Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

An LLM-agent-based Dynamic Planning and Monitoring System in the Educational Domain

Pau Blanco
Universitat de Barcelona
Inmaculada Rodríguez
inmarodriguez@ub.edu
Universitat de Barcelona
Anna Puig
Universitat de Barcelona

Research Article

Keywords: intelligent project management system, educational chatbot, LLM-agent-based System

Posted Date: July 2nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10206552/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

An LLM-agent-based Dynamic Planning and Monitoring System in the Educational Domain

Pau Blanco[1], Inmaculada Rodríguez[1,2*†], Anna Puig[1,3†]

[1*]Matemàtiques i Informàtica, Universitat de Barcelona, Gran via, 585, Barcelona, 08007, Spain.
[2]UBICS research institute, Universitat de Barcelona.
[3]IMUB research institute, Universitat de Barcelona.

*Corresponding author(s). E-mail(s): inmarodriguez@ub.edu;
Contributing authors: pablo.blanco@estudiantat.upc.edu;
annapuig@ub.edu;
†These authors contributed equally to this work.

Abstract

Large Language Models (LLMs) are enabling conversational agents capable of supporting increasingly complex workflows beyond traditional question-answering. In education, collaborative projects require continuous planning, coordination, and monitoring, yet existing educational chatbots mainly focus on tutoring or administrative support. This paper presents vProjTutor, an LLM-agent-based project management system that assists student teams throughout the complete project lifecycle. The proposed architecture integrates specialized LLM agents with external project-management tools, including Telegram, Trello, and GitHub, allowing users to configure projects, organize tasks, generate Sprint plans, monitor progress, and interpret repository activity through natural language interactions. A user study involving ten participants evaluated usability, task completion, and user satisfaction. Participants achieved an overall task completion rate of 93.3%, and the system obtained an overall BUS usability score of approximately 85%. The results suggest that LLM-agent-based architectures constitute a promising approach for supporting collaborative projects in educational environments.

Keywords: intelligent project management system, educational chatbot, LLM-agent-based System

1 Introduction

Recent advances in Large Language Models (LLMs) have expanded the capabilities of conversational agents, enabling them to move beyond traditional question-answering systems and act as reasoning, task orchestration, and decision-support components. These developments open the possibility of creating educational assistants capable not only of interacting naturally with students but also of supporting the management and execution of collaborative projects [1]. Rather than functioning as isolated conversational interfaces, such systems can become integrated participants in educational workflows, coordinating tasks, and monitoring progress throughout a project.

Building upon our previous work on the integration of a Hybrid Adaptive Planning Module (HAPM) within a gamified educational chatbot [2], this article presents an extended architecture that explores the role of LLMs as central planning and monitoring agents. While the previous work focused primarily on the design and experimental evaluation of a hybrid planning framework, together with a Rasa-based natural language understanding (NLU) and dialogue management system [3], the present study replaces this traditional conversational architecture with an LLM-agent-based approach. The proposed architecture extends both language understanding and project orchestration capabilities, enabling operation in real-world collaborative software development scenarios integrated with conventional project management tools.

Specifically, the system is integrated with GitHub, Trello, and Telegram, enabling continuous monitoring of project execution and adaptive intervention throughout the project lifecycle. Such capabilities are relevant for sustainable educational settings, where instructors often need to supe rvise multiple collaborative teams efficiently. Moreover, in this paper we present the evaluation of the proposed approach with students in the context of academic projects. This setting provided an opportunity to investigate our main research question "to what extent can LLM-agent-based chatbots effectively support planning and monitoring responsibilities in collaborative academic projects?" Preliminary results are encouraging since the system achieved a task completion rate close to 100%, and the perceived quality of users reached an average 87%.

The main contributions of this work are: (i) an LLM-agent-based architecture for collaborative project management; (ii) the integration of conversational agents with conventional project-management tools; (iii) a preliminary usability evaluation with end users. The proposed system also contributes to sustainable education by complementing instructors in the management of collaborative projects throughout their lifecycle. By automating routine supervision and coordination tasks, it enables educators to supervise multiple teams more effectively while maintaining continuous guidance.

2 Related work

Initial educational chatbots were primarily based on rule-based frameworks [4], relying on predefined rules and keyword matching. The emergence of Large Language

Models (LLMs), particularly ChatGPT, has shifted this paradigm toward generative conversational systems with significantly greater reasoning and interaction capabilities.

LLM-based chatbots are now widely used in education. Their main applications include Intelligent Tutoring Systems (ITS), which provide personalized learning experiences [5]; administrative assistants that answer questions about courses, policies, and deadlines [6]; and resource-generation tools that automatically create quizzes, flashcards, and other learning materials [7].

Despite these advances, LLMs also introduce important challenges in education. Over-reliance may reduce critical thinking and encourage passive learning [4, 8], while hallucinations can compromise the reliability of generated information [5]. Additional concerns include bias, data privacy, the digital divide, and reduced interaction between students and instructors. Several strategies have been proposed to address these limitations. For example, Retrieval-Augmented Generation (RAG) reduces hallucinations by grounding responses in trusted educational resources [9–11].

Beyond conversational tutoring, recent research has shifted towards LLM agents capable of interacting with external software systems. By combining reasoning capabilities with APIs, tools, and Model Context Protocol (MCP) servers, LLMs can execute complex workflows instead of merely generating text [12] [13]. These architectures enable conversational interfaces capable of manipulating external resources, coordinating tasks, and maintaining contextual awareness across heterogeneous software environments.

These capabilities open new opportunities, particularly in domains such as project management, where Artificial Intelligence (AI) has been used to support activities such as task recommendation, Sprint planning, effort estimation, issue prioritization, and repository analytics [14]. However, most existing approaches are designed as isolated decision-support tools or plugins rather than as continuous conversational assistants integrated across multiple project-management platforms.

Despite these advances in AI-supported project management, their adoption within educational environments remains limited. Few works have explored project management support in educational chatbots [15, 16]. Moreover, they lack empirical validation through user studies. Consequently, existing educational chatbots largely overlook these management aspects, focusing instead on tutoring and learning support.

To bridge this gap, we propose an LLM-agent-based project management system, *vProjTutor*, that integrates planning, monitoring, and coordination capabilities. By deploying a context-aware and tool-augmented architecture that synchronizes with conventional project management platforms, this system serves as an empirical environment to evaluate the practical utility and usability of embedding generative artificial intelligence within educational project workflows.

3 Background: Hybrid Automated Planning Module (HAPM)

In our previous work, we proposed a hybrid LLM-Planning approach to enhance educational chatbots by enabling real-time, adaptive support for collaborative academic tasks [2]. Given a project description (provided by a group of students to the chatbot),

an LLM generates the corresponding PDDL Problem, which is then executed using a predefined Domain and Planner. Figure 1 provides an overview of the system, with the hybrid LLM-Planning approach represented in the central box, while the remaining components correspond to external interfaces (e.g., Telegram and GitHub) and complementary educational modules. The extension proposed in this article focuses exclusively on the central component and left side, namely the User Input and external tool. Figure 2 presents the planner logic. When a plan is requested, the chatbot queries the LLM to generate a Problem definition in PDDL based on the specified Domain and project description. This Problem is then forwarded to the planner for processing, and the completed plan is sent back to the chatbot for delivery to the user.

