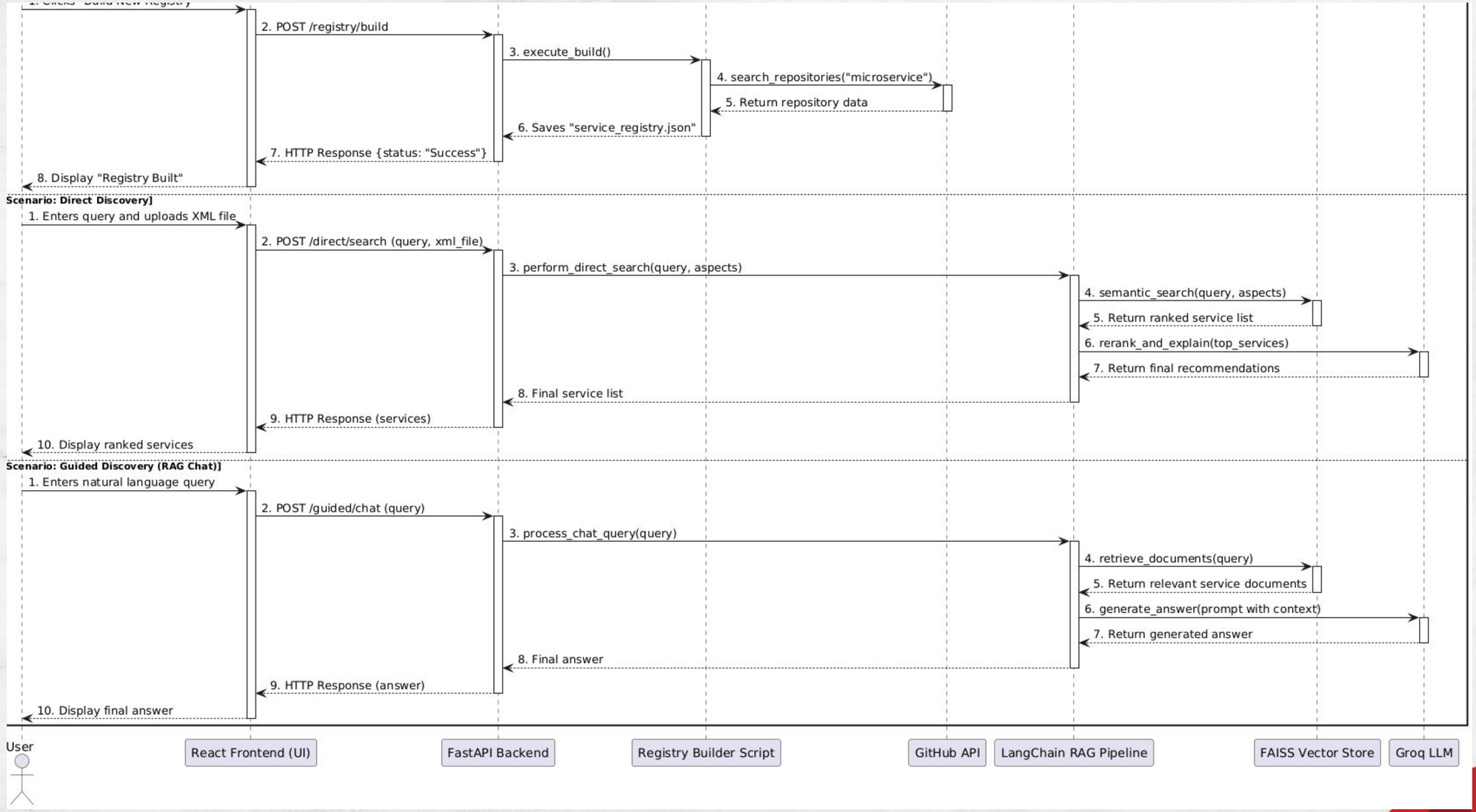
Methodology

Guided Service Discovery



Methodology

Sequence Diagram



Methodology

Task Allocation

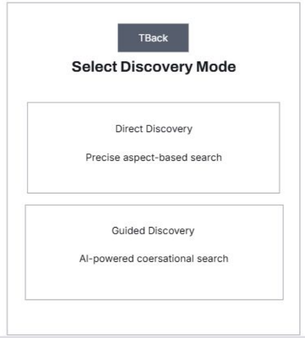
Name (ID)	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
Youssef Badreldin (20100294)	FAST API Endpoint Testing	API Documentation	Feedback System Implementation	Integration Testing	Cross-Browser Testing	Search History Feature	Demo Preparation	Final Project Report
Omar EL hamrawy (20100357)	System Architecture Documentation	UI/UX Design	React.js Frontend Implementation	Direct Module UI Integration	Guided Module UI Integration	Bug Fixing & Optimizatio n	Final UI Polish & Responsivene ss	Final Project Report
Aly Farrag (20100297)	Dataset research	Mythodology research	Guided module V1 Rag (DIFY)	V1 vector DB (MyScale)	V2 JSON RAG	V2 XML RAG	V2 conversational chatbot	V2 query expansion
Mohamed Ibrahim (21100837)	DIFY RAG in docker env	Langchain RAG V1	Build dataset (data registry)	V2 UML RAG	V2 SWE Document RAG			
Shahenda Adel (21100796)	Research on solutions	Filter enhancements	Naive Parallel Search Algorithm	Ranked Parallel Search Algo	Implement w/ LLMs & MCP			
Ahmed Mahdi (21100822)	Syntacting Search Algorithm	Syntacting with Aspects	Sequential Semantic Algorithm	Semantic Search with LLMs	Implement w/ LLMs & MCP			

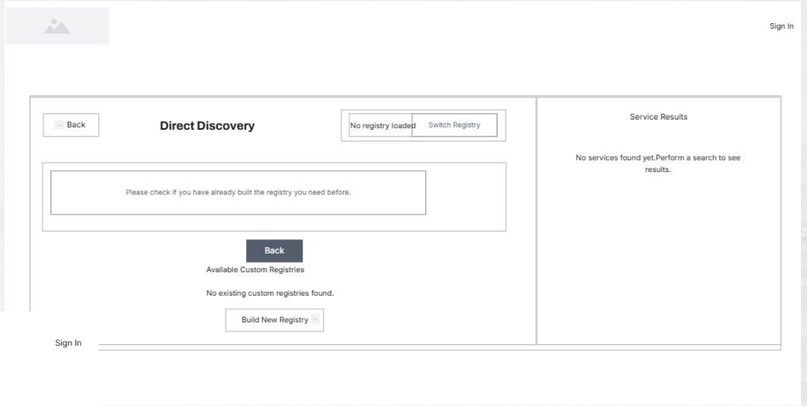
Methodology

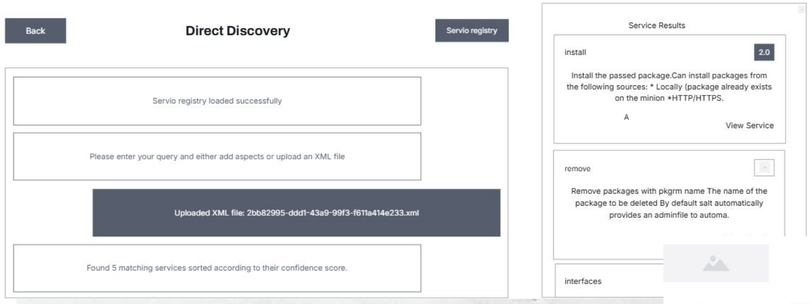
Tools & Technologies

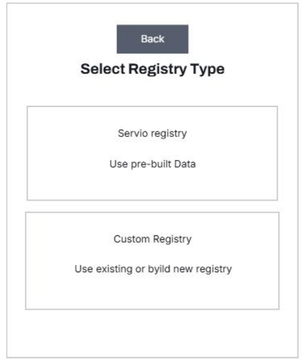
Web Development	RAG	LMMs	Database
React	LangChain	groq™	MYSCALE
FastAPI	Dify	Ollama	mongoDB

