A Conversational AI Agent for FIB Academic Information: Hierarchical Design and LLM-as-Judge Evaluation

Ákos Schneider
akos.schneider@estudiantat.upc.edu

Ignasi Cervero
ignasi.cervero@estudiantat.upc.edu

Human Language Engineering
Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona

January 6, 2026

Abstract

This project presents a conversational AI agent for the Facultat d'Informàtica de Barcelona (FIB) that provides natural language access to university academic information. The agent integrates with the official FIB API to retrieve course catalogs, exam schedules, professor contacts, and user-specific data through OAuth authentication. We implement a hierarchical agent architecture using the deepagents library [1], where a root agent handles user context and private tools while delegating public API queries to a specialized subagent. The system supports multiple Gemini models and includes an internet search fallback for queries outside the FIB domain. We additionally evaluate local models (Qwen 2.5 7B, Llama 3.2 3B) to explore privacy-preserving deployment options.

To systematically evaluate agent performance, we developed an LLM-as-judge framework with seven custom metrics: relevance, helpfulness, conciseness, structure, tone, error handling, and tool appropriateness. Evaluation on a curated dataset of 45 questions across 10 categories demonstrates strong performance, with the agent achieving 0.90 relevance and 0.89 helpfulness scores. We also provide a Model Context Protocol (MCP) server for tool interoperability with other AI assistants.

The contributions of this work are fourfold. First, we present an open-source FIB API agent featuring a hierarchical architecture. Second, we introduce a university-domain evaluation dataset annotated with complexity levels. Third, we establish a reusable LLM-as-judge evaluation framework complete with detailed rubrics. Finally, we document prompt engineering patterns designed to handle ambiguous queries, date reasoning, and course disambiguation in educational contexts.

The source code is available at https://github.com/akossch0/upc-fib-agent.

Contents

1 Introduction

Natural language interfaces have significantly transformed how users interact with information systems. In educational contexts, students frequently need quick access to academic data, such as course offerings, exam schedules, professor contacts, and enrollment status, which is typically scattered across multiple university systems. Traditional navigation through web portals often requires familiarity with their structure and can be time-consuming, especially for simple queries that could be more efficiently handled through direct conversation.

This project presents a conversational AI agent for the Facultat d'Informàtica de Barcelona (FIB) at Universitat Politècnica de Catalunya (UPC). The agent leverages large language models (LLMs) and the official FIB API to provide natural language access to university information, enabling students to ask questions in plain language rather than navigating complex interfaces.

1.1 Problem Statement

Students at FIB face several challenges when seeking academic information. First, academic data is often fragmented across multiple systems, including the FIB API, Racó (the student portal), and the official website, each featuring different interfaces and authentication requirements. Second, navigation complexity can be a barrier, as finding specific information like exam rooms, professor emails, or course prerequisites often requires multiple interactions and prior knowledge of the information architecture. Furthermore, student queries frequently contain implicit context, such as asking "When is my next exam?" which traditional search engines cannot interpret without access to the user's specific enrollment data. Finally, the need for real-time information is critical, particularly during exam periods when immediate access to schedule changes or announcements is essential. These challenges highlight the need for an intelligent interface capable of understanding natural language, integrating diverse data sources, and personalizing responses based on user context.

1.2 Objectives

The primary objectives of this project center on building a comprehensive conversational agent. We aim to develop a system that understands natural language queries regarding FIB academic information, including courses, exams, professors, schedules, and news. A key technical goal is to integrate with the FIB API to access both public endpoints, such as the course catalog and faculty directory, and private OAuth-protected endpoints that provide user-specific data like enrolled courses and personal schedules. To manage this complexity, we intend to implement a hierarchical agent architecture that delegates specialized queries to subagents, allowing for modular tool organization and focused system prompts. Additionally, we seek to create a rigorous evaluation framework using the LLM-as-judge methodology to systematically assess response quality across dimensions such as relevance, helpfulness, and appropriate tool usage. Finally, we aim to provide tool interoperability through a Model Context Protocol (MCP) server, thereby enabling other AI assistants to access and utilize FIB API tools.

1.3 Scope

This project focuses specifically on information retrieval from the FIB API. The system capabilities include course catalog search with detailed information, exam schedule queries by course or date range, professor and faculty search, and retrieval of FIB news and classroom listings. Crucially, the agent supports user-specific data access via OAuth, allowing students to query their profiles, enrolled courses, personal schedules, and notices. For information not available within the FIB API, the system includes an external web search fallback. Conversely, certain functionalities are explicitly excluded from the current scope. These include administrative actions such as course enrollment or grade submission, multi-turn conversation memory where queries maintain state across interactions, and integration with non-FIB university systems.

2 Methodology

This section describes the theoretical foundations, design approach, and technologies underlying the FIB conversational agent.

2.1 Background

2.1.1 Conversational AI Agents

Conversational AI has evolved significantly from rule-based chatbots to sophisticated agents powered by Large Language Models (LLMs) capable of reasoning, planning, and taking actions [2]. Modern agents combine the extensive natural language understanding of LLMs with the ability to interact with external tools and APIs [3]. The key distinction between a standard LLM chatbot and an agent is this capacity for tool use; agents can call external functions to retrieve information or perform actions, subsequently incorporating these results into their responses. This paradigm, known as tool-augmented language models [4], enables LLMs to overcome the limitations of their static training data by accessing real-time information [5].

2.1.2 ReAct: Reasoning and Acting

The ReAct paradigm [3] proposes a synergistic approach where reasoning traces, or "thoughts," are interleaved with actions. Building upon chain-of-thought prompting [6], which elicits step-by-step reasoning in LLMs, ReAct extends this by grounding thoughts in concrete actions. The agent follows a continuous loop consisting of a thought, where it reasons about the current state and decides on a course of action; an action, which involves calling a tool or API; and an observation, where it processes the tool's response. This cyclical approach has proven effective for knowledge-intensive tasks requiring multi-step reasoning. Our implementation builds on these principles through the deepagents library, a hierarchical agent harness that structures this interaction.