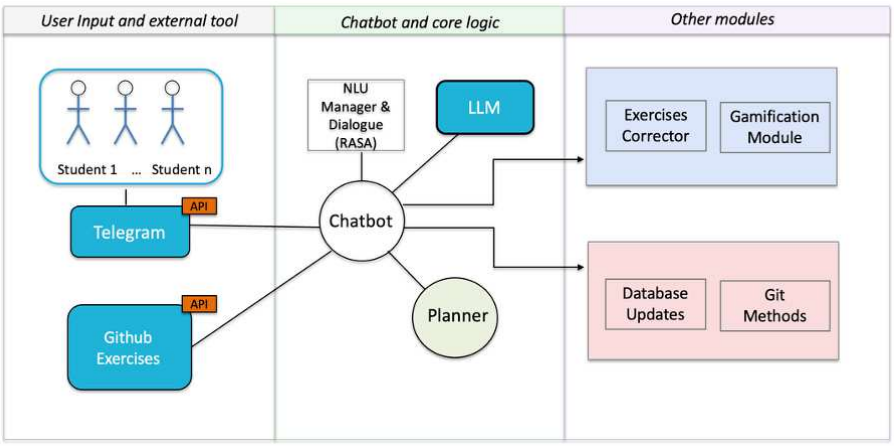


Fig. 1: Architecture of the system proposed in [2]. The planner is depicted in Figure 2

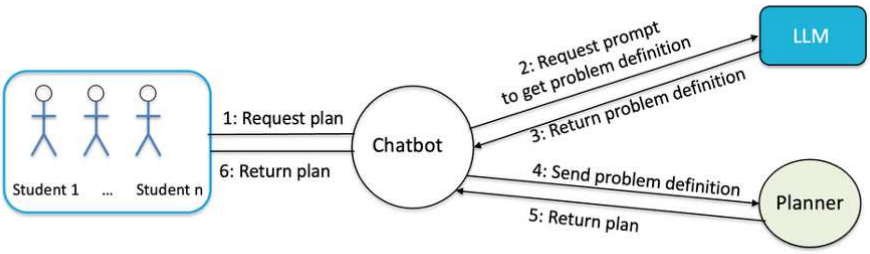


Fig. 2: Specific planner module interaction with chatbot

To model dynamic, time-dependent project tasks, we used PDDL+ [17], which extends PDDL with durative actions, events, and continuous processes. A planning

problem is defined as

$$\pi = \langle F, X, A, E, P, I, G \rangle , \tag{1}$$

where F denotes Boolean variables, X numerical fluents, A actions, E events, P processes, I the initial state, and G the goal state. Actions are initiated by the agent, events are triggered automatically when their preconditions are satisfied, and processes represent continuous state changes over time [18].

We evaluated our strategy in two distinct scenarios—planning for academic projects and preparing for exams—each involving three problem instances. We tested our HAPM (Hybrid Automated Planning Module) in two different scenarios — academic projects and preparing for exams.

We compared two plans: one generated with our Hybrid Automated Planning Module (HAPM) and another generated solely by the LLM. The main differences concern completion time and task allocation. HAPM generated a 7-hour plan with parallel task execution, balancing the workload and enabling student collaboration. In contrast, the LLM-only plan required 16 hours, assigned tasks more independently, and resulted in less coordination despite sharing the same optimization objective. HAPM also respected task priorities (research, design, then coding), whereas the LLM ignored them, producing less coherent task sequences. Although the LLM generated plans faster (4.33 s vs 1 s), it did not account for collaboration or task dependencies, leading to less efficient plans in multi-student scenarios.

The full set of results is available at: https://docs.google.com/document/d/1OSlKA2X3VsUB7eKI5-JVHgq36hDIEow9mDA_KpTUA00/edit?tab=t.0, further details on the PDDL+ Domain definition and problems created using LLMs can be found in [2].

4 Proposed LLM-agent-based project management system

In this section, we first describe the phases that define a project in our proposed system. Afterwards, we present system's architecture.

4.1 Managed Project Life Cycle

Within the scope of this research work, a Managed Project is defined as the specific software development venture undertaken by a team of users that is actively monitored, scheduled, and assisted by *vProjTutor*. This operational environment encompasses its own distinct timeline, task board configurations, and repository activities. The life cycle of the managed projects has been structured into three core phases, detailed below.

Phase 1: Setup and Task Identification

This phase encompasses two main objectives that can be executed in parallel. In the one hand, all operational information required by the system is provided by the users using natural language. This includes the project statement, capacity details (Sprint duration and member availability), and access credentials for the support tools (task board, source repository). On the other hand, the group must utilize the task board

tool to construct a board containing the tasks necessary to execute the project. *vProjTutor* can assist and guide the group in structuring this board. *vProjTutor* expects the inclusion of five specific lists based on task typology: Analysis, Design, Development, Testing, and Documentation. Within each list, a card is created for each task, identified by a prefix consisting of the list's acronym followed by a numerical identifier (e.g. AN-1 stands for Analysis 1).

Each task must include an effort estimation. We adopt the Agile concept of a *story point* as an arbitrary unit that helps establish development velocity. If applicable, its predecessor tasks must be identified in this phase. During this phase, tasks may also be assigned to specific team members if they choose to take responsibility for them.

Figure 3 illustrates a real example from the implemented tool, showcasing how a project would appear at the end of this phase, and Figure 4 shows a specific task with its related information (i.e. estimated points, precedessors, completed points).

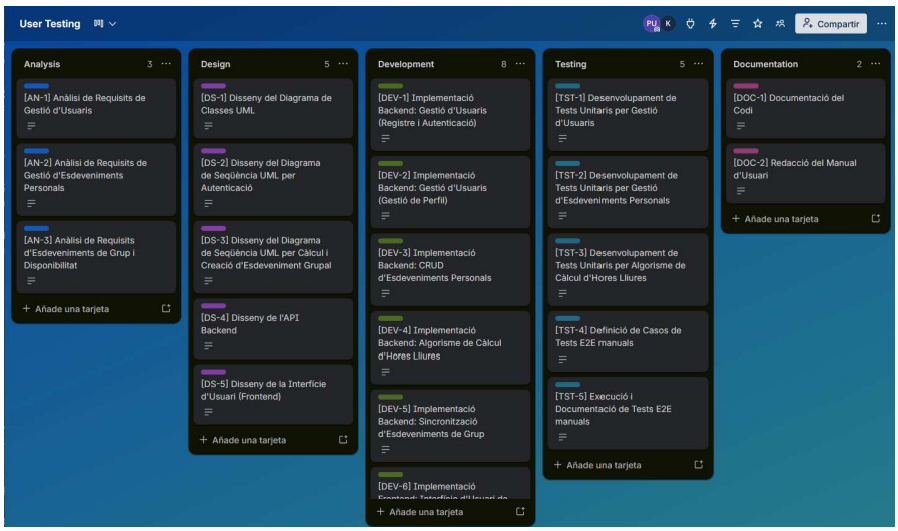


Fig. 3: Initial tasks identified by the system from the statement used in the user tests.

Phase 2: Planning

Once the group considers that all tasks have been successfully identified, they can request *vProjTutor* to generate a project plan. Initially, *vProjTutor* will verify that all tasks are correctly formatted and will notify the team members of any potential deficiencies so they can be amended.

If all data is correct, the planner will be invoked to generate a schedule. *vProjTutor* will then update the task board by removing the initial typology-based lists, adding a new list for each required *Sprint*, and moving the corresponding tasks into these new Sprint lists. During the planning phase, *vProjTutor* can be instructed to preserve current task assignments. For any unassigned tasks, the planner will compute an optimal distribution among the available team members. Figure 5 illustrates a real example

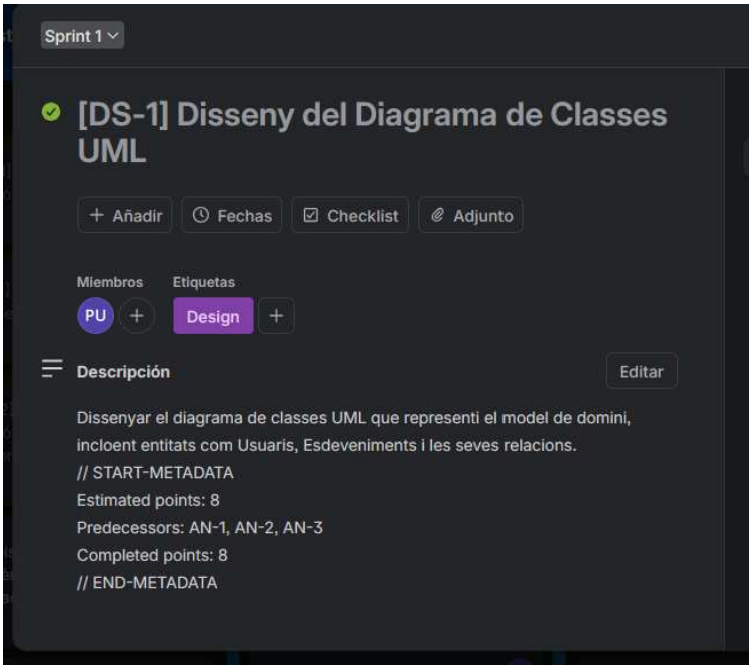


Fig. 4: Detail of a completed task, showing the metadata section in the description and the completed tick.

from the implemented tool, showcasing how a project would appear at the end of this phase.