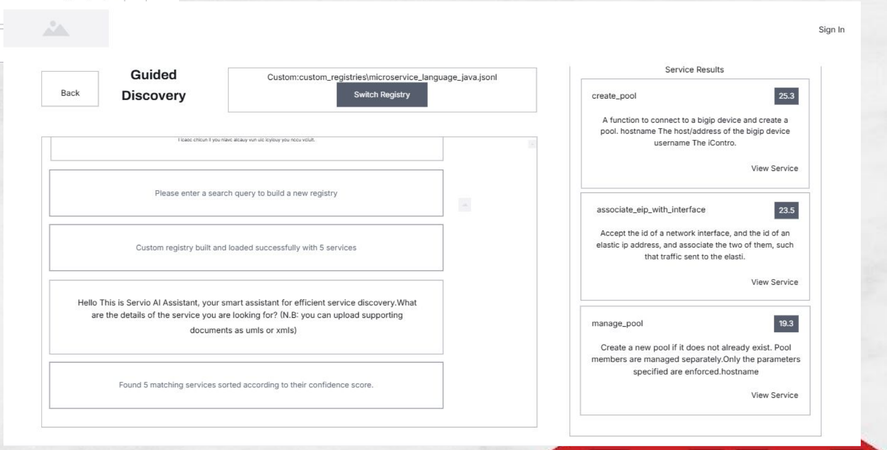
Design
Wireframes











Available Datasets

Dataset	Type	Size	Strengths	Weaknesses
CodeSearchNet	Code + Docstrings	2M+ funcs	Deep code semantics, multi-language	No API structure, noisy
API.guru	OpenAPI Specs	~3K APIs	Clean endpoint metadata	Web-API only, small scale
GitHub Archive	Raw Repositories	50M+ repos	Real-world projects, rich metadata	Requires heavy preprocessing
Stack Overflow	Q&A Posts	50M+ posts	Problem-solution pairs, tutorials	Unstructured, opinionated
Postman Public APIs	API Collections	25K+ APIs	Testable requests, environments	Vendor-locked format
Apache Jena	Ontologies	100s of KGs	Formal service taxonomy	Academic, sparse adoption

Chosen Dataset Description

CodeSearchNet Dataset

Field	Description
`repository_name`	Source repo (e.g., `apache/spark`)
`func_path_in_repository`	File path (e.g., `src/auth/jwt.py`)
`func_name`	Function name (e.g., `validate_jwt`)
`func_code_string`	Raw implementation code
`func_documentation_string`	Docstring (e.g., `"Validates JWT using RSA-256"`)
`language`	Programming language
`func_code_url`	GitHub permalink

Chosen Dataset Description

{"repo": "saltstack/salt", "path": "salt/modules/neutronng.py", "func_name": "security_group_rule_delete", "original_string": "def security_group_rule_delete(auth=None, **kwargs):\n   '''\n   Delete a security group\n\n name_or_id\n   The unique ID of the security group rule\n\n neutronng.security_group_rule_delete name_or_id=1dcac318a83b4610b7a7f7ba01465548\n\n   '''\n   cloud = get_operator_cloud(auth)\n   kwargs = _clean_kwargs(**kwargs)\n   return cloud.delete_security_group_rule(**kwargs)", "language": "python", "code": "def security_group_rule_delete(auth=None, **kwargs):\n   '''\n   Delete a security group\n\n   name_or_id\n   The unique ID of the security group rule\n\n   CLI Example:\n\n   .. code-block:: bash\n\n   salt '*' neutronng.security_group_rule_delete name_or_id=1dcac318a83b4610b7a7f7ba01465548\n\n   '''\n   cloud = get_operator_cloud(auth)\n   kwargs = _clean_kwargs(**kwargs)\n   return cloud.delete_security_group_rule(**kwargs)", "code_tokens": ["def", "security_group_rule_delete", "(", "auth", "=", "None", ",", "*", "*", "kwargs", ")", ":", "cloud", "=", "get_operator_cloud", "(", "auth", ")", "kwargs", "=", "_clean_kwargs", "(", "*", "*", "kwargs", ")", "return", "cloud", ".", "delete_security_group_rule", "(", "*", "*", "kwargs", ")"], "docstring": "Delete a security group\n\n name_or_id\n   salt '*' neutronng.security_group_rule_delete name_or_id=1dcac318a83b4610b7a7f7ba01465548", "docstring_tokens": ["Delete", "a", "security", "group"], "sha": "e8541fd6e744ab0df786c0f76102e41631f45d46", "url": "https://github.com/saltstack/salt/blob/e8541fd6e744ab0df786c0f76102e41631f45d46/salt/modules/neutronng.py#L555-L571", "partition": "train"}

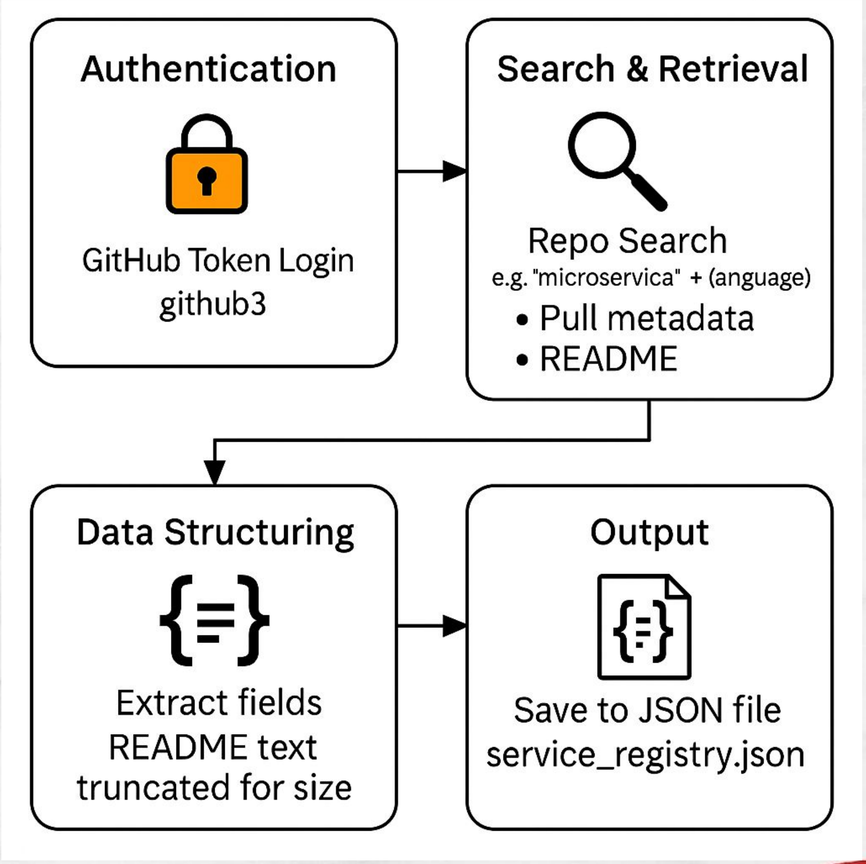
Need to use other Dataset??!

