Graduate Certificate in Intelligence Reasoning Systems

Practice Module

May 2025

Group 7: SprintPredict

Group Leader	Harshit AGARWAL	A0315374E	E1509813@u.nus.edu
Member	Chitrarath BHATTACHARJEE	A0315514M	E1511452@u.nus.edu
Member	Jia Hui GAN	A0260540R	E1092587@u.nus.edu

Table of Contents

1 Introduction

1.1 Overview

This project introduces SprintPredict, an AI-driven sprint planning and prioritization system that integrates concepts from Reasoning Systems, Machine Reasoning, and Cognitive Systems.

1. **Rule-Based Reasoning:** SprintPredict uses explicit rules to enforce planning constraints, manage task dependencies, and ensure sprint feasibility. For example, if a high-priority task is blocked, the system automatically postpones lower-priority items and respects developer capacity and skill requirements.

2. **Machine Reasoning (Supervised Learning):** The platform analyses historical sprint data to train regression models for velocity forecast. Therefore, allowing the system to adapt to changing team dynamics and improve sprint planning accuracy over time.

3. **Cognitive Systems Principles:** SprintPredict is designed for contextual awareness and user-centric planning. It considers real-time factors like team availability, holidays, and impediments, and simulates human-like decision-making by handling uncertainty and supporting team-specific planning heuristics.

By combining these AI techniques, SprintPredict offers a hybrid, intelligent approach to sprint planning, helping agile teams make better, faster, and more adaptive decisions in real-world project management.

2 Problem Description

Agile teams leverage tools like JIRA, Monday.com, ClickUp, Motion… etc to manage product backlogs and plan sprints. However, sprint planning remains a largely manual, time-intensive, and subjective process, resulting in several inefficiencies:

1. **Inefficient Workload Distribution**: Sprint allocation is often performed manually, leaving significant room for optimization. Achieving an optimal allocation plan that considers team members' availability, capacity, skill sets, preferences, task dependencies, and velocity is complex and challenging.

2. **Subjective Sprint Planning**: Decisions during sprint planning frequently rely on intuition rather than empirical data or business impact analysis, leading to inconsistent workload assignments.

3. **Lack of Data-Driven Insights**: The absence of predictive analytics and insights results in teams either overcommitting or under-committing, often overlooking individual preferences and workload balance.

2.1 Project Aim

The aim of this project is to design, develop and evaluate a web application that leverage AI techniques to optimize sprint planning, fostering greater agility and maximizing project outcomes.

2.2 Project Objectives

The objectives of this project are:

1. Develop an AI-powered sprint planning tool that optimizes task allocation and scheduling based on project requirements, team capacity, and historical data.
2. Develop a machine reasoning-based forecasting module that predicts sprint velocity.
3. Integrate a chatbot that not only addresses Frequently Asked Questions (FAQs), but also perform sensitivity analysis[1] allowing project teams to model 'what-if' scenarios by adjusting project variables and visualizing the potential effects on key performance indicators.

---

[1] Sensitivity analysis is a method used to evaluate how changes in key assumptions or input variables affect the output of a model or system. (Google Gemini, 2025)

2.3 Project Scope

This scope of this project shall apply to project teams that adopts agile / scrum methodologies.