2.1.3 Agent Harnesses and deepagents

As LLM agents have evolved from simple tool-calling patterns to complex multi-step workflows, the need for standardized agent harnesses has emerged [7]. These frameworks provide common infrastructure for planning, context management, and hierarchical task delegation. The **deepagents** library [1] addresses these needs by offering hierarchical subagent spawning to delegate subtasks to specialized agents with isolated contexts. It provides planning tools to decompose complex tasks into trackable steps, manages context to prevent window overflow through file system abstraction, and leverages LangGraph [8] for stateful graph execution and checkpointing. Our implementation utilizes this library to create a root agent equipped with private tools and a subagent dedicated to public FIB API queries, enabling focused system prompts and establishing clean security boundaries.

2.1.4 LLM-as-Judge Evaluation

Evaluating open-ended language generation presents challenges because traditional metrics like BLEU or ROUGE correlate poorly with human judgments of quality [9]. The LLM-as-judge approach addresses this by using a capable language model as an evaluator, prompted with specific criteria and rubrics. G-Eval [9] introduced a framework where the LLM judge follows structured evaluation steps to produce scores, while MT-Bench [10] demonstrated that strong LLMs can serve as scalable and explainable evaluators for multi-turn conversations. These methods achieve higher correlation with human judgments than reference-based metrics, particularly for dimensions like helpfulness and coherence that require semantic understanding.

2.1.5 Model Context Protocol

The Model Context Protocol (MCP) [11] is an open standard designed to connect AI assistants to external data sources and tools. It defines a client-server architecture where MCP Servers expose tools, resources, and prompts, and MCP Clients, such as AI applications, discover and invoke these capabilities. By implementing an MCP server, our FIB tools become accessible to any MCP-compatible client, thereby promoting interoperability across AI ecosystems.

2.2 Approach

2.2.1 Hierarchical Agent Design

Rather than employing a flat architecture with all tools available at a single level, we implement a hierarchical design comprising a root agent and a specialized subagent. The Root Agent handles user interaction, private authenticated tools, and internet search, and is responsible for task delegation. It operates with a comprehensive system prompt designed to handle implicit context, date reasoning, and ambiguous queries. The Public FIB Subagent specializes in public FIB API queries, such as those for courses, exams, professors, news, and classrooms. It utilizes a focused system prompt with specific tool selection guidelines and data presentation rules.

This separation of concerns provides several benefits. It allows for focused prompts, where each agent has a specialized system prompt, reducing cognitive load and improving instruction following. It creates a security boundary where private tools requiring OAuth remain at the root level, while public tools are delegated. Furthermore, it enhances modularity, as the subagent can be tested and refined independently. The root agent delegates to the subagent using a task tool call, which is an internal implementation detail hidden from users.

2.2.2 Tool Abstraction

Each FIB API endpoint is wrapped as a typed Python function. These functions include type-annotated parameters with default values, docstrings describing their purpose and argument semantics, Pydantic validation of API responses, and consistent error handling via decorators. This abstraction allows the LLM to understand tool capabilities through function signatures and docstrings, while Pydantic models ensure type safety and facilitate IDE support.

2.2.3 System Prompt Engineering

The system prompts serve as the primary mechanism for guiding agent behavior, with several key design decisions shaping their structure. For personalized queries, the agent is instructed to first establish user context by checking enrolled courses before searching public data. Common patterns implying implicit context, such as "my exam" or "tomorrow," are documented with specific resolution strategies. To handle date-relative queries without internet search, explicit weekday-to-integer mappings are provided. The prompt also includes a disambiguation strategy for ambiguous queries; for instance, if "Machine Learning" matches multiple courses, the agent is instructed to prioritize likely matches and present a few options rather than listing all. Finally, an output quality checklist ensures the agent verifies that there is no redundancy, no vague hedging, consistent formatting, and that all requested items are addressed before responding.

2.3 Tools and Technologies

The project employs a suite of technologies to achieve its goals. LangGraph serves as the underlying stateful graph execution engine with checkpointing capabilities. On top of this, deepagents functions as the hierarchical agent harness, providing subagent spawning, planning tools, and context management. The LLM backbone consists of Google Vertex AI models, specifically Gemini 2.5 Flash for speed and Gemini 2.5 Pro for complex queries. Data validation and type-safe API response parsing are handled by Pydantic. For evaluation, the project uses deepeval (GEval), an LLM-as-judge framework with customizable metrics. Tavily provides an external web search API for information beyond the FIB scope, and the MCP Protocol ensures tool server standardization for AI interoperability.

2.4 Data Sources

2.4.1 FIB API

The FIB API (`https://api.fib.upc.edu/v2`) provides programmatic access to university data and is the core data source for the agent. We integrate with a set of public endpoints that require a client ID, including the course catalog, exam schedules, faculty directory, classroom listings, academic terms, and news announcements. Additionally, we utilize private endpoints that require an OAuth2 bearer token to access user-specific information such as the user profile, enrolled courses, personal schedule, and course notices.

2.4.2 Evaluation Dataset

To ensure rigorous testing, we created a manually curated evaluation dataset consisting of 45 questions across 10 categories. These questions cover varying complexity levels. Simple queries involve direct factual questions like "How many credits is IA?". Multi-step queries require multiple tool calls, such as "Compare PRO1 and PRO2". Contextual queries depend on user context, for example, "When is my next exam?". Finally, ambiguous queries are vague and require clarification, such as "Help me" or "Is it hard?". Each question in the dataset is annotated with the expected tools, authentication requirements, and notes for evaluation.

3 Implementation

This section details the technical implementation of the FIB conversational agent, including its architecture, core components, and the solutions to key challenges encountered during development.

3.1 Architecture

The overall system architecture follows a hierarchical design where a root agent orchestrates user interaction and delegates specific tasks. The root agent manages user context, including authentication and profiles, and has direct access to private tools and external internet search via Tavily. When a query requires public university information, the root agent delegates this responsibility to a specialized public subagent via a specific task call. This public subagent then interacts with the public tools of the FIB API. This architecture effectively separates concerns: the root agent owns the user context and overall orchestration, the public subagent specializes in unauthenticated FIB API queries, and external search remains available at the root level for queries falling outside the university's scope.