Don’t worry! We are problem driven not data driven

GitHub Microservice Registry Builder

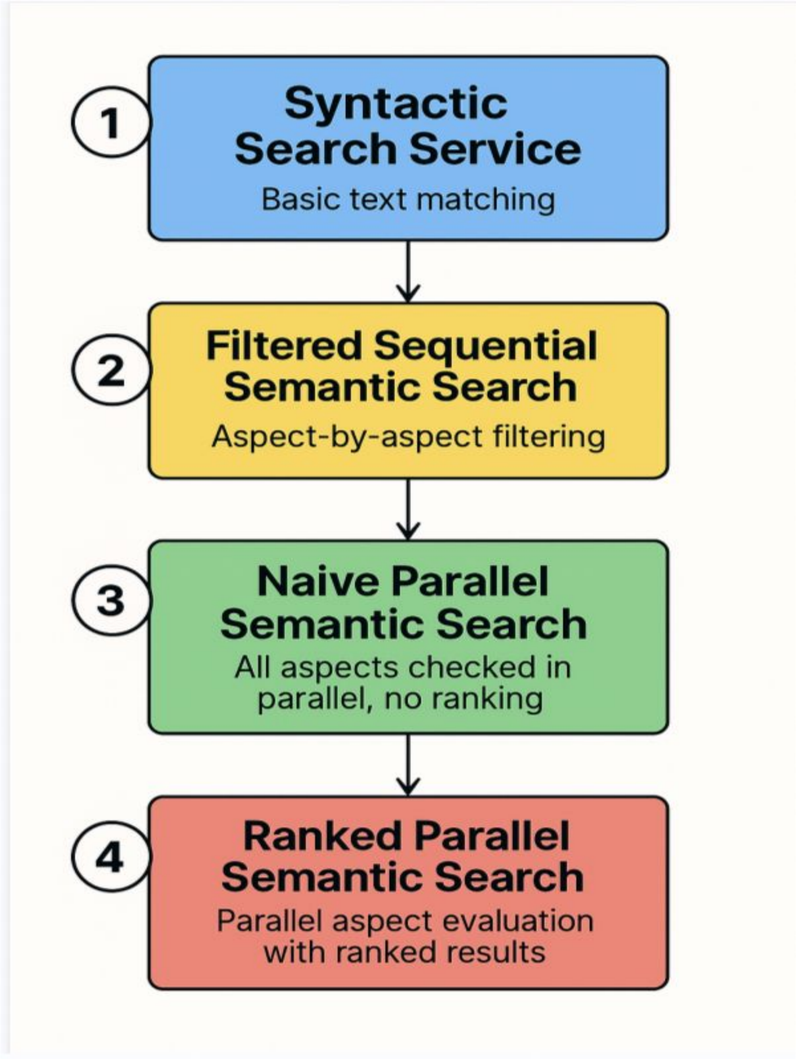
Algorithm Workflow

1. **Setup**
   - Install and import `github3.py`, `requests`, and `json`
   - Authenticate using a GitHub Personal Access Token (PAT)
2. **Define Repository Search**
   - Use GitHub's search API with the query `microservice language:python`
   - Sort repositories by stars, in descending order
3. **Fetch Repository Details**
   - Extract metadata:
     - Name, full name, description, URL, stars, forks
4. **Export Registry**
   - Save all structured data to `service_registry.json` for later use



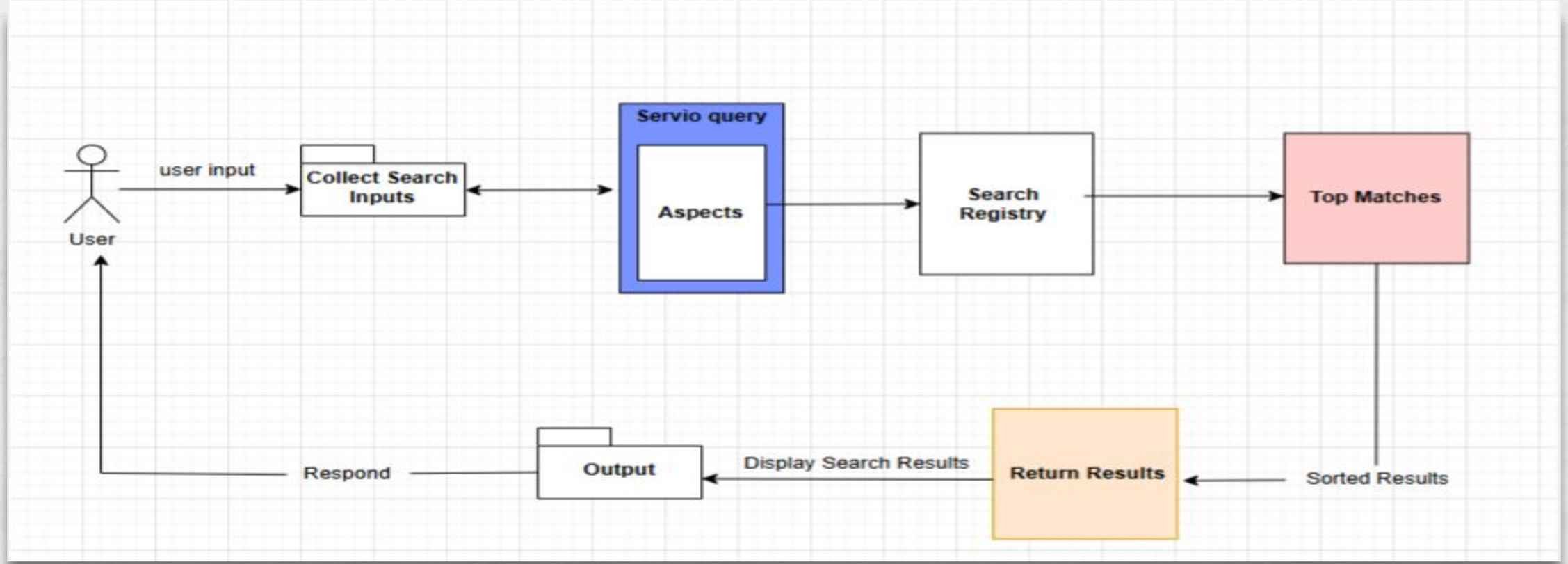
Implementation

Direct Service Discovery flow



Implementation

Direct Module V1 (Syntacting)



Implementation

Direct Module V1 (Syntacting)

Syntactic Service Matching

Load Registry
Read services from a JSONL file

↓

Parse Aspects
Extract aspects defined in an XMl.fle

↓

Preprocess Text
Lowercase and normalize the text

↓

Match Services
Compute overall syntactic similarity scores to find the top matches

Service Matching

Overall Matching:

- Iterate through all: defined aspects for each registry entry.
- Compute and sumn cosine similarity scores for each aspect.
- Only consider entries that meet or exceed a minimum similarity threshold for a set number of aspects.

Ranking

- Sort entries by their cumulative similarity score.
- Return the top matching services.