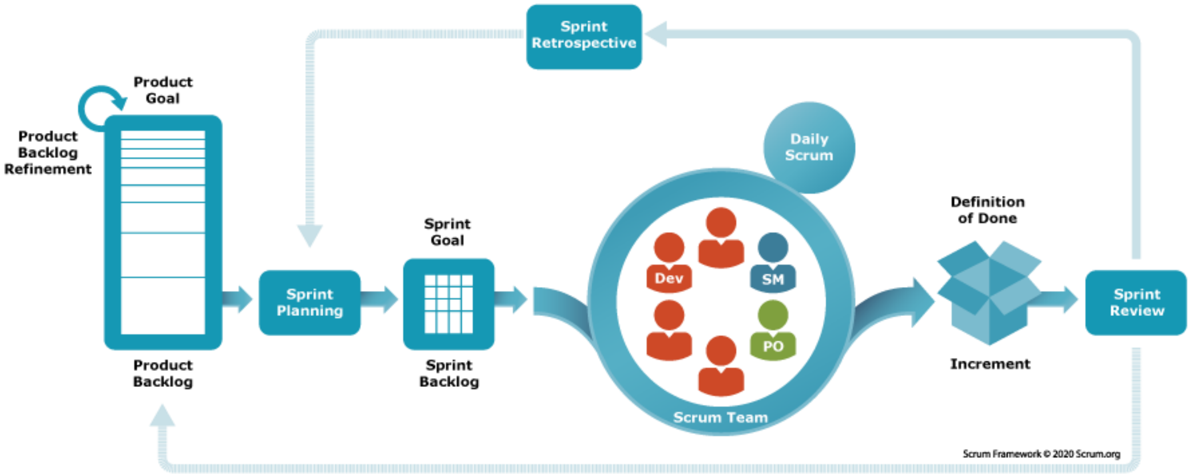

Figure 2-1: Scrum Framework (Scrum Organisation, 2020)

Our AI-powered sprint planning and prioritization system is fundamentally better suited for Agile methodologies than traditional Waterfall methodologies because of its core design principles and capabilities:

1. **Handling Uncertainty and Change**: Agile methodologies embrace iterative development and continuous feedback, recognizing that project requirements can evolve. Hence, our AI system need to be designed to support this dynamism. For example, the Predictive Sprint Planning feature can be used to re-evaluate sprint outcomes based on new information, and the Intelligent Task Allocation Engine can quickly re-allocate tasks if priorities shift. In contrast, Waterfall's sequential, rigid structure struggles to accommodate changes, making AI-driven re-planning less effective.

2. **Emphasis on Iterative Development:** Agile's iterative nature, with its short sprints and frequent releases, aligns perfectly with the AI system's ability to provide ongoing optimization. The system can continuously learn from sprint feedback, refine its predictions, and improve task allocations for each subsequent sprint. Features like the Chatbot interface facilitate this continuous feedback loop, enabling teams to provide real-time updates and allowing the AI to incorporate those updates into its reasoning. Whereas, Waterfall's "big bang" approach lacks these feedback loops, rendering the AI's iterative optimization less valuable.

3. **Value of Continuous Feedback:** Agile encourages continuous feedback from both the development team and stakeholders to guide development. The AI system enhances this process. For instance, the Chatbot Interface enables efficient feedback collection, and NLP for Task Understanding can analyse team's communication to identify potential issues or areas for improvement. This constant flow of information allows the AI to adapt and provide more relevant recommendations in an Agile context. Unlike Waterfall's limited feedback mechanisms restrict the AI's ability to learn and improve.

3 Proposed Solution

3.1 Overall System Architecture

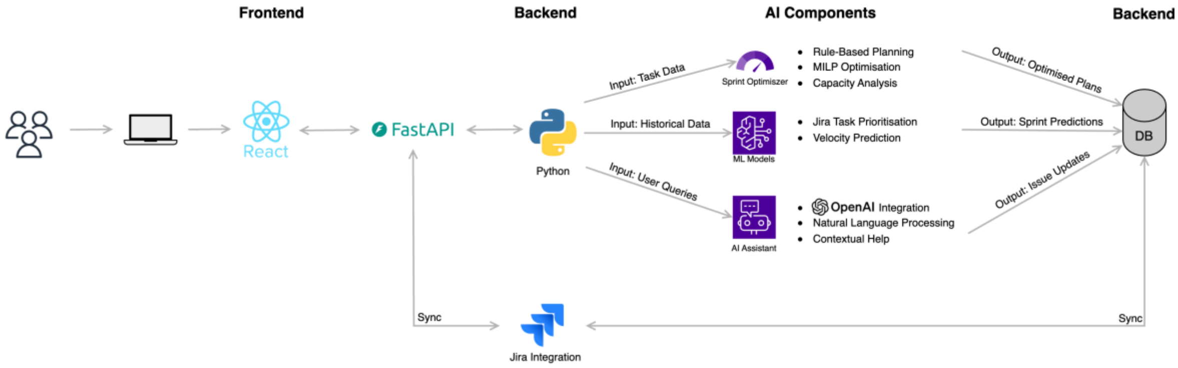

Figure 3-1: Overall System Architecture

SprintPredict is built with a modular, layered architecture to ensure scalability, maintainability, and clear separation of concerns. The main architectural layers are:

- **Frontend Layer:** Developed using React, the frontend provides an interactive dashboard for users to manage teams, product backlogs, and sprints. It communicates with the backend via RESTful APIs
- **Backend Layer:** Implemented with FastAPI, the backend handles business logic, authentication, data processing, and integration with external systems. It exposes REST endpoints for the frontend and for integrations
- **AI Components:** This layer includes:
    - **Sprint Optimizer:** Uses mathematical optimization (MILP) and rule-based reasoning to generate optimal sprint plans based on constraints and team capacity.
    - **ML Models:** Employs supervised learning (Random Forest Regressors) as a method to forecast sprint velocity using historical sprint data.
    - **AI Assistant:** Integrates with OpenAI to provide a conversational interface, natural language processing, and contextual help for users.

- **Data Layer:** Centralized storage for backlog data, team performance, sprint history, and training data for ML models. Data is typically loaded from and saved to CSV files.
- **External Integration:** Connects with JIRA for real-time backlog synchronization, issue tracking, and status updates.

3.2 Mixed Integer Linear Programming (MILP)

3.2.1 Architecture Design

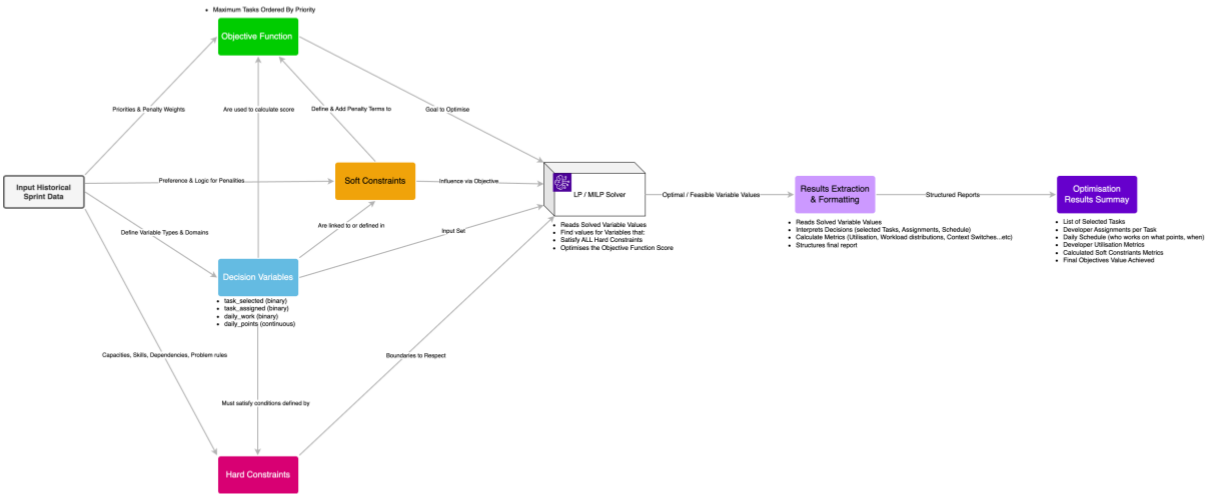

Figure 3-2: MILP Architecture Design

The sprint optimizer uses a Mixed Integer Linear Programming (MILP) approach to solve the sprint planning problem. It combines task selection, developer assignment, and daily scheduling while considering priorities, capacities, dependencies, and skills. The model uses binary variables for discrete decisions and continuous variables for points allocation, with both hard constraints (must be satisfied) and soft constraints (penalized in the objective).