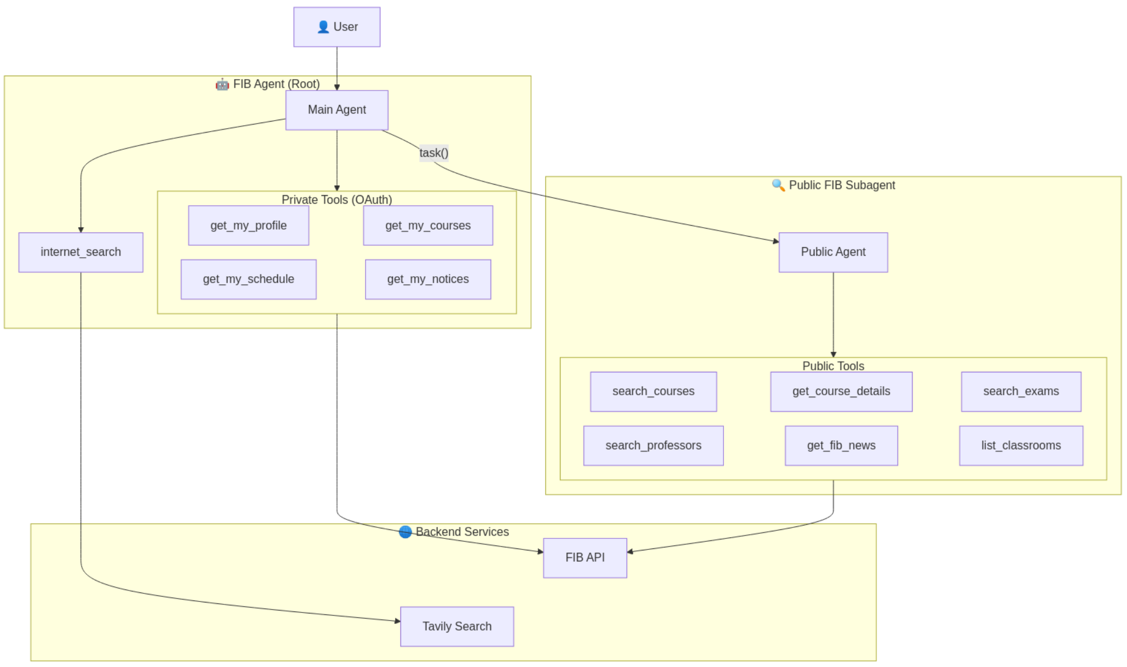


Figure 1: Hierarchical agent architecture showing the root agent with private tools and the public FIB subagent with public tools.

3.2 Core Components

3.2.1 FIB API Client

The `FIBAPIClient` class handles all HTTP communication with the FIB API. It is implemented using a singleton pattern, ensuring that a single client instance is reused across all tool calls to maintain connection pooling and authentication state. The client includes robust pagination handling, where the `_get_paginated()` method automatically follows pagination links to retrieve complete result sets. Additionally, a comprehensive error hierarchy with custom exceptions allows for precise error handling across the application, distinguishing between authentication issues, not-found errors, and rate limits.

3.2.2 Data Models

All responses from the FIB API are validated through Pydantic models to ensure data integrity. These models define the structure of various data types such as courses, exams, professors, classrooms, and news items, as well as user-specific data like profiles and schedules. Fields are explicitly typed and described, and properties are computed where necessary, such as determining if a course is active. This approach ensures that the data flowing through the system is well-defined and reliable.

3.2.3 Tool Layer

The tool layer wraps the API client methods into Python functions accessible to the agent. Each tool is decorated to handle API errors consistently. Public tools cover functionalities like searching courses, retrieving course details, finding exam schedules, looking up professors, and listing classrooms. Private tools provide access to the user's profile, enrolled courses, personal schedule, and notices. This abstraction allows the agent to interact with the API using high-level semantic actions rather than low-level HTTP requests.

3.2.4 Agent Creation

The agent is instantiated using a factory function that configures the authentication and selects the appropriate model strategy. The design supports different backend models, such as various versions of Gemini or local open-source models, through a flexible strategy pattern. The creation process assembles the toolset, including private tools and optional internet search, and initializes the hierarchical structure with the public subagent. This configuration ensures that the agent is correctly set up with the necessary capabilities and system prompts before handling user queries.

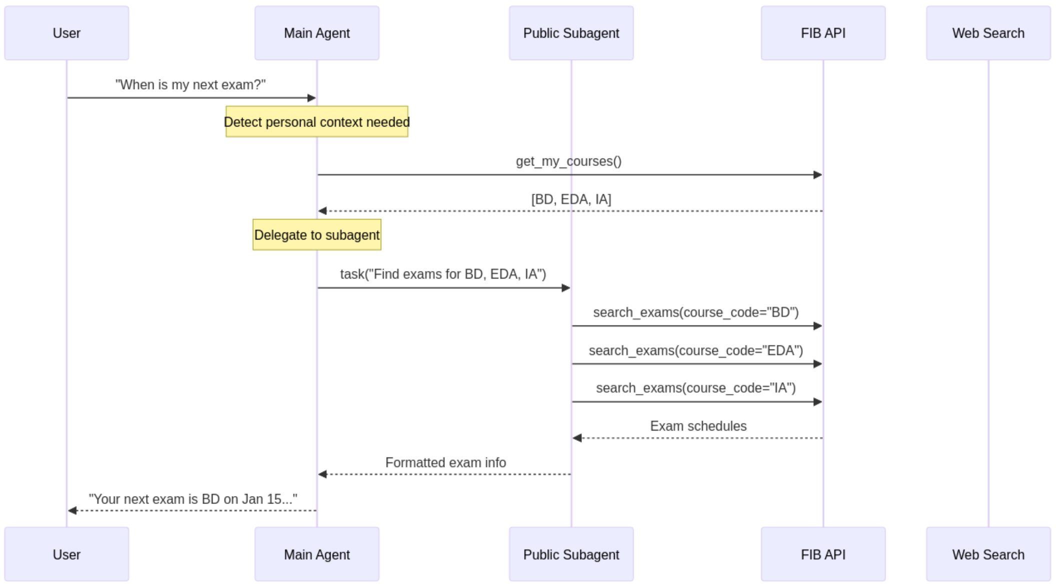


Figure 2: Agent execution flow showing the ReAct loop: the agent receives a query, reasons about it, selects and executes tools, observes results, and iterates until a final response is generated.