→

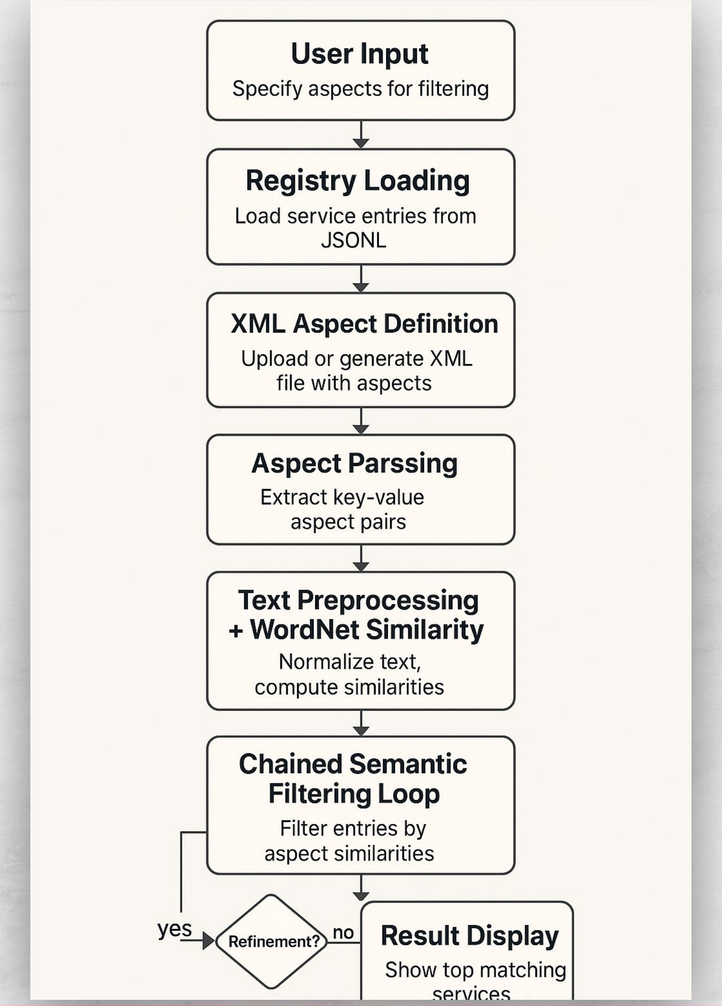
Optional User Refinement

Allow users to adjust aspect definitions and update search queries to improve matching results

Implementation

Direct Module V2 (Semantic Using WordNet Dictionary)

(Sequential)



Implementation

Direct Module V3 (Semantic Using WordNet Dictionary)

(naive parallel)

Text Preproccessing & Similarity Computation

Normalization: Convert text to lowercase and remove extra spaces

Enhanced Similarity Function

Substrings Check: Use WordNet synsset and Wu-Palmer similarity measure

Semantic Similarity : Combine basic string similarity with aspect similarity

Service Matching Algorithm

Overall Matching

- Compute similliarity scores tirough all defined aspects
- Hide weak matches for eacn aspect 1 per aspect
- Refine

Sort and return top entries based on cumulative similarity

User Interaction & Refinement Loop

Display Results

- Display score's
- Hide weak matches
- Refine

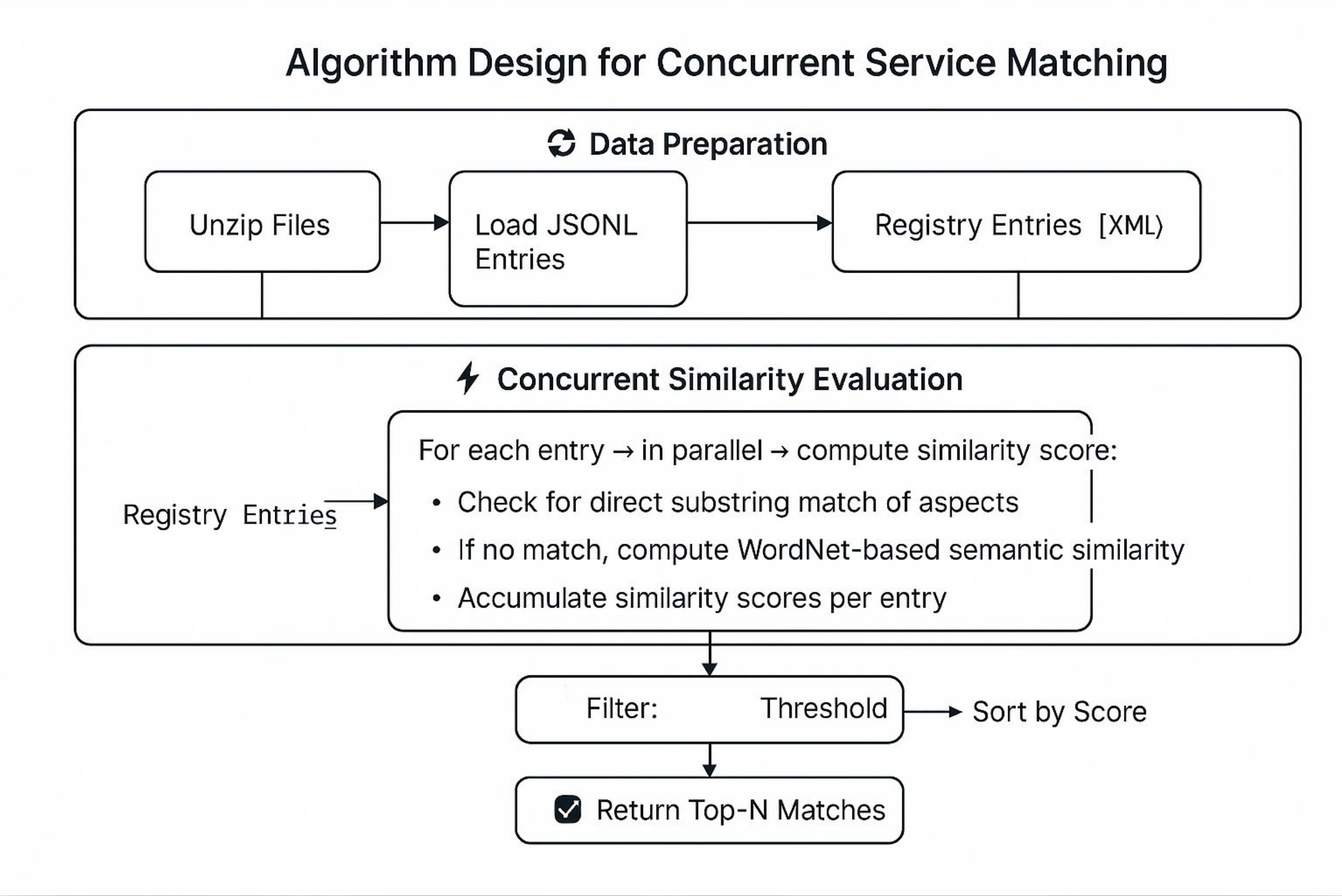
Per-Aspect Refenemet

- Evaluate and display matches for each aspect individually to create detailed insights

Implementation

Direct Module V4 (Semantic Using WordNet Dictionary)

(Ranked Parallel)