3.2.1.1 Variables

$y[t] \ \varepsilon \ \{0,1\}$	Task selection
$x[t][a] \ \varepsilon \ \{0,1\}$	Task assignment to Developer ‘a’
$w[t][a][d] \ \varepsilon \ \{0,1\}$	Developer ‘a’ works on task ‘t‘ on day ‘d’
$p[t][a][d] \ \varepsilon \ \Re^{+}$	Points of task t assigned to developer ‘a’ on day d”

3.2.1.2 Objective

Maximize $\Sigma(Priority[t] * y[t])$

- $a * (max\_workload - \min\_workload)$

- $\beta * \Sigma(w[t][a][d])$
- $Y * \Sigma(p[t][a][d] * day\_index[d])$

3.2.1.3 Key Constraints

$\Sigma(x[t][a]) = y[t]\ for\ all\ t$	Each selected task is assigned
$\Sigma(p[t][a][d]) \leq daily\_capacity[a][d]$	Respect daily capacity per developer
$total\ points\ per\ task = points[t] * y[t]$	Ensure task completion
$w[t][a][d] \leq x[t][a]$	Work implies assignment
$p[t][a][d] \leq M * w[t][a][d]$	Link points to work binary
Dependency Constraints	For each dependent task t2 that depends on t1, ensure t2 starts only after t1 is completed

3.2.1.4 Pseudocode

```
Initialize MILP Model

Define Binary Variables:
  y[t]         ← 1 if task t is selected, else 0
  x[t][a]      ← 1 if task t is assigned to developer a, else 0
  w[t][a][d]   ← 1 if task t is worked on by developer a on day d, else 0

Define Continuous Variables:
  p[t][a][d]   ← Number of story points of task t assigned to developer a on day d

Objective:
  Maximize total priority of selected tasks
  Penalize workload imbalance across developers
  Penalize context switching across tasks
  Penalize late completion using story point delays

Constraints:
1. Assignment Constraints:
  For each task t:
    Sum of x[t][a] over all developers a = y[t]

2. Completion Constraints:
  For each task t:
    Total story points completed across all a and d = estimated points for t * y[t]

3. Daily Capacity Constraints:
  For each developer a and day d:
    Sum of p[t][a][d] for all tasks t ≤ available capacity[a][d]

4. Work Assignment Linking:
  For each task t, developer a, and day d:
    w[t][a][d] ≤ x[t][a]
    p[t][a][d] ≤ M * w[t][a][d]     // M is a large constant

5. Dependency Constraints:
  For each task t2 that depends on t1:
    Ensure task t2 starts only after task t1 is completed
    (i.e., cumulative points of t1 ≥ points[t1] before t2 can start)

```

```
Solve the MILP Model

Extract Results:
  - Selected tasks
  - Assigned developers
  - Daily work plan
  - Point allocation per day

```

3.2.2 Analysis

While this constraint-based approach enables SprintPredict to generate feasible and optimized sprint plans, it also introduces computational complexity. As the number of tasks and developers increases, the MILP solver must search a much larger solution space, which can lead to longer solve times or, in very large cases, difficulty finding a solution at all. This is a known limitation of CSPs and MILP approaches (Garey & Johnson, 1979)

SprintPredict's codebase implements sprint planning as an optimisation problem, using hard constraints to guarantee feasibility and soft constraints to improve plan quality. This approach enables the system to automate and optimize sprint planning in a way that is both rigorous and adaptable to team needs.