3.3 System Prompts

System prompts are central to guiding the agent's behavior and ensuring high-quality responses. The Root Agent Prompt includes instructions for planning multi-step queries, detecting implicit context (such as interpreting "my exam" as a request to check enrolled courses), and handling dates with explicit weekday mappings. It also defines strategies for disambiguating vague course names and includes a response quality checklist. The Public Subagent Prompt focuses on tool selection guidelines, specifying when to use specific search tools versus listing tools, and provides rules for data presentation, such as using tables for multi-item results and ensuring output quality by avoiding speculation.

3.4 MCP Server

To expose the FIB tools to external clients, an MCP server is implemented following the Model Context Protocol specification. This server registers the available tools, such as course search, exam lookup, and personal schedule retrieval, and defines their input schemas using JSON Schema. It handles incoming tool calls by dispatching them to the appropriate underlying functions and returning structured results.

This design enables interoperability with MCP-compatible AI assistants. For example, Figure 3 demonstrates the integration with Claude Desktop, where the assistant uses the FIB MCP tools to answer a user's question about upcoming exams. The assistant automatically invokes `get_my_courses` to retrieve enrolled courses, `get_current_term` to determine the active academic period, and then calls `search_exams` for each course to compile a comprehensive exam schedule. This demonstrates how the modular tool design allows the same functionality to be accessed through different interfaces, whether our custom LangGraph agent or third-party AI assistants, without code duplication.

3.5 Challenges and Solutions

3.5.1 Ambiguous Course Queries

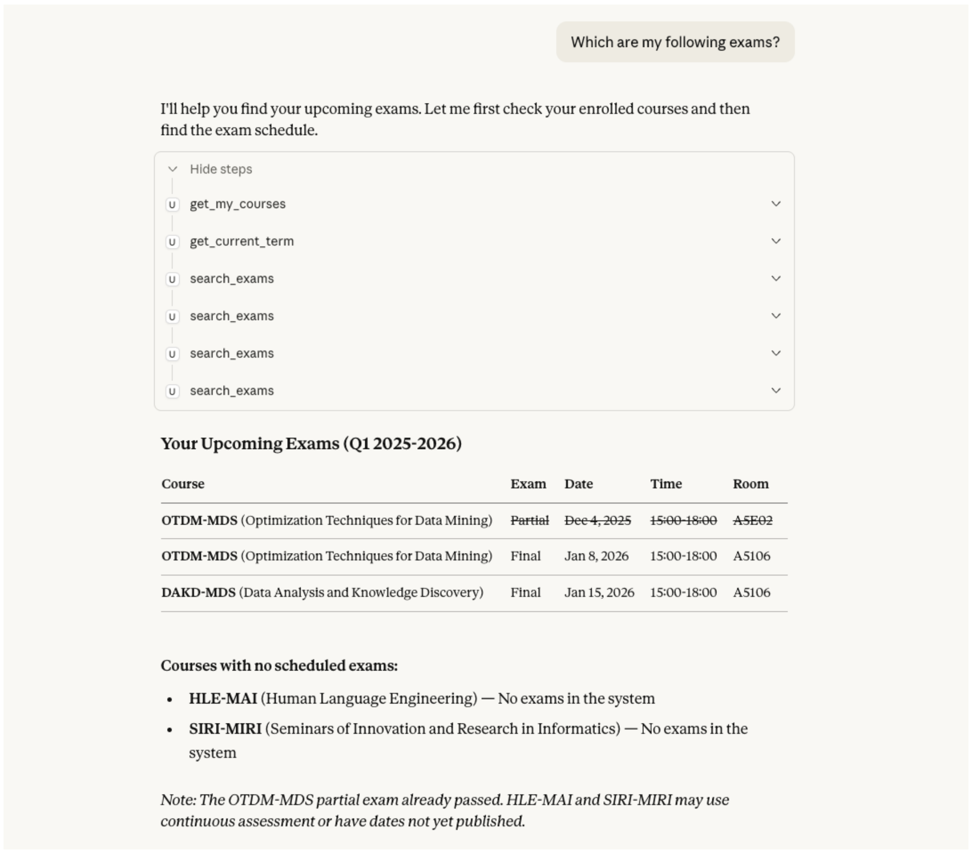
One significant challenge was handling ambiguous queries, such as a search for "Machine Learning," which matches multiple courses across different programs like the bachelor's and master's degrees. To address this, the system prompt instructs the agent to prioritize courses from the user's enrolled study plan if authenticated. For general queries, it prioritizes bachelor's courses and presents the most likely matches with their codes and programs, asking for clarification only after providing useful initial information.

3.5.2 Date-Relative Queries

Queries involving relative dates, such as "What do I have tomorrow?", posed a challenge as they require knowledge of the current date and its mapping to a weekday. The solution involved providing the agent with the current date in its context and explicitly documenting the day-to-integer mapping (e.g., Monday=1). This allows the agent to call the schedule tool with the correct parameters without needing to perform an internet search to resolve the date.

3.5.3 OAuth Token Management

Accessing private endpoints requires OAuth2 tokens that expire and need refreshing. The `FIBOAuthClient` class was implemented to handle the interactive authorization flow, including a local callback server. It persists tokens to a file and automatically refreshes


Which are my following exams?

I'll help you find your upcoming exams. Let me first check your enrolled courses and then find the exam schedule.

Your Upcoming Exams (Q1 2025-2026)

Course	Exam	Date	Time	Room
~~**OTDM-MDS** (Optimization Techniques for Data Mining)~~	~~Partial~~	~~Dec 4, 2025~~	~~15:00-18:00~~	~~A5E02~~
OTDM-MDS (Optimization Techniques for Data Mining)	Final	Jan 8, 2026	15:00-18:00	A5106
DAKD-MDS (Data Analysis and Knowledge Discovery)	Final	Jan 15, 2026	15:00-18:00	A5106

Courses with no scheduled exams:

- **HLE-MAI** (Human Language Engineering) — No exams in the system
- **SIRI-MIRI** (Seminars of Innovation and Research in Informatics) — No exams in the system

Note: The OTDM-MDS partial exam already passed. HLE-MAI and SIRI-MIRI may use continuous assessment or have dates not yet published.