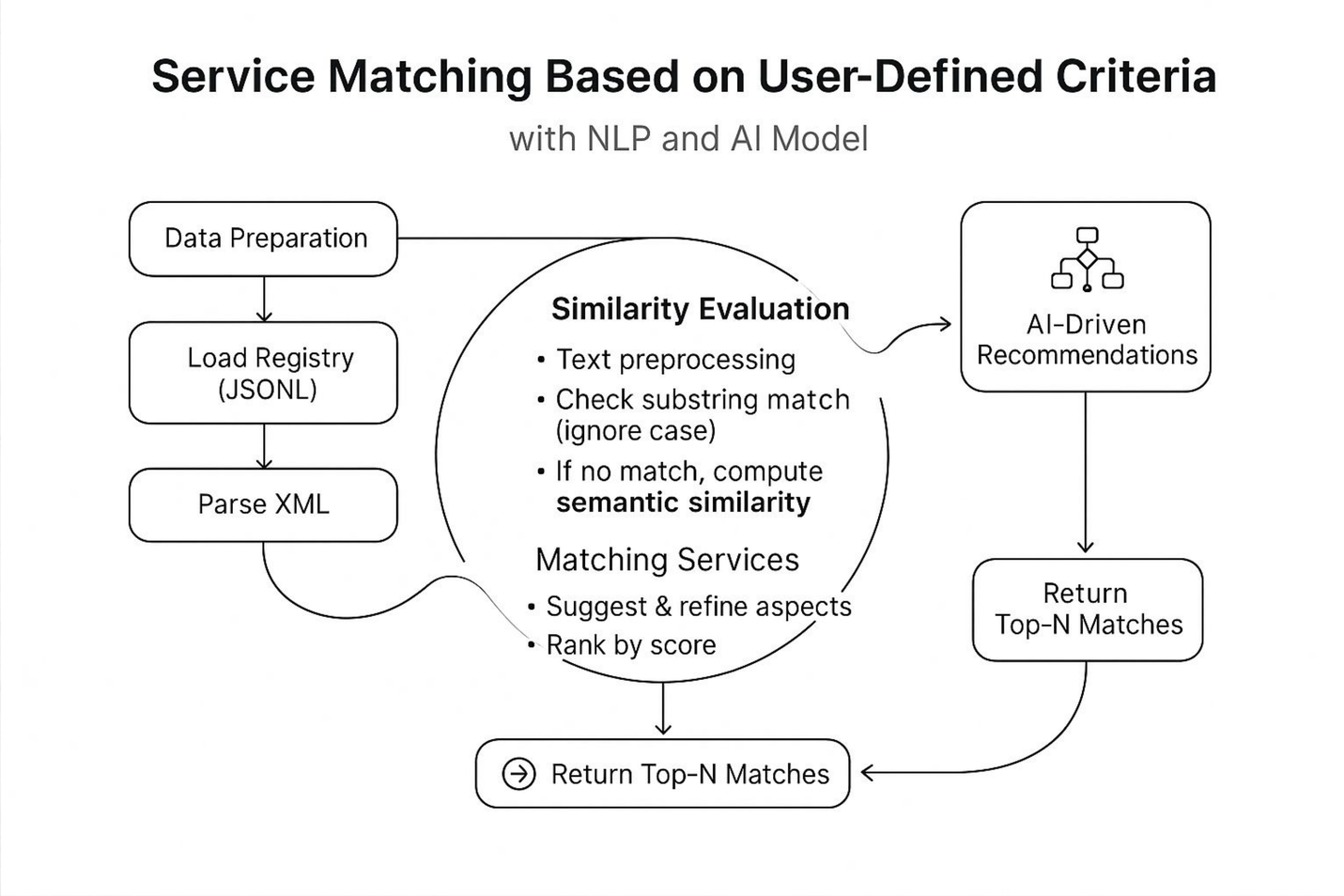


Implementation

Direct Module V5 (Semantic with LLM)

llama3

(Groq)

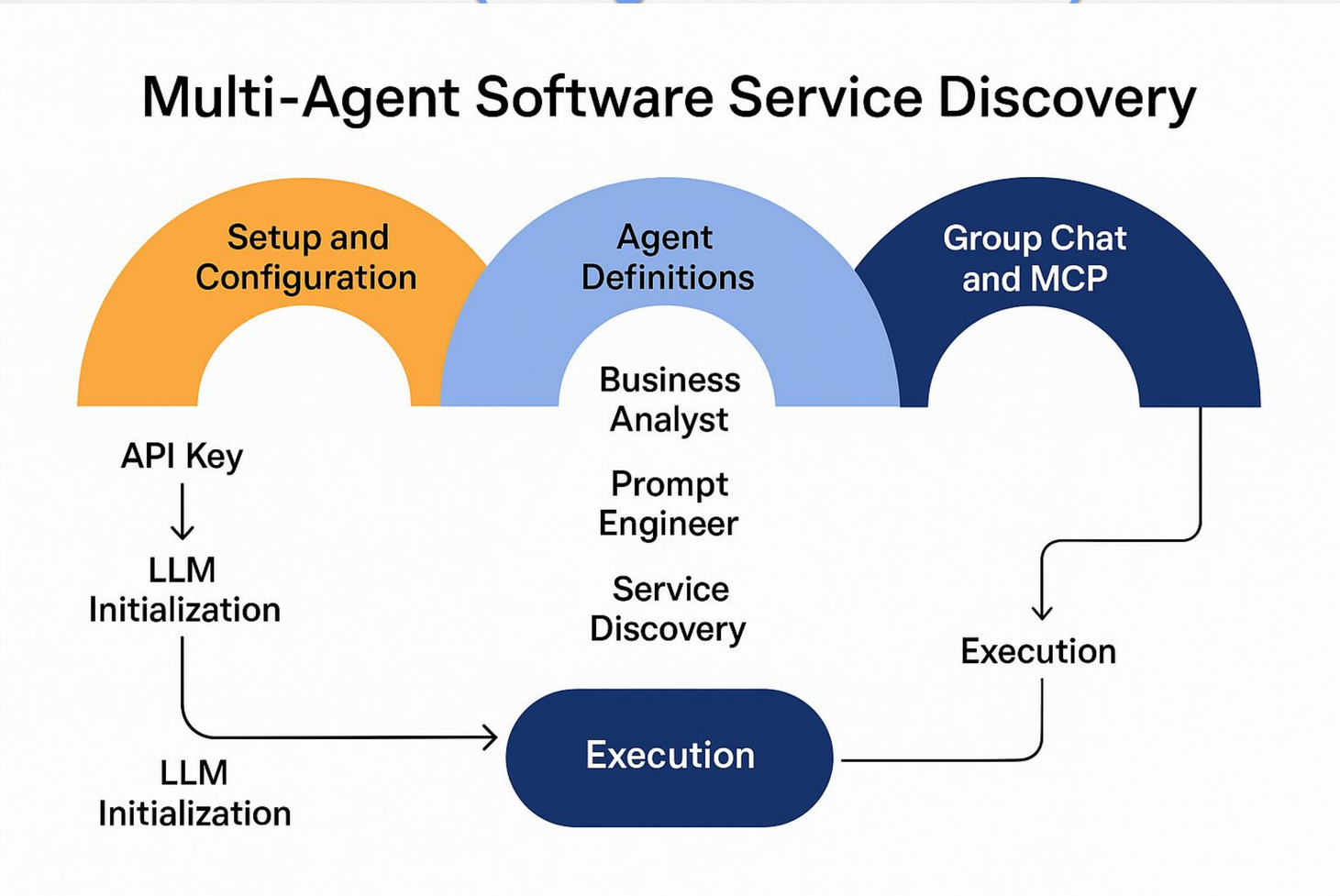


Implementation

Direct Module V5 (AI Agents with MCP)

llama4(Groq)