3.3 Random Forest Regressor Algorithm

3.3.1 Architecture Design

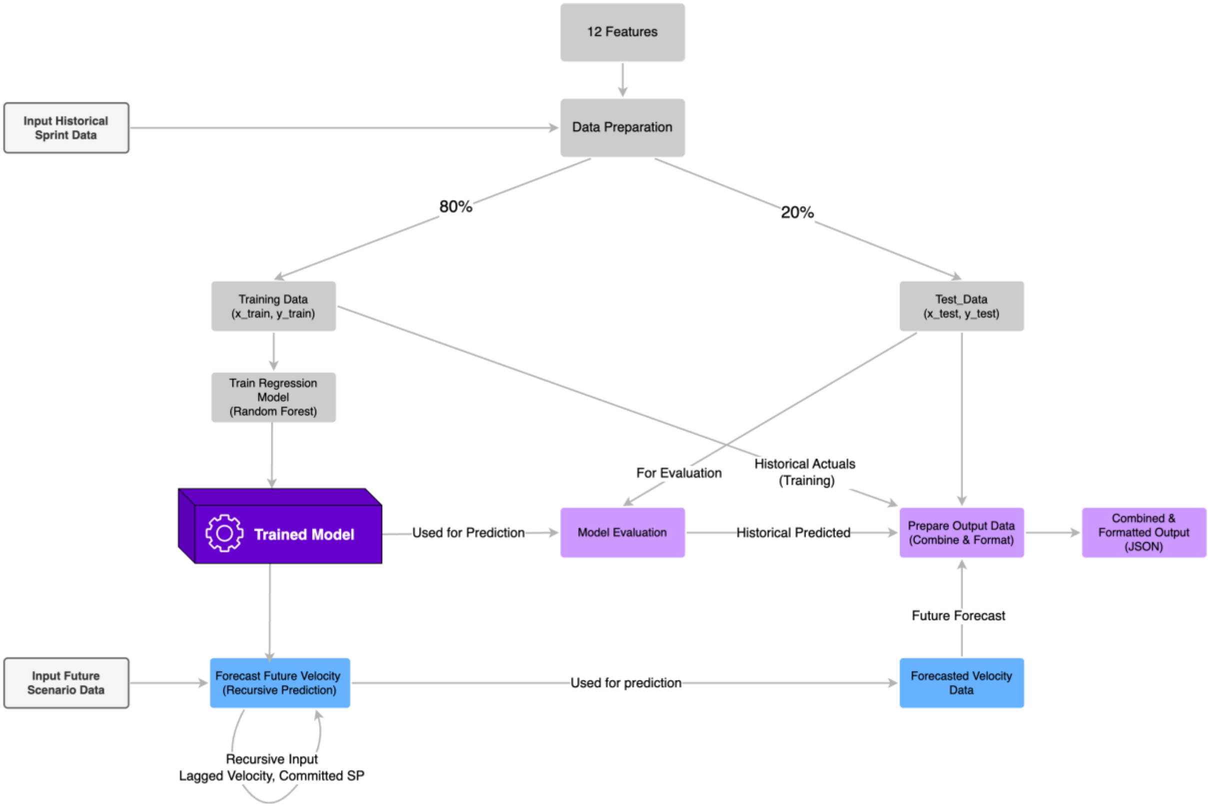

Figure 3-3: Architecture Design

Why Random Forest?

- Robustness to Noise: Random Forests are ensemble models that average the predictions of multiple decision trees, making them robust to noisy data and outliers—common in real-world sprint data.
- Handles Non-linear Relationships: The algorithm can capture complex, non-linear relationships between input features and the target variable (e.g., completed story points).
- Feature Importance: Random Forests provide insights into which features most influence the predictions, helping teams understand key drivers of sprint performance.
- Reduced Overfitting: By aggregating the results of many trees, Random Forests are less prone to overfitting compared to single decision trees.

The codebase (see backend/regression/sprint_velocity_forecast4.py and related files) uses the Random Forest Regressor algorithm from scikit-learn for sprint velocity forecasting. The model is trained on historical sprint data (either simulated or real), with features including team size, sprint duration, available person-days, committed story points, lagged velocity, major impediments, and backlog refinement percentage.

A trained model can be employed prior to sprint planning to enhance the accuracy of velocity forecasting. At this stage, more comprehensive data regarding backlog refinement percentage, team capacity, and committed story points (often derived from an optimization model) is available for input.

The model can be retrained with new data to improve its predictive accuracy over time (though this retraining is currently a manual process). As illustrated in Figure 3-4 below.

Figure 3-4: Supervised Machine Learning Process

1. **Historical Data**

The historical data can either be simulated (for testing and development) or loaded from real project records (such as CSV exports from JIRA or other agile tools).

Our supervised learning model was trained on a historical sprint dataset spanning two years (2023-2024). Why only two years? Because agile sprint cycles typically range from 1 to 4 calendar weeks, with most teams operating on 2-week sprints. Over 2 years, assuming biweekly sprints, an agile team would have completed approximately a total of 52 sprint cycles. This provides a substantial dataset of sprint outcomes.

Furthermore, recent data (within 2 years) is likely to reflect current team dynamics and organizational priorities, ensuring high data quality. As compared to older data (e.g. 5 years) which may include obsolete processes or team structures, thereby reducing its relevance and therefore mislead the model due to the noise.

2. **Prepare Data**

The target is the Completed Story Points (the actual velocity achieved in a sprint).