Figure 3: Claude Desktop using the FIB MCP server to answer an exam schedule query. The tool calls panel shows the automatic invocation of `get_my_courses`, `get_current_term`, and multiple `search_exams` calls.

them when they expire. If authentication is unavailable, the system gracefully falls back to public-only functionality.

3.5.4 API Pagination

List endpoints in the API return paginated results, which complicates data retrieval. A generic pagination method was implemented in the client to handle this transparently. This method iterates through the pages by following the `next` links provided in the response, aggregating all results into a single list before returning them to the caller.

3.6 Project Structure

The codebase is organized into a modular structure. The source code directory contains the main agent logic, the API client with its pagination and error handling, the OAuth authorization flow, and the MCP server implementation. Data models are defined separately using Pydantic, and tools are grouped by functionality. Scripts are provided for running inference and evaluation metrics, while the evaluation directory contains the dataset, results, and visualization tools.

4 Results

This section presents the evaluation methodology, quantitative results, and qualitative analysis of the FIB agent's performance.

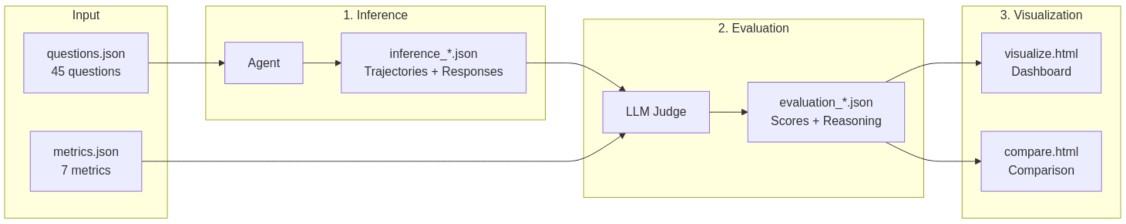


Figure 4: Evaluation pipeline: questions are fed to the agent, responses are collected with tool call trajectories, and the LLM judge scores each response across seven metrics.

4.1 Experimental Setup

4.1.1 Models Tested

We evaluated two Gemini models as the agent backbone. Gemini 2.5 Flash was tested as a fast, cost-effective model optimized for speed, while Gemini 2.5 Pro was evaluated for its higher capacity for complex reasoning. Both models were accessed through Google Vertex AI using application default credentials.

4.1.2 Evaluation Judge

For the LLM-as-judge evaluation, we employed Gemini 2.5 Flash Lite as the evaluator model. This lighter model provided sufficient capability for rubric-based scoring while maintaining cost efficiency across the substantial evaluation set, which involved 315 evaluations per run (45 questions multiplied by 7 metrics).

4.1.3 Evaluation Dataset

The dataset comprises 45 manually curated questions distributed across 10 categories, as detailed in Table 1.

Table 1: Question distribution by category

Category	**Count**	**Description**
courses	13	Course information queries
exams	5	Exam schedules and rooms
professors	4	Faculty information
news	3	FIB announcements
academic	2	Terms and semesters
classrooms	2	Room information
personal	4	User-specific (requires OAuth)
multi_tool	5	Complex multi-aspect queries
ambiguous	4	Vague queries needing clarification
web_search	3	External information queries

The questions span four complexity levels. Simple questions are direct factual queries requiring a single tool. Multi-step questions require multiple tool calls or comparisons.

Contextual questions depend on user context or implicit information. Finally, ambiguous questions are vague or incomplete queries that require the agent to ask for clarification.

4.1.4 Evaluation Metrics

We defined seven evaluation metrics, each accompanied by a detailed persona, rubric describing correct and incorrect behaviors, evaluation steps, and reminders for the LLM judge. Relevance measures whether the response addresses the user's question. Helpfulness assesses if the response is actionable and useful. Conciseness checks if the response is appropriately brief without losing information. Structure evaluates the organization and formatting of the response. Tone ensures the response is professional and appropriate for an academic context. Error Handling looks for graceful management of missing data or failures. Finally, Tool Appropriateness determines if the agent selected the right tools for the task. Each metric produces a score from 0.0 to 1.0 along with a textual reasoning.

4.2 Quantitative Results

Table 2 presents the average scores across all 45 questions for both models.

Table 2: Evaluation results comparing Gemini Flash and Pro models

Metric	Flash	Pro	Target
Relevance	0.90	0.89	>0.85
Helpfulness	0.89	0.93	>0.85
Conciseness	0.85	0.83	>0.80
Structure	0.77	0.80	>0.75
Tone	0.85	0.85	>0.80
Error Handling	0.51	0.43	>0.70
Tool Appropriateness	0.77	0.78	>0.80
Average	**0.79**	**0.79**	—

Several key observations emerge from these results. Both models achieve the target threshold for relevance and helpfulness, which are the two most critical metrics for user satisfaction. Conciseness, structure, and tone also meet their respective targets, indicating that the responses are well-formatted and professional. However, error handling significantly underperforms the target, a result that analysis suggests is partly due to overly strict rubric interpretation. Tool appropriateness is slightly below target, which is affected by the indirect tool calls inherent in subagent delegation.

4.2.1 Performance by Category

Table 3 shows relevance scores broken down by question category.

Table 3: Relevance scores by question category (Flash model)

Category	**Avg. Relevance**
courses	0.95
exams	0.92
professors	0.88
classrooms	0.90
academic	0.95
news	0.87
personal	0.85
multi_tool	0.82
ambiguous	0.93
web_search	0.90

Simple factual queries regarding courses, exams, and academic terms achieve the highest scores. Ambiguous queries also score well because the agent appropriately asks for clarification, behavior that the rubric explicitly scores as relevant for vague queries.

4.3 Qualitative Analysis

4.3.1 Successful Patterns

The agent demonstrates strong performance in handling simple course queries. For example, when asked "How many credits is the IA course?", the agent correctly identified the course and responded concisely with "The IA course (Artificial Intelligence) has 6 credits." This response achieved perfect scores for relevance, helpfulness, and conciseness.

The agent also handles ambiguous queries effectively. When presented with a vague request like "Help me", the agent responded by offering a helpful menu of capabilities, asking the user to specify their need regarding course information, exam schedules, professor contacts, or personal data. This approach of guiding the user rather than guessing resulted in high relevance and helpfulness scores.