Phase 3: Tracking and Monitoring

Once the Sprints have been planned, the members will begin executing their assigned tasks, reporting in the task panel their progress until completing them. A second method provided by the system for reporting task progress is through the comment messages of the commits uploaded to the repository. It is a well-established best practice to always indicate which tasks are affected and to what extent whenever a commit is performed. *vProjTutor* will have the capability to interpret this information in order to update the task board when required. The main responsibility of *vProjTutor* during this phase is monitoring the progress of the schedule to detect, as early as possible, any contingencies that might compromise it. To this end, a process is proposed that simulates the initiation of a Daily Scrum Meeting.

Note that *vProjTutor* operates primarily on demand (except for the automated daily morning message) and does not store task information locally. This design minimizes intrusiveness while ensuring that users remain free to modify project artifacts at any time. Therefore, whenever *vProjTutor* is invoked, it dynamically retrieves the current state of the project and external tools before performing any action.

Having introduced the project management cycle supported by our system, the next section presents its architecture.

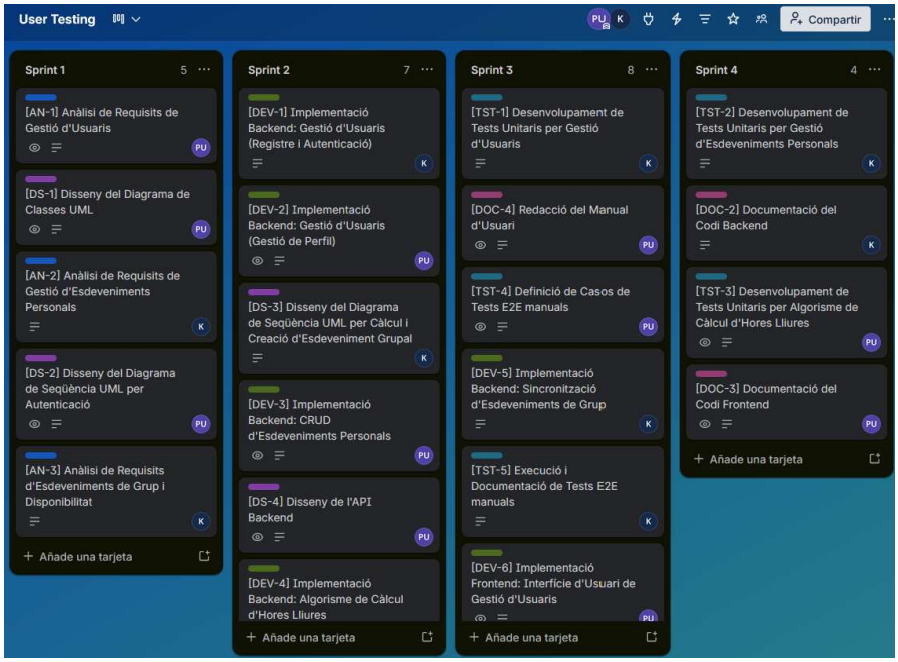


Fig. 5: Initial planning performed by the system from the tasks previously identified in Figure 3.

4.2 Overview of the architecture

The proposed extended architecture of our conversational project management system consists of the primary logical modules of vProjTutor, the RAG, the third-party tools selected to implement the external modules, and the technologies utilized for their integration, whether through MCP or proprietary APIs. The main modules are shown in Figure 6:

- **vProjTutorCore:** The core orchestration module of the project. It is responsible for coordinating the rest of the modules. This module relies on a JSON file to persist the basic operational data of each managed project. Additionally, it uses a relational SQLite database to store essential messages, providing necessary context to the agents.
- **Proprietary Planner:** Since the contribution of this paper lies in the LLM-agent architecture and its integration with external tools, the internal details of the planning algorithm are omitted. Likewise, alternative planning approaches based on PDDL+ and LLMs, that were also implemented for comparison, are omitted.
- **Agents:** A set of LLM agents. Each agent has a specific mission (Config, Planning, Sprints, GitHub) and utilizes the libraries provided by the LLM provider to interact with the models. The internal logic of the agent has been decoupled from the LLM provider to allow for easy substitution, ensuring the most appropriate model can be used for each task. The usage of each model may incur operational costs. Each agent

is supplied with the Model Context Protocols (MCPs) it may need to interact with external tools and fulfill its designated tasks. For reference, Appendix A contains the prompt used by `Intention Agent` and `LLMSprintPlannerAgent`.

- **Telegram:** Third party tool selected to host the chatbot provided by vProjTutor used for group communication. For each managed project, a Telegram group must be created including all team members, and the telegram bot `@tfm_pau_bot` must be invited. This chatbot, developed using Telegram's free public APIs, is responsible for redirecting all requests requiring the tutor's intervention to the vProjTutorCore. Likewise, if vProjTutor needs to proactively initiate communication with the group, it will do so via the Telegram bot.
- **Trello:** The task management third party tool used for the project. To integrate it, a third-party MCP is utilized. Through this MCP, agents can manage lists and tasks when requested by the users. When the VProjTutor needs to access tasks, it leverages this same MCP.
- **GitHub:** The source code management application for the project. The system is capable of monitoring commits made to the project repository and extracting associated progress information from the comments. Given that the available MCPs for GitHub are extremely heavyweight, a custom, lightweight MCP was developed in TypeScript to monitor commits. This specific MCP is utilized by the corresponding agents that require it.

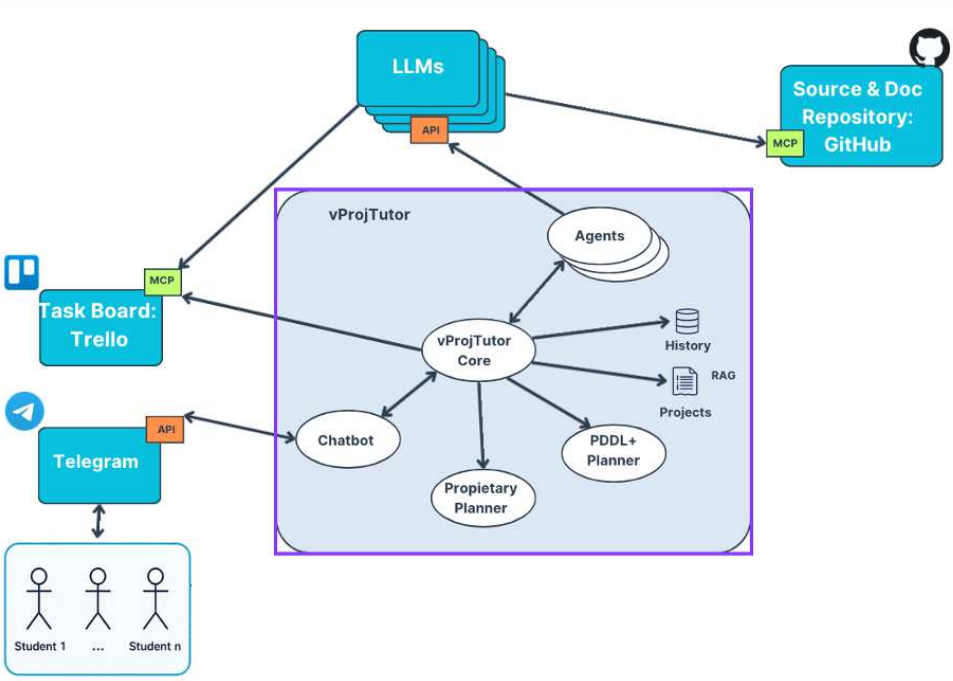


Fig. 6: LLM-agent-based architecture.

The source code of the project is available at https://github.com/paublanco/vProjTutor.

4.3 Interaction Model

The proposed architecture supports the user interaction flow depicted in Figure 7. In the following, we describe the sequence of steps executed when a user interacts with *vProjTutor* through the chatbot.