And the key features in our historical data are:

1	Sprint Number	A unique identifier for each sprint
2	Start Date & End Date	The time period covered by each sprint
3	Team Size	The number of developers participating in the sprint
4	Committed Story Points	The total number of story points the team committed to complete at the start of the sprint

5	Completed Story Points	The actual number of story points completed by the end of the sprint (used as the target variable for velocity prediction)
6	Planned Leave Days (Team)	The total number of planned leave days taken by the team during the sprint
7	Unplanned Leave Days (Team)	The total number of unplanned leave days (e.g., sick leave) during the sprint
8	Major Impediment	A binary flag or severity score indicating whether the sprint was affected by major blockers or impediments
9	Backlog Well-Refined Percentage	A measure of how well-defined and refined the backlog was at the start of the sprint (expressed as a percentage)
10	Sprint Duration (Days)	The number of working days in the sprint, excluding weekends and public holidays
11	Available Person-Days	The effective team capacity, calculated as team size * sprint duration, minus leave days
12	Lagged Velocity	The completed story points from the previous sprint, used as a feature to capture momentum or trends.

3. **Training the model**

- The historical data is split into training (80%) and testing (20%) sets
- Uses the fit() method of the RandomForestRegressor to train the model
- More specifically, it identifies how the different historical feature combinations (listed above) led to specific completed_story_points

4. **Evaluating the model**

- After training, evaluation on the model's performance on unseen data is performed.
- In this case, the chosen evaluation metric is Mean Absolute Error (MAE) from sklearn.metrics. MAE measures the average absolute difference between the predicted values and the actual values. A lower MAE indicates better model accuracy.

5. **Forecasting future sprint velocity**

- The forecast_velocity method uses the trained model to predict the completed_story_points for future sprints.
- It iterates through the desired number of future sprints.
- For each future sprint, it creates a set of feature values. Importantly, for the lagged_velocity feature of the next sprint, it uses the predicted velocity of the current sprint. This makes the forecasting process iterative.
- For other features like team_size, sprint_duration_days, and available_person_days, it uses future planned values (e.g. from self.future_team_size_changes and calculations based on future dates and holidays). For features where future specific values are unknown. For example, committed_story_points, is calculated based on 5% increase from previous forecast velocity. While, major_impediment, and backlog_well_refined_percentage, uses the average values from the training historical data. Committed_story_points.
- The trained self.model's predict() method takes these future sprint features as input and outputs the forecasted completed_story_points (velocity).

3.3.2 Analysis

In summary, supervised machine learning has not only improved the predictive capability by learning from past sprint data. This adaptive capability means that the system is able to swiftly recalibrate task assignment to minimize disruptions and maintain overall sprint productivity.