Autonomous Service Discovery Agents



Direct Module versions

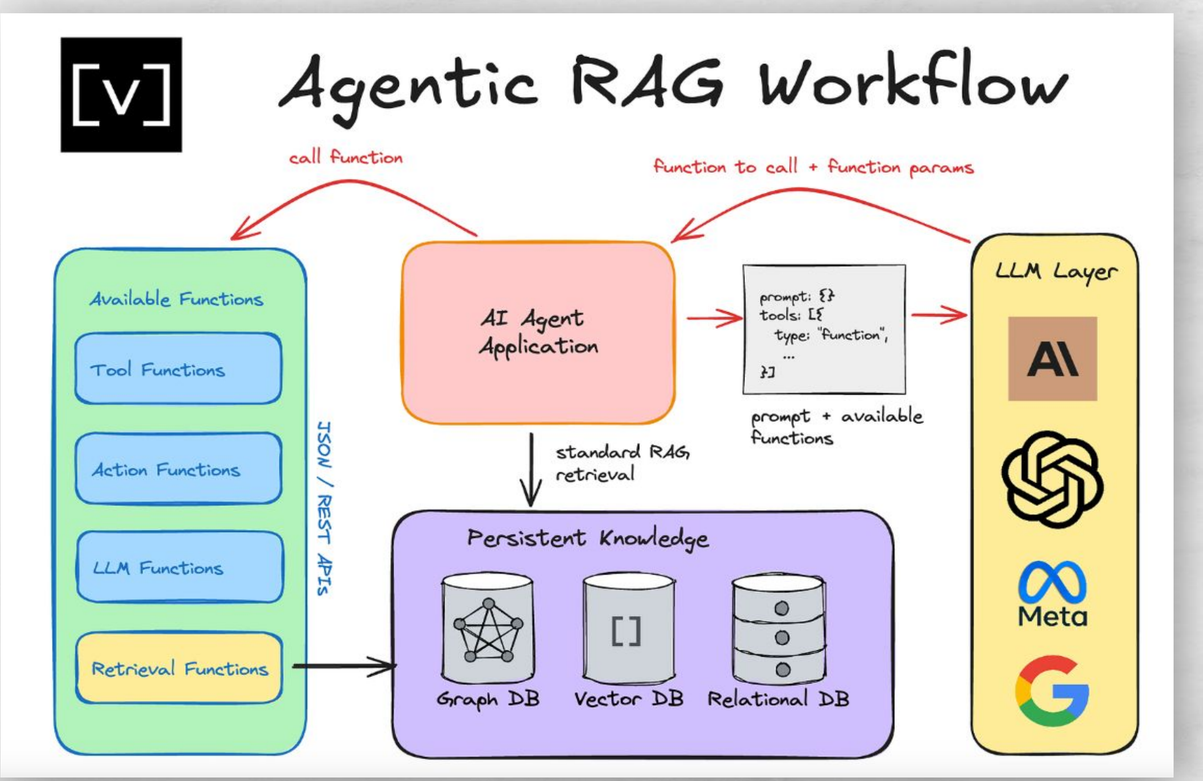
Strategy	Implementation Approach	Execution Style	Result Quality	Complexity
🟦 **Syntactic Search**	Basic string matching (e.g., keyword or substring)	Simple sequential	Low — surface-level matches	🟢 Very Low
🟨 **Filtered Sequential Semantic**	Sequential filtering — each aspect filters previous result set	Aspect-by-aspect narrowing	Medium to High	🟡 Moderate
🟩 **Naive Parallel Semantic**	All aspects matched in parallel, independently	Parallel per-aspect	Medium	🟠Medium -High
🟥 **Ranked Parallel Semantic**	Parallel search + weighted scoring + ranking	Parallel with cumulative scoring	🔥 High — most accurate	🔴 High
🟩 **Semantic with LLM**	Parallel search + weighted scoring + ranking + LLM agent(Groq)	Parallel with cumulative scoring + LLM	🔥 Very High — most accurate	🔴Very High

Implementation

Guided Module

RAG Chatbot

Built a rag agent by first preprocessing the raw data into a vector database using **Myscale**, then using the RAG method using **Dify.ai** Platform.

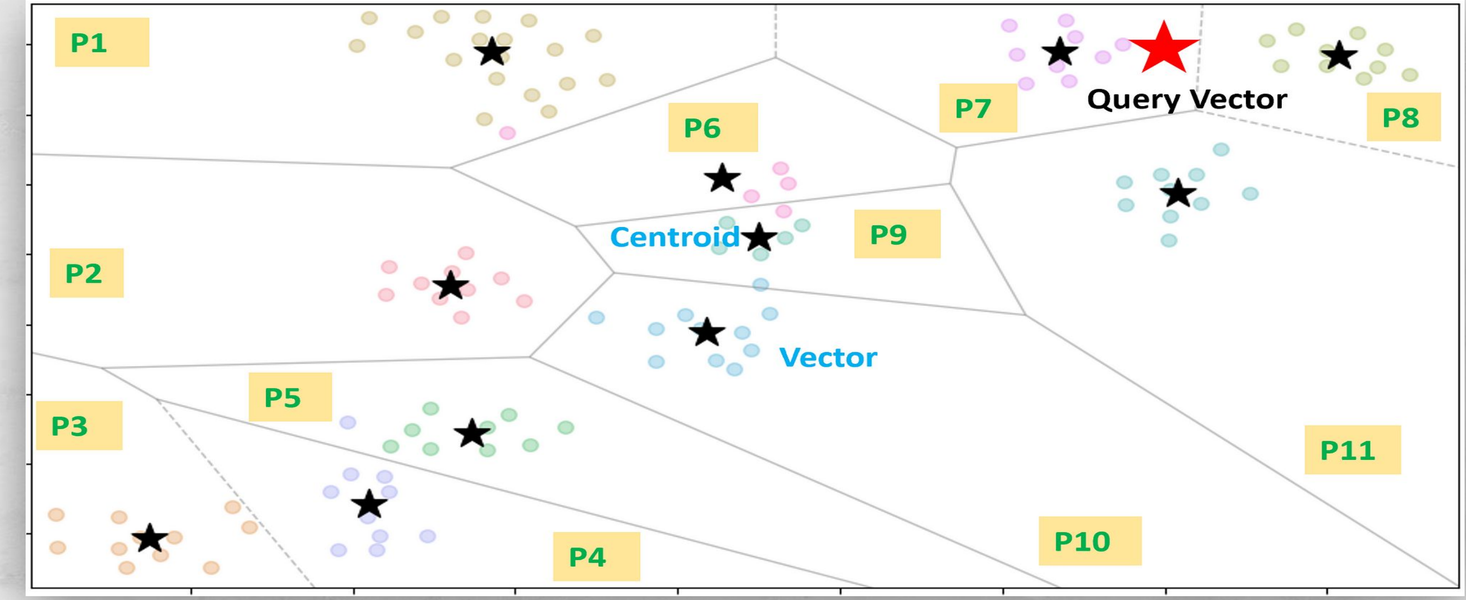


Implementation

Guided Module

Vector similarity search Using Milvus

Built a scalable vector similarity search system by first preprocessing the raw data into a vector database using Myscale, then implementing efficient nearest-neighbor search using **FAISS** for high-performance similarity retrieval.