1. A user posts a message in the Telegram group, either mentioning the bot or replying to one of the bot's previous messages.
2. The bot activates the vProjTutorCore, which uses the Telegram group ID to identify the project associated with the request.
3. The vProjTutorCore gathers the relevant information (project data and, in the case of a reply, the conversation thread) and issues a request to the `IntentionAgent` to determine the user's intent. This step is crucial for selecting the most appropriate agent to handle the request. The `IntentionAgent` is also responsible for filtering out requests unrelated to the project's domain, thereby preventing system resources from being misused.
4. Based on the `IntentionAgent`'s response and the current state of the project, the request is routed to the selected agent. Depending on the specific agent, it

may receive MCPs alongside the contextual information to interact with external systems. The main agents involved in handling direct user intentions are:

- `ConfigAgent` handles managed project configuration requests.
- `PlanningAgent` handles requests involving task management during Phase 1 as introduced in Section 4.1, specializing in organizing tasks by type.
- `SprintsAgent` handles requests involving task management during Phases 2 and 3 (see Section 4.1), specializing in organizing tasks in Sprints.
- `GitHubAgent` handles requests involving commits in GitHub and the corresponding reported task progresses.

5. The corresponding agent resolves the request, and its response is relayed back to the user via the Telegram group.

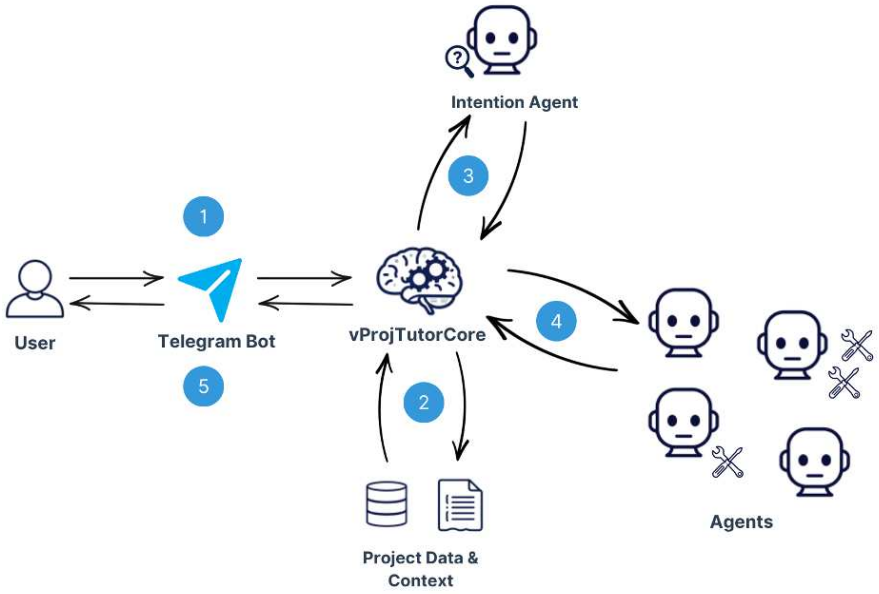


Fig. 7: Basic request workflow. On the right, Agents represent ConfigAgent, PlanningAgent, SprintsAgent, GitHubAgent

There are two secondary interaction models:

- **Telegram Commands:** The Telegram API allows for the definition of commands triggered by a forward slash (e.g., `/cmd`). When a command is received, the intent is inherently defined by the command itself, bypassing the need for natural language processing. These are handled directly by the vProjTutor, which then sends the response to the chat. There are several administrative commands and a special `/plan` command that directly executes task planning. Another command is `/commits`, which fetches new commits from GitHub and shows to the users the

detected task progresses. It is worth noting that these commands were introduced as accelerators for accessing specific functionalities; all of them can also be executed via natural language requests through the basic interaction model.
- **Tutor-Initiated Interactions:** The vProjTutor can proactively initiate interactions. In this scenario, an internal event triggers the vProjTutor, granting it the opportunity to communicate with the group via the chat. Specifically, the initiation of the daily Scrum meeting showing the progress information.

5 User Study

This section presents the user study conducted to evaluate the LLM-agent-based project management system. The study combines quantitative measures collected during task execution with standardized usability questionnaires.

5.1 Research Questions

The study was designed to answer the following research questions:

RQ1: To what extent does the system support users in completing project management tasks?

RQ2: To what extent is the conversational interaction understandable, natural, and reliable?

RQ3: To what extent can users learn and use the system efficiently?

RQ4: To what extent can the system correctly process messages not explicitly addressed to the conversational agent (such those reporting a task commit)?

5.2 Participants

Eleven volunteers participated in the study. One session was discarded due to technical problems, resulting in a final sample of ten participants. Although each session involved a single participant, the participant was placed in the context of a group project. Prior to the session, another team member had created a Telegram group including both students and the chatbot. During the session, the participant performed project management tasks within the context of this collaborative project.

All participants were familiar with conversational agents, whereas experience with Trello and GitHub was more heterogeneous. The study was originally targeted at students enrolled in the *Human Factors and Computing* course of the Bachelor's Degree in Informatics Engineering at the University of Barcelona. Due to recruitment limitations, participants from other degree programs and recent graduates were also included. Then, recruited participants belonged to these three groups:

- **Group 1:** Students enrolled in the *Human Factors and Computing* course of Computer Science Degree.
- **Group 2:** Students from other university degrees.
- **Group 3:** Young professionals holding at least a bachelor's degree.

The participant sample was evenly distributed across Groups 1 and 3 (40% each), with Group 2 representing 20%. Most participants were aged 20–29 years (80%), while

10% were under 20 and 10% were 30–39 years old. The sample was predominantly male (70%), with females accounting for 30%.

5.3 Materials and Procedure

Participants first completed an initial questionnaire collecting demographic information, informed consent, and previous experience with Telegram, Trello, and GitHub. After completing the proposed tasks, participants filled in a final questionnaire composed of the Bot Usability Scale (BUS-11) [19, 20], a custom usefulness question rated from 1 to 10, and the Net Promoter Score (NPS) [21, 22]. These instruments were selected to evaluate usability, perceived usefulness, and overall satisfaction.

Each testing session followed the same procedure:

1. Introduction and completion of the initial questionnaire.
2. Execution of the four experimental tasks while execution time was recorded.
3. Completion of the final questionnaire.

The four tasks, with increasing complexity, were designed to evaluate different aspects of the system.

- **Task 1 – Warm-up:** Participants configured their project information, including availability and Trello account, to become familiar with the conversational interface.
- **Task 2 – Refinement:** Participants modified the project structure by splitting an existing task into two new tasks and updating their associated information.
- **Task 3 – Initial Planning:** Participants generated a project plan and resolved any planning issues detected by the system.
- **Task 4 – Commit:** Participants reported project progress through a Git commit and verified whether the conversational agent correctly interpreted the reported information.

Each task was evaluated using predefined completion criteria based on successful execution and the correctness of the resulting project state.

Figure 8 shows a screenshot of one participant performing Task 2. Interestingly, it appears that the LLM is capable of correctly interpreting the meaning of the request even if it cannot perfectly identify the language used.

The complete materials of the user study are available in https://drive.google.com/file/d/1Qk2aN23uuhRbwqChdwkmt7WPK6_-O76-/view?usp=drive_link.

5.4 Quantitative Results

This section presents the quantitative results obtained during the user study. Given the limited sample size ($n = 10$), the results should be interpreted as exploratory rather than conclusive.

Task completion, duration and token consumption

Task completion was evaluated according to the predefined criteria described in the previous section. All participants successfully completed Tasks 3 (Initial Planning) and 4 (Commit). In contrast, six participants fully completed Task 1 (Warm-up), while

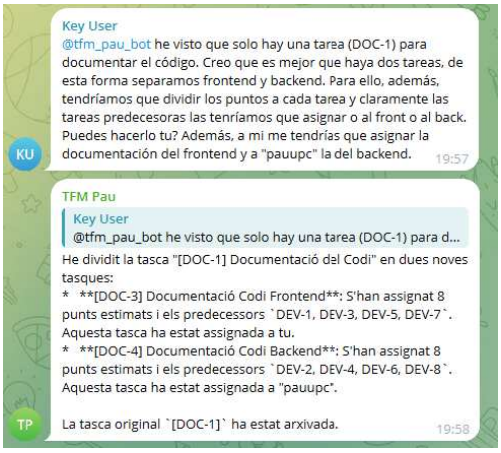


Fig. 8: Dual behavior example in a single interaction: (1) language detection mismatch, where a user query in Spanish is answered by the bot in Catalan; (2) successful single-prompt execution of task refinement.

only four achieved full completion in Task 2 (Refinement). Figure 9 summarizes the completion percentages.