3.4 Natural Language Processing

3.4.1 Architecture Design

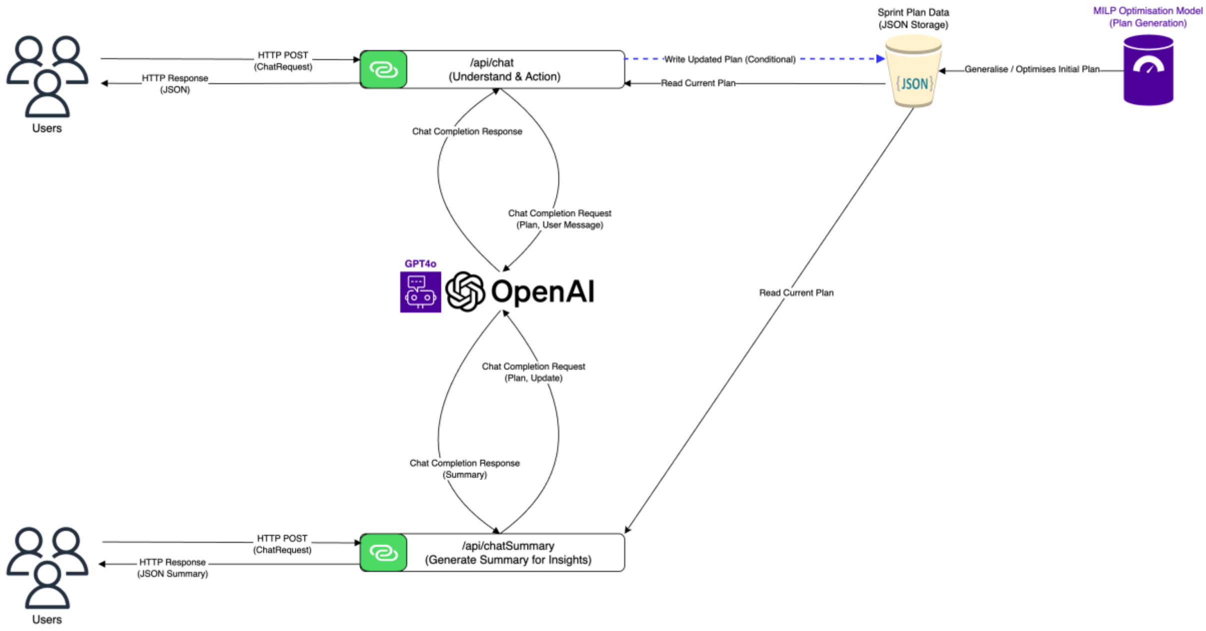

Figure 3-5: LLM Architecture Design

1. **OpenAI API Integration**
   - The backend uses the OpenAI Python SDK to call the GPT-4.1 model with a high token limit (max_tokens = 32768).

2. **Structured Responses and Plan Updates**
   - The assistant is instructed to always provide the following:
   - <u>Answer</u>**:** Direct, structured answer to the user's question.
   - <u>Actionable Suggestions</u>**:** If relevant, suggestions for next steps or plan changes.
   - <u>Updated Sprint Plan</u>**:** If a change is requested, the assistant returns the full updated plan in JSON, wrapped in ```JSON ... ``` markers.
   - <u>Summary</u>**:** Brief summary of what was changed or answered.

3. **Sprint Plan Extraction Logic**
   - The backend parses the assistant's response to check for an updated sprint plan
   - It looks for JSON blocks (either inside ```JSON ...``` or as plain JSON)

3.4.2 Analysis

In summary, the LLM developed in SprintPredict is a context-aware, session-based assistant that leverages GPT-4.1 to provide intelligent, structured, and actionable responses about the sprint plan. It can answer questions, suggest improvements, and directly update the sprint plan, all while maintaining a smooth and modern user experience.

Furthermore, designing a chatbot is a valuable tool for capturing user feedback in real-time which the other AI techniques might have missed.

4 Key System Features

This section highlights the key features in our SprintPredict.

4.1 Dashboard

At first glance, user will see the “Velocity Over Time” graph. This graph tracks how many story points the team has completed in previous sprints and forecasts future performance (Figure 4-1).

The blue line represents historical velocity used for training, while green shows actual performance, yellow represents model predictions, and orange projects future sprint velocity. These trends help teams make more informed decisions, accounting for changes in team size, leaves, or productivity dips.