4.3.2 Challenging Cases

Multi-tool complexity presents a greater challenge. In response to "Tell me everything about the EDA course", the agent provided a comprehensive answer including course details, professors, and exam schedules. However, the structure score suffered due to inconsistent formatting, and tool appropriateness was marked down because the agent made multiple search calls that could have been consolidated.

Error handling verbosity also proved to be an issue. When unable to find a course code, the agent tended to offer apologetic and speculative explanations rather than a direct statement. For instance, responding to a query about a non-existent course "XYZ" with a lengthy apology and potential reasons for the failure resulted in a low error handling score, as the rubric prefers more direct communication of the error.

4.4 Discussion

4.4.1 Error Handling Metric Analysis

The error handling metric significantly underperforms expectations. Upon analysis, we identified two contributing factors. First, there were false negatives where responses that correctly answered questions received low error handling scores because the judge interpreted "error handling" as requiring explicit acknowledgment of limitations even when none were encountered. Second, when errors did occur, the agent tended toward verbose apologies with speculative explanations, which the rubric penalizes. This suggests that both prompt refinement to reduce verbosity on errors and rubric clarification are needed.

4.4.2 Tool Appropriateness and Subagent Delegation

Tool appropriateness scores are affected by the hierarchical architecture. When the root agent delegates to the subagent via the task tool, the evaluation only sees "task" as the tool call, not the underlying tools called by the subagent. This is a limitation of the current evaluation framework that could be addressed by flattening tool call trajectories across agent boundaries or by scoring delegation as appropriate when the subagent handles the query correctly.

4.4.3 Model Comparison

Gemini Flash and Pro achieved nearly identical overall scores. The Pro model showed marginal advantages in helpfulness and structure, providing slightly more comprehensive and better-formatted responses. Conversely, Flash showed advantages in conciseness and error handling, offering less verbose responses. Given the minimal quality difference and Flash's faster response times and lower cost, Gemini 2.5 Flash is recommended as the default model for this application.

4.5 Local Model Experiments

To explore deployment options without cloud dependencies, we evaluated two local models using llama-cpp-python: Qwen 2.5 7B Instruct and Llama 3.2 3B Instruct, both in Q4_K_M quantization. This required implementing a custom `ChatLlamaCppTools` wrapper that formats tool definitions using Qwen's XML-based tool calling syntax, as the standard LangChain integration does not reliably generate tool calls with local models.

4.5.1 Local Model Results

Table 4 compares the local models against the cloud-based Gemini models. We tested each local model at two temperature settings (0.1 and 0.5) to assess the impact of sampling randomness on tool-calling reliability.

Table 4: Evaluation results comparing cloud and local models

Model	Temp	Relevance	Helpfulness	Tool Approp.
Gemini 2.5 Flash	–	0.90	0.89	0.77
Gemini 2.5 Pro	–	0.89	0.93	0.78
Qwen 2.5 7B	0.1	0.74	0.80	0.42
Qwen 2.5 7B	0.5	0.74	0.80	0.43
Llama 3.2 3B	0.1	0.22	0.29	0.10
Llama 3.2 3B	0.5	0.22	0.32	0.09

4.5.2 Analysis

The results reveal a significant capability gap between model sizes for agentic tasks. Qwen 2.5 7B achieves approximately 80% of Gemini's relevance and helpfulness scores, demonstrating that mid-sized local models can handle straightforward queries. However, its tool appropriateness score (0.42) is notably lower than Gemini's (0.77), indicating difficulties with complex multi-step tool orchestration.

Llama 3.2 3B performs poorly across all metrics, with relevance below 0.25. Analysis of the inference logs reveals that the 3B model frequently fails to generate properly formatted tool calls, often responding with generic text instead of invoking the appropriate API tools. This suggests that reliable agentic behavior requires models above a certain capability threshold.

Interestingly, temperature has negligible impact on performance for either local model. This contrasts with common intuition that lower temperatures improve tool-calling reliability, and suggests that the primary bottleneck is model capability rather than sampling strategy.

These findings support using Gemini for production deployments while demonstrating that local models like Qwen 2.5 7B could serve as cost-effective alternatives for simpler use cases or privacy-sensitive environments. The modular model strategy pattern in our implementation enables easy swapping between cloud and local backends.

5 Conclusions

This section summarizes the project's achievements, acknowledges limitations, and outlines directions for future development.

5.1 Summary

This project successfully developed a conversational AI agent for the Facultat d'Informàtica de Barcelona (FIB) that provides natural language access to university academic information. The agent integrates with the official FIB API to access course catalogs, exam schedules, professor directories, news, and user-specific data through OAuth authentication. It implements a hierarchical agent architecture using LangGraph and the deepagents library, featuring a root agent that handles user context and a specialized subagent for public API queries. The system achieves strong performance on relevance and helpfulness metrics, the two dimensions most critical for user satisfaction. Furthermore, it includes a comprehensive evaluation framework using the LLM-as-judge methodology with custom metrics and curated test questions. Finally, the project provides tool interoperability through an MCP server, enabling other AI assistants to access FIB data. The evaluation demonstrates that modern LLM agents, when properly configured with domain-specific tools and carefully engineered prompts, can effectively serve as natural language interfaces to structured APIs.

5.2 Contributions

The main contributions of this project are fourfold. First, we provide an open-source FIB API Agent, a fully functional conversational agent with a hierarchical architecture available for the FIB student community. Second, we introduce an Evaluation Dataset, a curated set of 45 questions across 10 categories and 4 complexity levels, annotated with expected tools and authentication requirements to enable systematic evaluation. Third, we established a reusable LLM-as-Judge Framework with 7 custom metrics, detailed rubrics, and visualization dashboards. Fourth, we implemented an MCP Server that exposes FIB tools, demonstrating AI tool interoperability standards in an educational context. Additionally, we have documented prompt engineering patterns for handling ambiguous queries, date reasoning, course disambiguation, and user context detection in university chatbot domains.