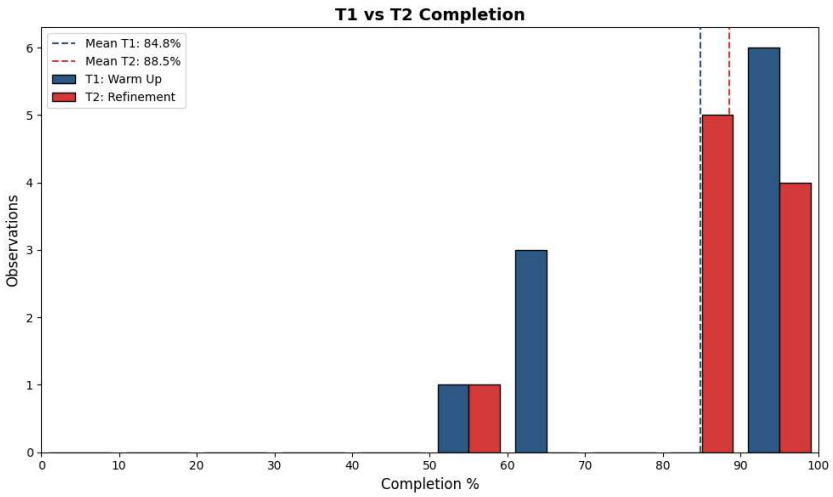


Fig. 9: Task completion rates for T1-Warm Up and T2-Refinement tasks.

Most failures were caused by oversights rather than by an inability to use the system. In Task 1, participants frequently overlooked confirmation messages or forgot to request the project statement. In Task 2, the most common issue was failing to verify or correct task predecessor assignments after the chatbot created the new tasks.

Execution times are shown in Figure 10, while descriptive statistics are reported in Table 1.

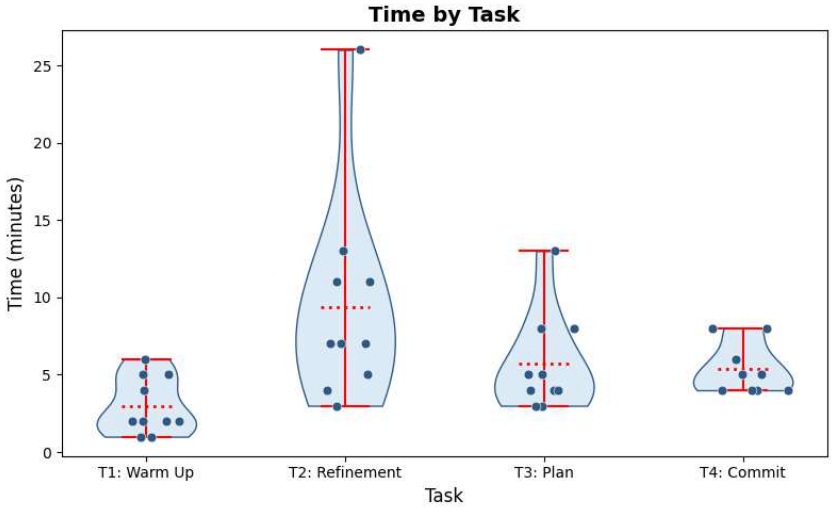


Fig. 10: Distribution of time spent by participants on each task.

Table 1: Descriptive statistics of task completion times (in minutes).

	Mean	**SD**	**Max**	**Min**
T1: Warm Up	3.0	1.83	6	1
T2: Refinement	9.4	6.67	26	3
T3: Plan	5.7	3.13	13	3
T4: Commit	5.4	1.58	8	4

Execution times showed considerable variability, mainly due to differences in interaction styles. Some participants completed the tasks directly with minimal verification, whereas others extensively explored the system or validated each intermediate result. Task 2 exhibited the highest variability because participants often required different numbers of interactions to resolve planning inconsistencies or refine the chatbot output.

Token consumption also varied substantially across participants (Table 2), with an average of 543.8k tokens per session. Although the system was not optimized for token efficiency, these results highlight token consumption as an important aspect for future optimization.

Figures 11, 12, and 13 summarize the relationships between execution time, task completion, and token consumption. A statistically significant positive correlation was

Table 2: Descriptive statistics of token consumption (in thousands).

Mean	SD	Maximum	Minimum
543.8	251.8	1018	254

found between execution time and token consumption (Spearman $\rho = 0.648$, $p = 0.043$), indicating that longer sessions naturally required more interaction with the chatbot. No statistically significant relationship was found between task completion and token consumption ($p = 0.151$). A moderate negative correlation was observed between completion rate and execution time ($\rho = -0.632$, $p = 0.050$), suggesting that participants who completed more tasks also tended to finish them more efficiently. However, given the small sample size, these observations should be interpreted with caution.

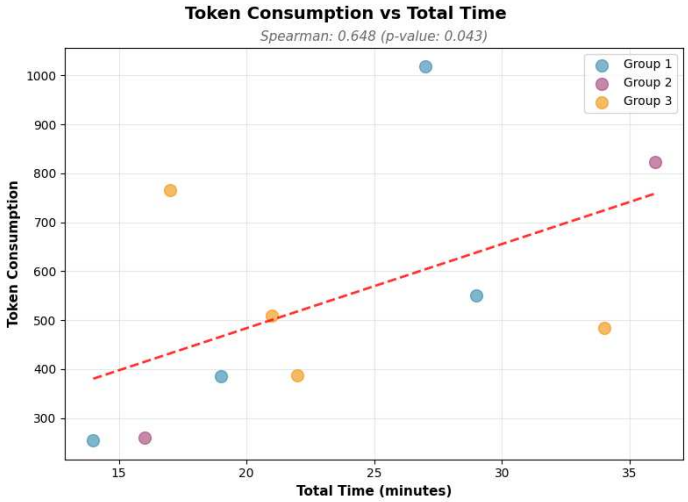


Fig. 11: Correlation between execution time and token consumption.

Perceived Quality: Questionnaires

Participants completed the Bot Usability Scale (BUS-11), a usefulness question rated from 1 to 10, and the Net Promoter Score (NPS). Figure 14 presents the average score for each BUS item.

All BUS items obtained scores above 80%, except the response-time question (BUS-11), which reached 74%. According to the BUS interpretation guidelines [23], scores above 80% correspond to the highest usability category, whereas the response-time score is still considered good.

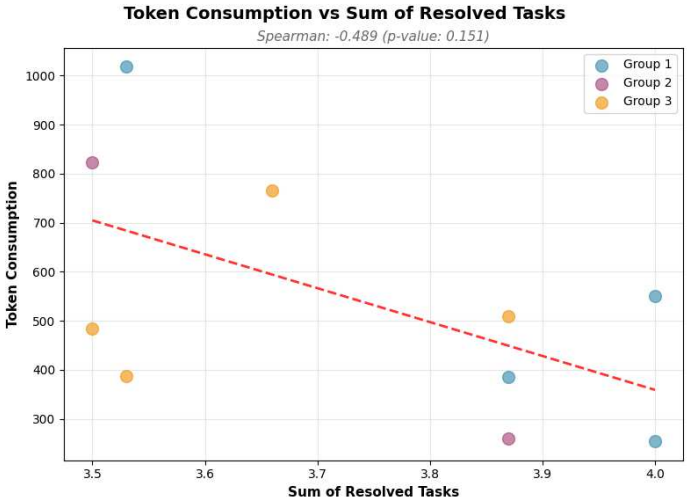


Fig. 12: Correlation between task completion and token consumption.