Implementation

Guided Module

LLM by Langchain

Orchestrating LLM queries with vector database retrieval for scalable similarity search.

Implementation

Guided Module

First Version

RAG Chatbot in Python environment

- **Transformers Library**
  - Model: `sentence-transformers/all-MiniLM-L6-v2`.
  - Purpose: Embedding textual data & Conversation functionality.

- **Custom Implementation**
  - Query Expansion: Leveraging **WordNet** for synonym generation.
  - Answer Generation: Retrieve best matched service
  - Query Refinement: Allow user to add more info to the query for more specified answer.

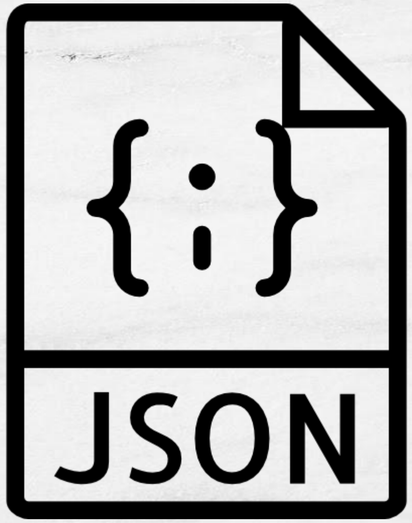
Implementation

Guided Module

Three specialized RAG pipelines

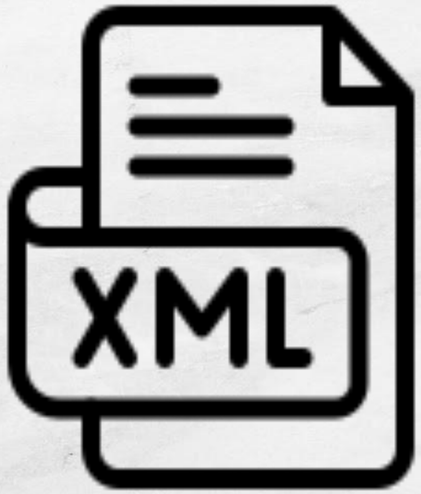
JSON RAG

- Processes structured JSON service registries
- Extracts metadata like endpoints.



XML RAG

- Handles XML-based service definitions
- Uses custom parsers to normalize hierarchical data before vectorization.



UML RAG

- A unique pipeline for visual inputs

Implementation

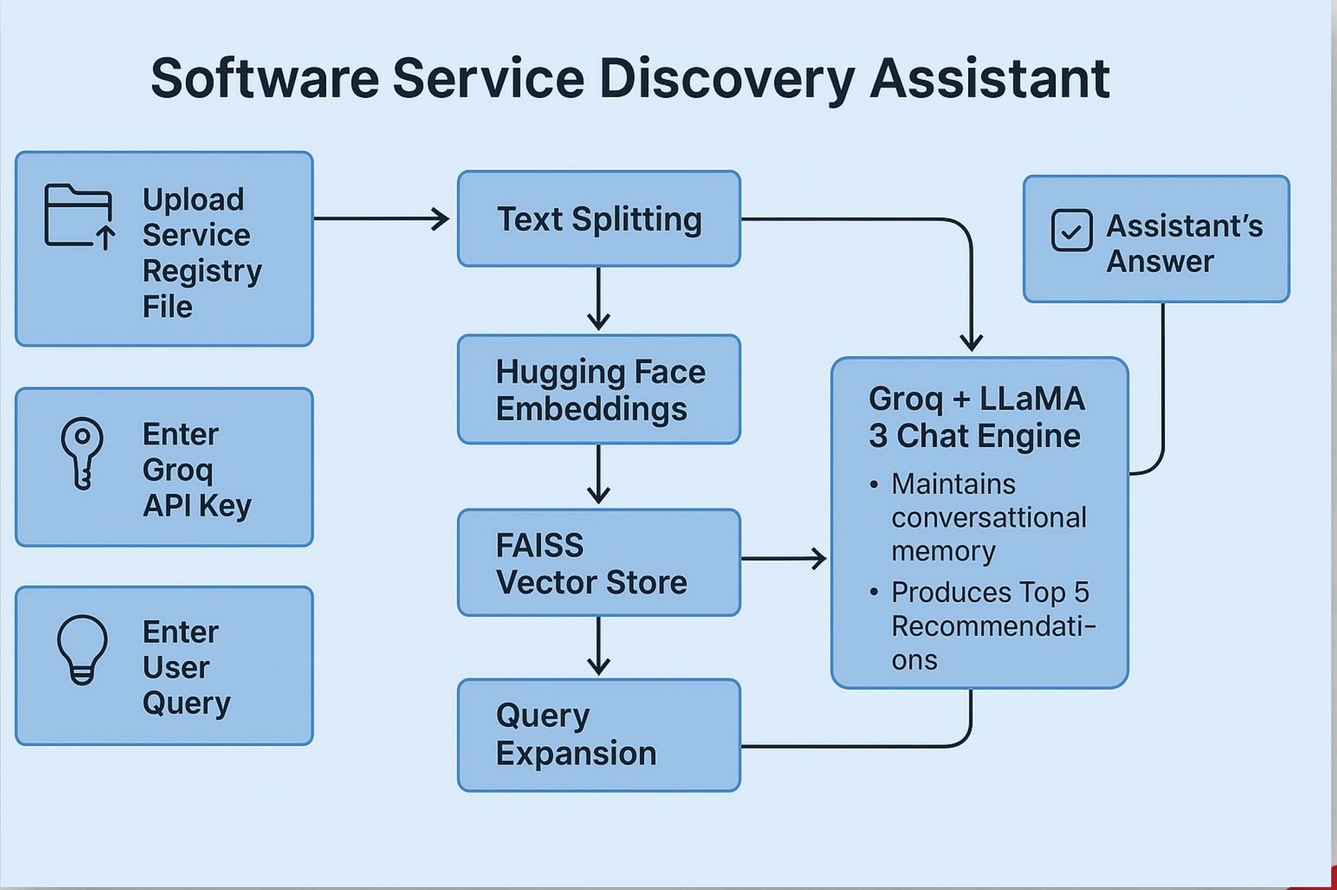
Guided Module

Input
- Upload service registry (JSON/XML)
- Submit user query

Processing
- Text splitting → Chunking
- Hugging Face embeddings
- FAISS vector storage

AI Enhancement
- Query expansion
- Groq + LLaMA 3 chat engine
- Maintains conversation memory