5.3 Limitations

Several limitations should be acknowledged. The private endpoints require manual OAuth token acquisition, which adds complexity and limits accessibility for casual users; a production deployment would require a proper OAuth flow with institutional integration. The LLM-as-judge methodology, while using detailed rubrics, inherits potential biases from the evaluator model, meaning perfect alignment with human judgment is not guaranteed. The current implementation treats each query independently without conversational memory, so multi-turn dialogues are not supported. The agent is also domain-specific, tightly coupled to FIB's API structure and terminology, meaning adaptation to other universities would require significant modification. As discussed in the results section, the error handling evaluation metric shows inconsistent scoring due to rubric interpretation issues. finally, the agent makes synchronous API calls, which can result in slower responses for queries requiring multiple tool invocations.

5.4 Future Work

Several directions could extend this work. Integrating Retrieval-Augmented Generation (RAG) with course syllabi, lecture notes, and historical data would enable the agent to answer questions beyond the current API scope. Implementing conversation memory using LangGraph's persistence layer would enable natural follow-up questions and contextual references. Building a user-facing interface, such as a Telegram bot or web chat, with proper authentication flows would make the agent accessible to the broader FIB community. Abstracting the tool layer to support multiple university APIs through a common interface would enable adaptation to other institutions. Extending the system to support agentic capabilities, such as setting reminders for exams or subscribing to announcements, would increase its practical utility. Improving the error handling metric rubric and adding human evaluation baselines would strengthen the evaluation framework. Finally, implementing streaming output would improve perceived latency for complex queries requiring multiple tool calls.

References

[1] LangChain. *deepagents: Hierarchical Agent Library*. 2024. URL: https://github.com/langchain-ai/deepagents (visited on 12/15/2024).

[2] Lei Wang et al. “A Survey on Large Language Model based Autonomous Agents”. In: *Frontiers of Computer Science* 18.6 (2024). URL: https://arxiv.org/abs/2308.11432.

[3] Shunyu Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models”. In: *International Conference on Learning Representations (ICLR)*. 2023. URL: https://arxiv.org/abs/2210.03629.

[4] Timo Schick et al. “Toolformer: Language Models Can Teach Themselves to Use Tools”. In: *Advances in Neural Information Processing Systems* 36 (2023). URL: https://arxiv.org/abs/2302.04761.

[5] LangChain. *LangChain Agents Documentation*. 2024. URL: https://python.langchain.com/docs/concepts/agents/ (visited on 12/15/2024).

[6] Jason Wei et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In: *Advances in Neural Information Processing Systems*. 2022. URL: https://arxiv.org/abs/2201.11903.

[7] Shishir G. Patil et al. “Gorilla: Large Language Model Connected with Massive APIs”. In: *arXiv preprint arXiv:2305.15334* (2023). URL: https://arxiv.org/abs/2305.15334.

[8] LangChain. *LangGraph: Build Stateful, Multi-Actor Applications with LLMs*. 2024. URL: https://langchain-ai.github.io/langgraph/ (visited on 12/15/2024).

[9] Yang Liu et al. “G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2023, pp. 2511–2522. URL: https://arxiv.org/abs/2303.16634.

[10] Lianmin Zheng et al. “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena”. In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023. URL: https://arxiv.org/abs/2306.05685.

[11] Anthropic. *Model Context Protocol*. 2024. URL: https://modelcontextprotocol.io/ (visited on 12/15/2024).

A Appendix

This appendix provides supplementary materials including extended data, configuration details, and usage instructions.

A.1 Extended Data

A.1.1 Evaluation Question Categories

Table 5 provides the complete breakdown of evaluation questions by category with example questions.

Table 5: Full question category breakdown with examples

Category	Count	Example Questions
courses	13	"How many credits is IA?", "What AI courses are available?", "Compare PRO1 and PRO2"
exams	5	"When is the BD final?", "What exams are next week?", "Are there conflicts between EDA and BD?"
professors	4	"Who teaches IA?", "What is Jordi Petit's email?", "Find database professors"
news	3	"What's new at FIB?", "Any upcoming hackathons?", "International opportunities?"
academic	2	"What is the current semester?", "When does next semester start?"
classrooms	2	"How many classrooms?", "Classrooms in building A5?"
personal	4	"When is my next exam?", "What courses am I enrolled in?", "What do I have tomorrow?"
multi_tool	5	"Tell me everything about EDA", "I want to take AI, who teaches it?"
ambiguous	4	"Help me", "The course", "Is it hard?", "What should I do?"
web_search	3	"Job prospects in Barcelona?", "How does FIB compare to other schools?"

A.1.2 Metric Rubrics

Each evaluation metric includes detailed rubrics for the LLM judge. Table 6 summarizes the key correct and incorrect behaviors for each metric.

Table 6: Summary of evaluation metric rubrics

Metric	Correct Behaviors	Incorrect Behaviors
Relevance	Directly answers question; stays on topic; addresses all key elements	Provides unrelated info; ignores key aspects; goes off-topic
Helpfulness	Actionable information; includes links/resources; clear next steps	Vague/generic info; missing details; leaves user confused
Conciseness	No filler; no redundancy; appropriate length for question complexity	Excessive introductions; repeated info; disproportionately long
Structure	Appropriate formatting (lists, tables); logical organization; easy to scan	Wall of text; inconsistent formatting; scattered information
Tone	Professional; friendly; appropriate for academic context	Rude; dismissive; overly casual; condescending
Error Handling	Clear acknowledgment of limitations; factual; suggests alternatives	Fabrication; vague non-answers; excessive apologies
Tool Approp.	Correct tool selection; efficient usage; appropriate parameters	Wrong tools; redundant calls; failure to use needed tools

A.1.3 Complete Tool List

Table 7 lists all available tools with their parameters.

Table 7: Complete tool reference

Tool	Key Parameters	Auth
`search_courses`	query, semester, study_plan, course_type, credits_min/max	Public
`get_course_details`	course_code	Public
`search_exams`	course_code, start_date, end_date, semester, year, exam_type	Public
`get_upcoming_exams`	days_ahead, study_plan	Public
`search_professors`	name, course, department	Public
`get_academic_terms`	—	Public
`get_current_term`	—	Public
`get_fib_news`	limit	Public
`list_classrooms`	building	Public
`get_my_profile`	—	OAuth
`get_my_courses`	include_grades	OAuth
`get_my_schedule`	day	OAuth
`get_my_notices`	course_code	OAuth
`internet_search`	query, max_results, topic	—

A.2 Configuration Details

A.2.1 Environment Variables

The system requires several environment variables for proper operation. Key variables include the FIB API credentials (client ID and secret), LangSmith tracing keys for debugging, Tavily API key for web search capabilities, and Google Cloud project settings for Vertex AI access. These should be defined in a standard environment file.