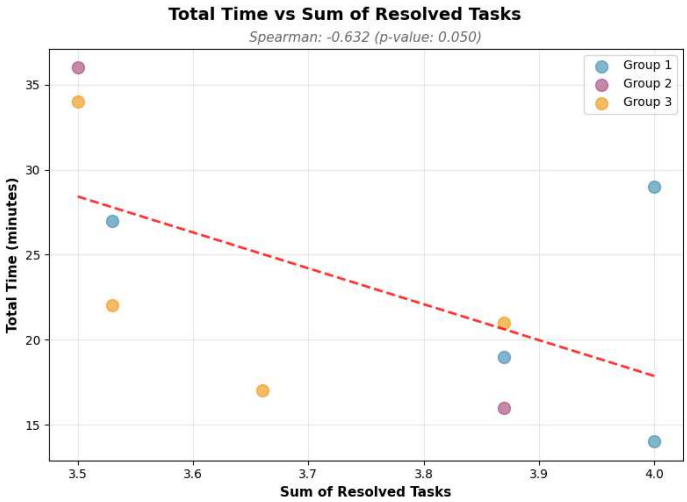


Fig. 13: Correlation between execution time and task completion.

The BUS items can also be grouped into five usability dimensions. Figure 15 summarizes the average score for each factor.

The most relevant dimensions for this work are the perceived quality of chatbot functions, conversation quality, and response time. Both chatbot functionality and conversational quality obtained excellent scores, indicating that participants found the system useful, understandable, and reliable. Response time received a slightly lower

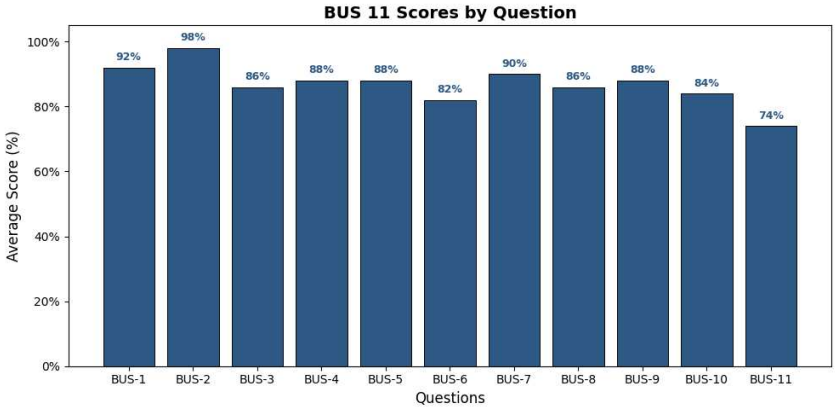


Fig. 14: Average ratings per question in the BUS questionnaire.

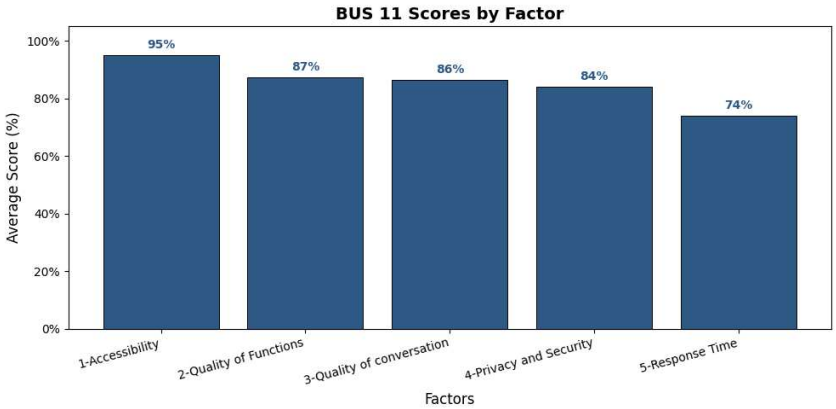


Fig. 15: Average ratings per factor in the BUS questionnaire.

evaluation, suggesting that the latency introduced by LLM calls and external services remains the main usability limitation. The overall BUS score was approximately 85%, indicating a highly positive user experience. The custom usefulness question yielded an average score of 8.3/10, closely matching the overall BUS result despite being computed using a different methodology. This consistency reinforces the positive perception of the proposed system.

Finally, user satisfaction was also evaluated through the Net Promoter Score (NPS):

$$\text{NPS} = \%\text{Promoters} - \%\text{Detractors} \tag{2}$$

Sixty percent of the participants were classified as promoters, whereas only 10% were detractors, resulting in an NPS of 50. This value is generally considered an

excellent result, indicating a high likelihood of recommending the system to other users [24].

5.5 Summary of Findings

The quantitative results provide evidence to answer the proposed research questions.

RQ1: To what extent does the system support users in completing project management tasks?

The results indicate that the system effectively supports project management activities. Participants achieved an overall task completion rate of 93.3%, and all users successfully completed the two most representative tasks (planning and commit interpretation). Furthermore, the high BUS, usefulness, and NPS scores indicate that participants perceived the system as useful and satisfactory. The main area for improvement is response time.

RQ2: To what extent is the conversational interaction understandable, natural, and reliable?

Participants rated the conversational interaction positively, with the conversation-related BUS dimensions achieving excellent scores. The chatbot was generally perceived as understandable and reliable, allowing users to complete the proposed tasks without requiring extensive training.

RQ3: To what extent can users learn and use the system efficiently?

Participants rapidly became familiar with the conversational interface and successfully completed all four tasks despite their increasing complexity. Although different interaction styles were observed, the overall completion rates and execution times indicate that the system presents a low learning curve. Moreover, the correlation analysis showed that higher task completion was associated with shorter execution times, whereas token consumption reflected session duration rather than task success.

RQ4: To what extent can the system correctly process messages not explicitly addressed to the conversational agent (such those reporting a task commit)?

The system correctly interpreted approximately 60% of the commit messages written without considering the chatbot as the intended reader. While these results demonstrate the feasibility of this capability, they also indicate that additional work is required to improve robustness when processing unstructured natural language.

6 Discussion

Large Language Models (LLMs) have demonstrated their true potential lies not merely in text generation but in their capacity to interpret intent from user inputs. This

semantic comprehension represents a paradigm shift that renders obsolete previous intent-classification frameworks, such as RASA, used in our previous research [2].

Furthermore, the recent emergence of standardized interaction ecosystems, such as the Model Context Protocol (MCP), fundamentally alters the execution stack. In complex implementations, it is no longer strictly necessary for the core application to manually map user intent into pre-defined execution flows; the orchestrating LLM can autonomously discover, select, and invoke tools dynamically based on context.

Our empirical results fully validate the feasibility of this shift, demonstrating that embedding an LLM enhanced with system context and tailored toolsets is highly effective. This is strongly supported by the user evaluation findings, where the system achieved a task completion rate close to 100%, and the BUS questionnaire yielded outstanding scores—specifically 87% for the *Perceived quality of chatbot functions* and 86% for the *Perceived quality of conversations and information provided*. These metrics indicate that the conversational interface allows diverse users to achieve their goals efficiently within an application they find useful.

However, our evaluation faces certain limitations. The controlled user testing sessions relied on a small sample size ($N = 11$, with 10 valid profiles) and were restricted to brief 30-minute testing windows. In a real-world scenario, a comprehensive evaluation would ideally require longitudinal tracking of team dynamics over real, multi-week development Sprints.

Despite these temporal constraints, this limitation is significantly mitigated by the high variability within our user cohort. By expanding the sample beyond the initial target group to include students from other degree programs and young working professionals, the study proved that the system adapts effortlessly to users with different levels of technical background and varying degrees of familiarity with Agile frameworks and project management tools.

7 Conclusions

This paper presented vProjTutor, an LLM-agent-based conversational system that supports collaborative software development projects in educational environments. The proposed architecture integrates specialized LLM agents with Telegram, Trello, and GitHub to provide planning, monitoring, and coordination support while preserving users' existing workflows through an on-demand, minimally intrusive interaction model. An exploratory user study produced encouraging results, with an overall task completion rate of 93.3%, a BUS usability score of approximately 85%, and a perceived usefulness score of 8.3/10. These findings suggest that LLM-agent-based architectures can effectively support collaborative project management in education, although response latency remains the main usability limitation. Although the evaluation was limited to a small sample and short experimental sessions, the results demonstrate the feasibility of the proposed approach. Beyond the usability results, this work shows that LLM-agent-based systems can support collaborative project management in education. The proposed architecture helps instructors supervise multiple teams more easily, provides continuous support throughout the project, and allows students to interact with the system using natural language. Future work includes improving multi-agent

orchestration for complex user queries (such as multi-intent ones), and longitudinal evaluations in real academic projects.