Output
- Top 5 recommended services
- Context-aware explanations



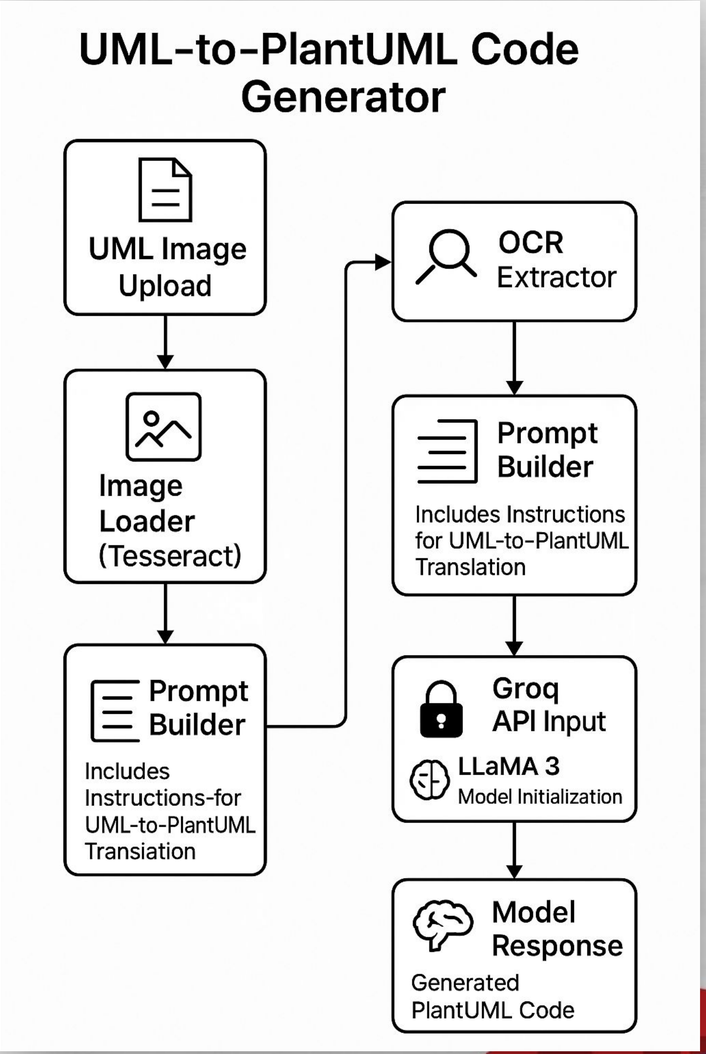
Implementation

Guided Module

UML Service Search

Algorithm Workflow

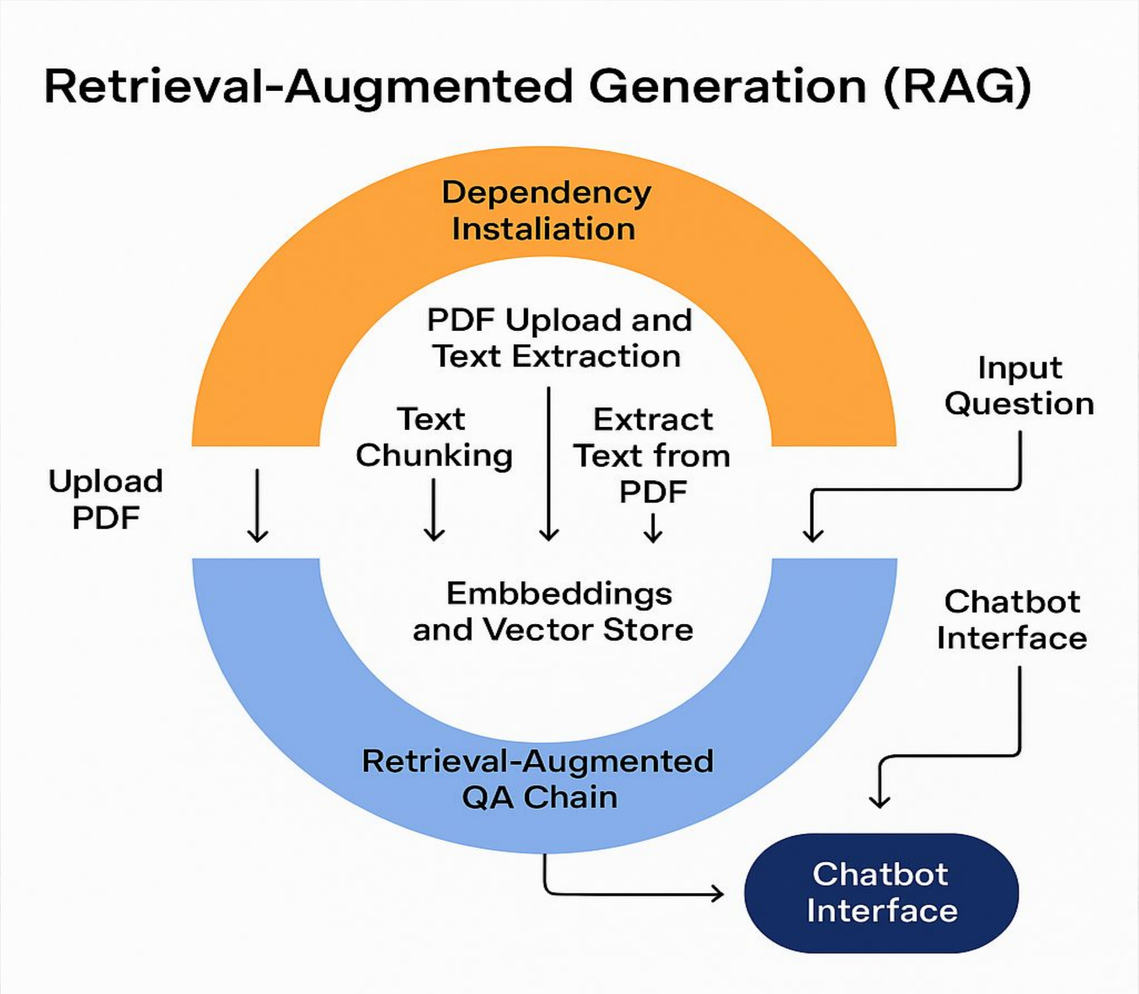
1. **OCR Extraction**
   - Use `pytesseract.image_to_string()` to extract textual data from UML image
2. **Model Initialization**
   - Securely check or prompt for Groq API key
   - Initialize the LLaMA 3 model via `LangChain` with the Groq backend
3. **Code Generation**
   - Submit the prompt to the model
   - Parse and display generated PlantUML code
   - Display resulted services



Implementation

Guided Module

Chat with software documentation



Evaluation & Results

Using 100 Queries from dataset

Discovery Type	Accuracy (%)
Direct (Syntactic, V1)	27
Direct (Sequential Semantic, V2)	48
Direct (Naive Parallel Semantic, V3)	52
Direct (Ranked Parallel Semantic, V4)	63
Direct(Semantic with LLM)	80
Guided (RAG, V4)	87

Conclusion and Future Work

- **High Accuracy**: Guided Discovery (RAG-based) achieves 87% accuracy, outperforming syntactic baselines (Eureka, Consul) by over 203% (Section 4.7).
- **User-Friendly Design**: Intuitive React.js UI meeting usability goals.
- **Scalable Architecture**: Modular design supports multi-format inputs (JSON, XML, UML) and real-time query processing.
- **Impact**: Reduces development time by ~50% via AI-driven service discovery, promoting sustainable software reuse.

- **Service Composition**
- **Multilingual Chatbot**
- **Audio Input Integration**

References

- Elgedawy, I. (2015). USTA: An aspect-oriented knowledge management framework for reusable assets discovery. *The Arabian Journal for Science and Engineering*.
- Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*.
- LangChain Documentation. (2024).
- Hugging Face. (2024). *Transformers Documentation*.
- Facebook AI Research. (2024). *FAISS: A library for efficient similarity search*.

DEMO

Thank you

Questions?