A.2.2 FIB API Registration

To utilize the system, one must first obtain FIB API credentials. This involves visiting the FIB API registration page, logging in with valid university credentials, and registering a new application. The registration process requires setting the OAuth redirect URI to the local callback address, after which the client ID and secret are generated.

A.2.3 OAuth Authentication

Accessing private endpoints necessitates an OAuth authentication flow. The provided authentication script initiates a local server and opens a browser window for the user to log in via the FIB portal. Upon successful authentication, the resulting tokens are securely stored locally for subsequent use by the agent.

A.2.4 Google Cloud Setup

For the LLM backbone, the system relies on Google Vertex AI. Users must install the Google Cloud CLI, authenticate their session, set the active project, and configure application default credentials to authorize the API calls.

A.3 User Guide

A.3.1 Quick Start

Getting started with the agent involves installing the necessary dependencies using the project's package manager. Once installed, the agent can be tested interactively using LangGraph Studio or invoked programmatically via Python scripts.

A.3.2 Evaluation Pipeline

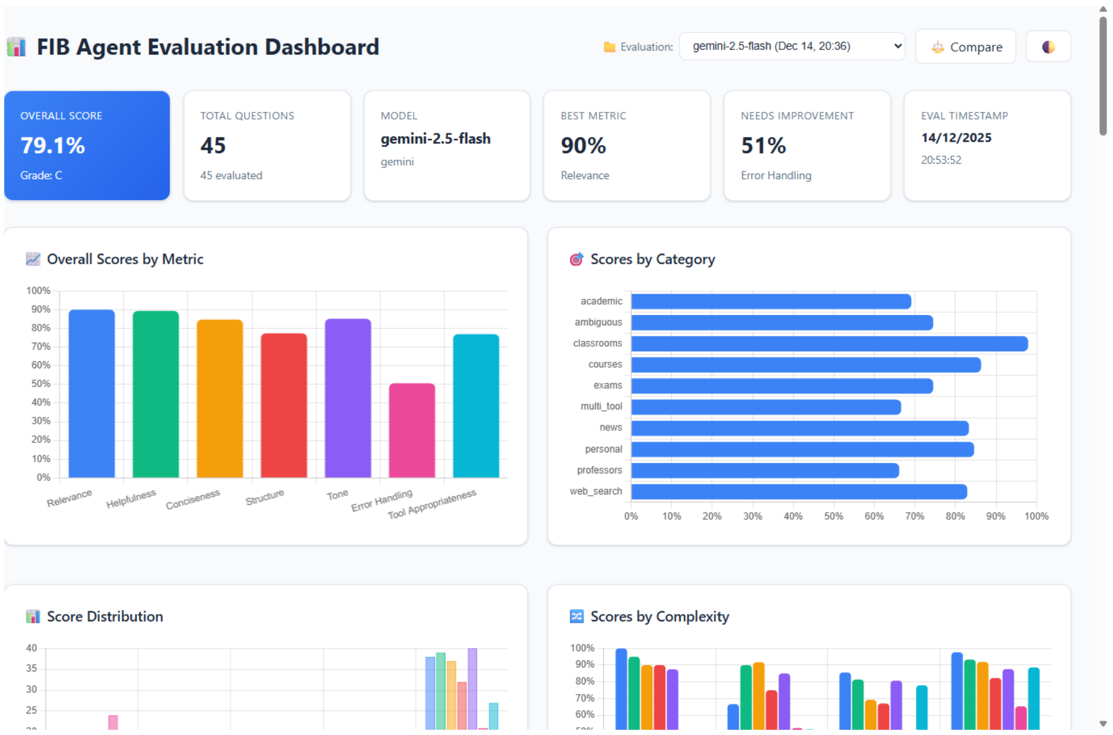
The evaluation pipeline is automated via make commands. Users can run inference on the full question set using a specific model and then trigger the evaluation process on the generated results. The system also includes a visualization dashboard that can be served locally to inspect the evaluation metrics and reasoning.

A.3.3 MCP Server Usage

The project includes a Model Context Protocol server that exposes the agent's tools to other applications. This server can be started via the command line and uses standard input/output for communication, making it compatible with various MCP clients such as Claude Desktop.

A.3.4 Visualization Dashboard

The evaluation results include two HTML dashboards. The first visualizes data for a single evaluation run, offering an overview of metric scores, per-question breakdowns, and tool call trajectories. The second dashboard enables comparison between multiple evaluation runs, allowing for side-by-side metric analysis and tracking of model performance trends. These dashboards operate on static JSON files and do not require a backend server.



FIB Agent Evaluation Dashboard

Evaluation: gemini-2.5-flash (Dec 14, 20:36)

OVERALL SCORE	TOTAL QUESTIONS	MODEL	BEST METRIC	NEEDS IMPROVEMENT	EVAL TIMESTAMP
79.1%	45	gemini-2.5-flash	90%	51%	14/12/2025
Grade: C	45 evaluated	gemini	Relevance	Error Handling	20:53:52

Figure 5: Evaluation dashboard showing metric scores, per-question results, and filtering capabilities for detailed analysis.

A.4 Sample Inference Trajectories

A.4.1 Simple Query Trajectory

For a simple query such as asking about the credits for a specific course, the interaction typically follows a linear path. The Root Agent receives the user's question and delegates the task to the Subagent. The Subagent then calls the course search tool with the appropriate query. Upon receiving the course details, including the credit value, the Subagent formulates a response, which the Root Agent then relays back to the user.

A.4.2 Personal Query Trajectory

Personal queries, such as asking about an upcoming exam, require a more complex flow involving authentication. The Root Agent first calls the private tool to retrieve the user's enrolled courses. With this context, it then instructs the Subagent to find upcoming exams specifically for those courses. The Subagent queries the exam schedule tool and returns the relevant dates and locations. Finally, the Root Agent synthesizes this information into a personalized response for the user.