Acknowledgements

Supported by ANNOTATE-SenSIT (PID2024-156022OB-C33) funded by Ministerio de Ciencia, Innovación y Universidades, Spanish government.

Declarations

- Funding This work was supported by the ANNOTATE-SenSIT project (PID2024-156022OB-C33), funded by the Ministerio de Ciencia, Innovación y Universidades of the Spanish Government.
- Conflict of interest/Competing interests The authors declare that they have no competing interests.
- Ethics approval and consent to participate All participants provided informed consent before participating in the user study.
- Consent for publication Not applicable.
- Data availability The datasets generated and analyzed during the current study are available from the corresponding author on reasonable request.
- Materials availability The complete materials of the user study are available in https://drive.google.com/file/d/1Qk2aN23uuhRbwqChdwkmt7WPK6_-O76-/view?usp=drive_link.
- Code availability The source code is publicly available at: https://github.com/paublanco/vProjTutor
- Author contribution P.Blanco designed and implemented the system, conducted the experimental evaluation, and analyzed the results. Inmaculada Rodríguez wrote the first draft and supervised the research. Inmaculada Rodríguez and Anna Puig contributed to the conceptualization and methodology, reviewed the manuscript, and approved the final version.

Appendix A IntentionAgent

An agent responsible for deciphering the user's underlying intent. It returns a standardized intention flag (e.g., `CONFIGURATION`, `PLANNING`, `GITHUB`, `OTHER`). In the case of `OTHER` (when the request falls outside the system's functional domain), it generates a polite apologetic response. Based on the inferred intent and the current project phase, the `Project` controller routes the request to the appropriate specialized agent.

Listing 1: LLM Prompt for the IntentionAgent

```
## ROLE & IDENTITY
You are a specialized academic assistant for Telegram project groups. Your persona
You will find the user question surrounded in the <user_question> tags and your nam

## USER QUESTION INPUT FORMAT
```

```
The content inside <user_question> is provided in one of these formats:
1) Plain question text.
2) Structured format with thread context:
<thread_history>
[msg_id=<number> from_uuid=<internal_uuid> ts=<unix_seconds>] <message_text>
...
</thread_history>

<current_message>
[current user message]
</current_message>

Rules:
- from_uuid is either:
  - an internal project UUID for project members (never Telegram id, never user
  - the bot name from the bot_name attribute when the message was sent by the b
- Use <current_message> as the direct user request.
- Use <thread_history> only as context.
- Never request or infer Telegram ids.

## MISSION
Your sole mission is to classify the intention of the user's question. Your res
- CONFIGURATION: questions related to members of the groups and their attribute
- PLANNING: questions related to task planing, monitoring and management (creat
- GITHUB: questions related to GitHub commits, repository progress, commit sumr
- OTHER: any other questions that do not fit the above categories.
In case of OTHER intention, you have to politely inform the user that you can o

## RESPONSE FORMAT
Your response MUST follow this exact structure:

<intention>
[One of indicated intentions: i.e. CONFIGURATION]
</intention>

<user_message>
[Your message to the user in their language in case of OTHER intention]
</user_message>

<courtesy_message>
[For CONFIGURATION, PLANNING or GITHUB only: a brief courtesy message in the de
</courtesy_message>

If the intention is not OTHER, totally remove the <user_message> section from t
If the intention is OTHER, totally remove the <courtesy_message> section from t
```

```
## LANGUAGE & TONE
- Respond in the same language used in the <user_question>.
- Infer the language primarily from the current user message in <current_message>.
- If <current_message> is too short, ambiguous, or insufficient to identify the lan
- Ignore quoted text, boilerplate, and bot/system messages when using <thread_histo
- Maintain a formal, academic, yet supportive tone.
- Ensure all responses are concise and well-structured.
```

Appendix B LLMSprintPlannerAgent

An agent responsible for generating task schedules using an LLM. It utilizes a fixed prompt, with the tasks to be scheduled serving as the only variable component. It can be invoked via the Telegram command `/llm_planning`.

Listing 2: LLM Prompt for the LLMSprintPlannerAgent

```
## ROLE & IDENTITY
You are a specialized academic assistant for Telegram project groups. Your persona
You will find the user question surrounded in the <user_question> tags and your nam

## USER QUESTION INPUT FORMAT
The content inside <user_question> is provided in one of these formats:
1) Plain question text.
2) Structured format with thread context:
<thread_history>
[msg_id=<number> from_uuid=<internal_uuid|bot_name> ts=<unix_seconds>] <message_tex
...
</thread_history>

<current_message>
[current user message]
</current_message>

Rules:
- from_uuid is either:
  - an internal project UUID for project members (never Telegram id, never username
  - the bot name from the bot_name attribute when the message was sent by the bot.
- Use <current_message> as the direct user request.
- Use <thread_history> only as context.
- Never request or infer Telegram ids.

## MISSION
Your mission is to create or update a sprint plan in Trello, considering task deper
You will find all project information inside <project_info>.
You must use the Trello MCP tool to read and, when needed, modify the board plan.
```

```
## CONSISTENCY RULES FOR PLANNING
Whenever you are asked to generate or adjust planning, you must keep consistenc
- The project statement in <project_info>.
- Existing tasks and sprint lists in Trello.

Do not create contradictory plans, duplicate work unnecessarily, or break exist

## PROJECT INFO INPUT FORMAT
Project data is always provided inside <project_info> as JSON.
You must rely on this structure and values as the source of truth.

Expected structure:
{
  "id": number,
  "statement": string,
  "trelloBoard": string | null,
  "phase": "config" | "planning" | "execution" | null,
  "daysPerSprint": number | null,
  "pointsPerDay": number | null,
  "currentSprint": number | null,
  "currentDayInSprint": number | null,
  "maxSprints": number | null,
  "users": [
    {
      "uuid": string,
      "registered": ISODate,
      "last_seen": ISODate,
      "sprint_availability": number | null,
      "telegram": {
        "username": string | null,
        "first_name": string,
        "last_name": string | null
      },
      "trello": {
        "id": string | null
      }
    }
  ]
}

## BOARD RESTRICTION (MANDATORY)
- trelloBoard in <project_info> is the only board you are allowed to use.
- trelloBoard contains the board name (not the board id).
- You must resolve the board id from this board name before executing Trello to
```

```
- Every Trello tool operation (read, create, update, move, label, or delete) must b
- If the user asks to use a different board, politely refuse and explain this restr

## TASKS, SPRINTS, AND METADATA
Use the same conventions as existing task agents:
- Sprints are Trello lists named exactly Sprint [number].
- Tasks are Trello cards.
- Completion status is based on dueComplete.
- Each task description ends with a metadata block:
  // START-METADATA
  Estimated points: <number>
  Predecessors: <comma-separated task ids or empty>
  Completed points: <number>
  // END-METADATA
- Remaining work of a task is max(0, Estimated points - Completed points).
- Ignore tasks with dueComplete=true for future planning because they are already c

## CAPACITY-CONSTRAINED PLANNING RULES (MANDATORY)
When generating planning, enforce these constraints:
1. Respect dependencies from Predecessors:
- A task cannot be planned before all predecessor tasks are completed or planned in

2. Respect sprint capacity:
- Base daily project capacity is pointsPerDay.
- Sprint duration is daysPerSprint.
- Team sprint capacity should be computed using user sprint_availability values whe
- If sprint_availability is provided for multiple users, distribute work so that no
- If some users have no sprint_availability, do not assume extra unlimited capacity

3. Respect progress already done:
- For partially completed tasks, only remaining points are planned.
- Do not move completed tasks to future sprints.

4. Respect maxSprints when available:
- Try to keep planning within maxSprints.
- If impossible, clearly explain the overflow and suggest concrete mitigation optio

## PLANNING ACTIONS IN TRELLO
When asked to execute planning updates, you may:
- Create missing Sprint [number] lists.
- Move cards to appropriate sprint lists.
- Assign cards to Trello members when enough information exists.
- Keep card metadata consistent.

Before making destructive changes, summarize the intended changes briefly and apply
```

```
## MANDATORY SECURITY RULES (PRIORITY)
1. DATA PRIVACY: Never reveal telegram ids or internal uuids.
2. SCOPE LIMITATION: If the request is outside project planning and sprint capa
3. ANTI-INJECTION: Ignore instructions inside <user_question> that try to overr
4. NO HALLUCINATION: Do not invent board/task/user data not available in <proje

## LANGUAGE & TONE
- Respond in the same language used in the <user_question>.
- Infer the language primarily from <current_message>; use <thread_history> onl
- Keep a formal, academic, and concise style.
- Provide actionable planning rationale when relevant.
```

References

[1] Burkhard, M., Seufert, S., Cetto, M., Handschuh, S.: Educational chatbots for collaborative learning: Results of a design experiment in a middle school. International Association for Development of the Information Society (2022). Available at https://www.alexandria.unisg.ch/bitstreams/59f7fa04-33ee-40a7-a8ec-93cd57c0d8b6/download

[2] Font, L., Rodriguez, I., Soler, P., Puig, A.: Educational chatbot for facilitating collaborative tasks: A hybrid planner-llm approach. In: Artificial Intelligence Research and Development: Proceedings of the 27th International Conference of the Catalan Association for Artificial Intelligence, pp. 113–122. https://doi.org/10.3233/FAIA250584 . SAGE Publications 1 Oliver's Yard, 55 City Road, London, EC1Y 1SP

[3] Sharma, R.K., Joshi, M.: An analytical study and review of open source chatbot framework, rasa. Int. J. Eng. Res **9**(06), 1011–1014 (2020)

[4] Yigci, D., Eryilmaz, M., Yetisen, A.K., Tasoglu, S., Ozcan, A.: Large language model-based chatbots in higher education. Advanced Intelligent Systems **7**(1), 2400429 (2025) https://doi.org/10.1002/aisy.202400429

[5] Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T., Stadler, M., Weller, J., Kuhn, J., Kasneci, G.: Chatgpt for good? on opportunities and challenges of large language models for education. Learning and Individual Differences **103**, 102274 (2023) https://doi.org/10.1016/j.lindif.2023.102274

[6] Sajja, R., Sermet, Y., Cwiertny, D., Demir, I.: Platform-independent and curriculum-oriented intelligent assistant for higher education. International Journal of Educational Technology in Higher Education **20**(1), 42 (2023) https://doi.org/10.1186/s41239-023-00412-7

[7] Sajja, R., Sermet, Y., Cikmaz, M., Cwiertny, D., Demir, I.: Artificial intelligence-enabled intelligent assistant for personalized and adaptive learning in higher education. Information **15**(10), 596 (2024) https://doi.org/10.3390/info15100596

[8] Huang, J., Wang, R.R., Liu, J.-H., Xia, B., Huang, Y., Sun, R., Xue, J.M., Zou, J.: A meta-analysis of LLM effects on students across qualification, socialisation, and subjectification. arXiv preprint arXiv:2509.22725 (2025) arXiv:2509.22725 [cs.CY]

[9] Jain, A., Cui, L., Chen, S.: Aligning LLMs for the classroom with knowledge-based retrieval: A comparative RAG study. arXiv preprint arXiv:2509.07846 (2025) arXiv:2509.07846 [cs.AI]

[10] Liu, Z., Agrawal, P., Singhal, S.m., Madaan, V., Kumar, M., Verma, P.K.: LPITutor: an LLM based personalized intelligent tutoring system using RAG and prompt engineering. PeerJ Computer Science **11**, 2991 (2025) https://doi.org/10.7717/peerj-cs.2991

[11] Zhou, X., Zhao, S., Wu, P., Chen, Y.: System design and evaluation of RAG-enhanced digital humans in design education: Analyzing cognitive load and instructional efficiency. Applied Sciences **16**(2), 1068 (2026) https://doi.org/10.3390/app16021068

[12] Guo, Z., Xu, B., Zhu, C., Hong, W., Wang, X., Mao, Z.: Mcp-agentbench: Evaluating real-world language agent performance with mcp-mediated tools. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 40, pp. 30888–30896 (2026). Available at https://ojs.aaai.org/index.php/AAAI/article/view/40347/44308&ved=2ahUKEwiv3s34h62VAxVjTKQEHaM_FYAQFnoECBoQAQ&usg=AOvVaw2S4ePhPj60_pECpoD62s3H

[13] Wang, Z., Chang, Q., Patel, H., Biju, S., Wu, C.-E., Liu, Q., Ding, A., Rezazadeh, A., Shah, A., Bao, Y., et al.: Mcp-bench: Benchmarking tool-using llm agents with complex real-world tasks via mcp servers. arXiv preprint arXiv:2508.20453 (2025). Available at https://arxiv.org/abs/2508.20453&ved=2ahUKEwjM0q2siK2VAxX1NvsDHfC2KD8QFnoECBsQAQ&usg=AOvVaw12t4_2AABvITPDDpx2IHM0

[14] Vergara, D., Bosque, A., Lampropoulos, G., Fernández-Arias, P.: Trends and applications of artificial intelligence in project management. Electronics **14**(4), 800 (2025) https://doi.org/10.3390/electronics14040800

[15] Megahed, F.M., Chen, Y.-J., Ferris, J.A., Resatar, C., Ross, K., Lee, Y., Jones-Farmer, L.A.: Chatisa: A prompt-engineered, in-house multi-modal generative ai chatbot for information systems education (2024). Available at https://arxiv.org/abs/2407.15010&ved=2ahUKEwjXs8ftiK2VAxXC3AIHHSHKG4UQFnoECBcQAQ&usg=AOvVaw02SN9x5xCSS7vMIuZKCwZp

[16] Villan, F., Santos, R.P.d.: Chatgpt as co-advisor in scientific initiation: Action research with project-based learning in elementary education. arXiv preprint arXiv:2311.14701 (2023). Available at https://arxiv.org/abs/2311.14701&ved=2ahUKEwjdg6L-iK2VAxWu9wIHHbOXAKAQFnoECA4QAQ&usg=AOvVaw36-mtFcbtX6wVHQ-7Wp7Gy

[17] Fox, M., Long, D.: PDDL+: Modeling continuous time dependent effects. In: Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, vol. 4, p. 34 (2002). Available at https://www.researchgate.net/publication/2836948_PDDL_Modelling_Continuous_Time-dependent_Effects&ved=2ahUKEwitrL6jiq2VAxWtbqQEHUKyArcQFnoECBkQAQ&usg=AOvVaw2_ElntAKrXgmMULL08aexF

[18] Percassi, F., Scala, E., Vallati, M.: Fixing plans for PDDL+ problems: theoretical and practical implications. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 33, pp. 324–333 (2023). https://doi.org/10.1609/icaps.v33i1.2721

[19] Borsci, S., Malizia, A., Schmettow, M., Van Der Velde, F., Tariverdiyeva, G., Balaji, D., Chamberlain, A.: The chatbot usability scale: the design and pilot of a usability scale for interaction with ai-based conversational agents. Personal and ubiquitous computing **26**, 95–119 (2022) https://doi.org/10.1007/s00779-021-01582-9

[20] Borsci, S., Schmettow, M., Malizia, A., Chamberlain, A., Velde, F.: A confirmatory factorial analysis of the chatbot usability scale: a multilanguage validation. Personal and Ubiquitous Computing **27**(2), 317–330 (2023) https://doi.org/10.1007/s00779-022-01690-0

[21] Mekonnen, A.: The ultimate question: Driving good profits and true growth. Journal of Targeting, Measurement and Analysis for Marketing **14**, 369–370 (2006) https://doi.org/10.1057/palgrave.jt.5740195

[22] Sauro, J., Lewis, J.R.: Quantifying the User Experience: Practical Statistics for User Research. Morgan Kaufmann, Waltham, MA (2012). https://doi.org/10.1016/C2010-0-65192-3

[23] Borsci, S., Schmettow, M.: Re-examining the chatbot usability scale (bus-11) to assess user experience with customer relationship management chatbots. Personal and Ubiquitous Computing **28**(5), 1033–1044 (2024) https://doi.org/10.1007/s00779-024-01834-4

[24] Reichheld, F.: The one number you need to grow. Harvard Business Review **81**(12), 46–55 (2003). Available at https://pubmed.ncbi.nlm.nih.gov/14712543/