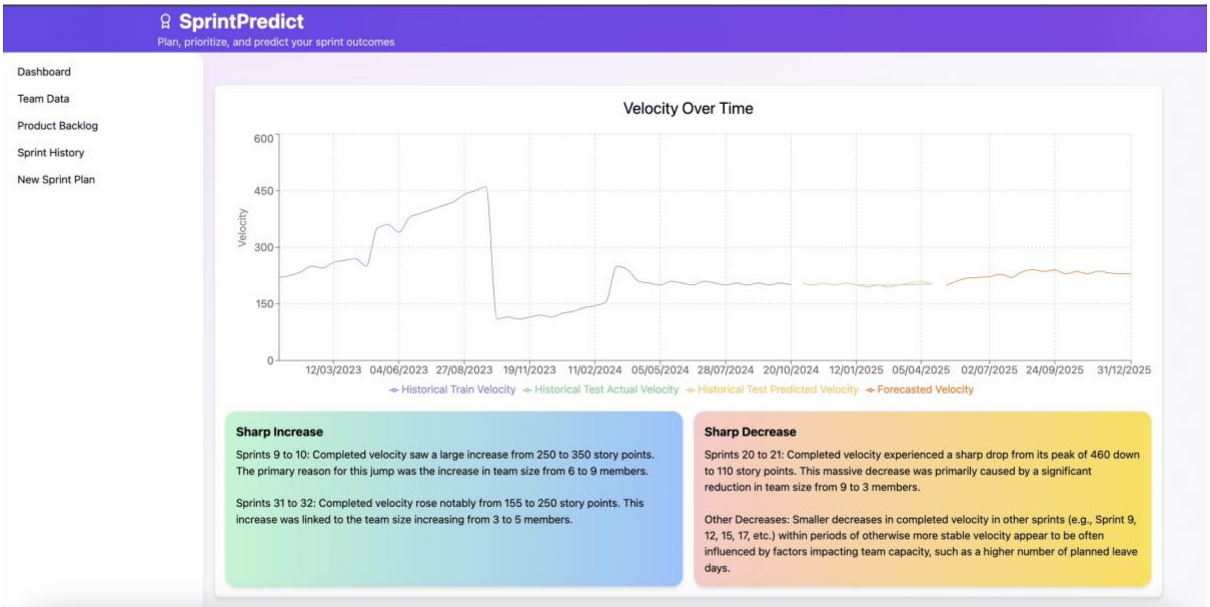

Figure 4-1: Main Dashboard screen

4.2 Team Details

There are 2 aspects in this screen.

The first aspect is on the “Team Details” screen (Figure 4-2) which displays each member’s role, capacity, and technical skills like Python, React, or Java. This data ensures that SprintPredict assigns tasks to the right people based on their availability and expertise, preventing overload and mismatch.

The second aspect is the “Leave and Public Holiday Calendar” screen (Figure 4-3) that integrates both personal leave and public holidays. For example, red marks show when developers are on leave, and blue highlights holidays like Labour Day. The planner adjusts team capacity accordingly, so no work is assigned when team members are unavailable.

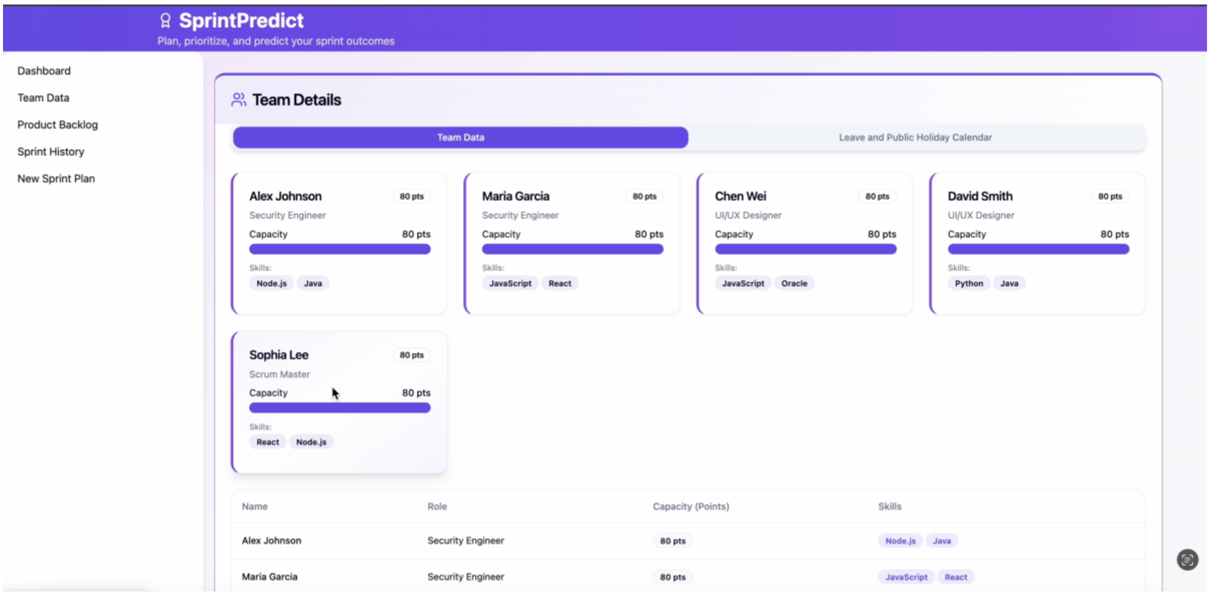

Figure 4-2: Team Detail screen

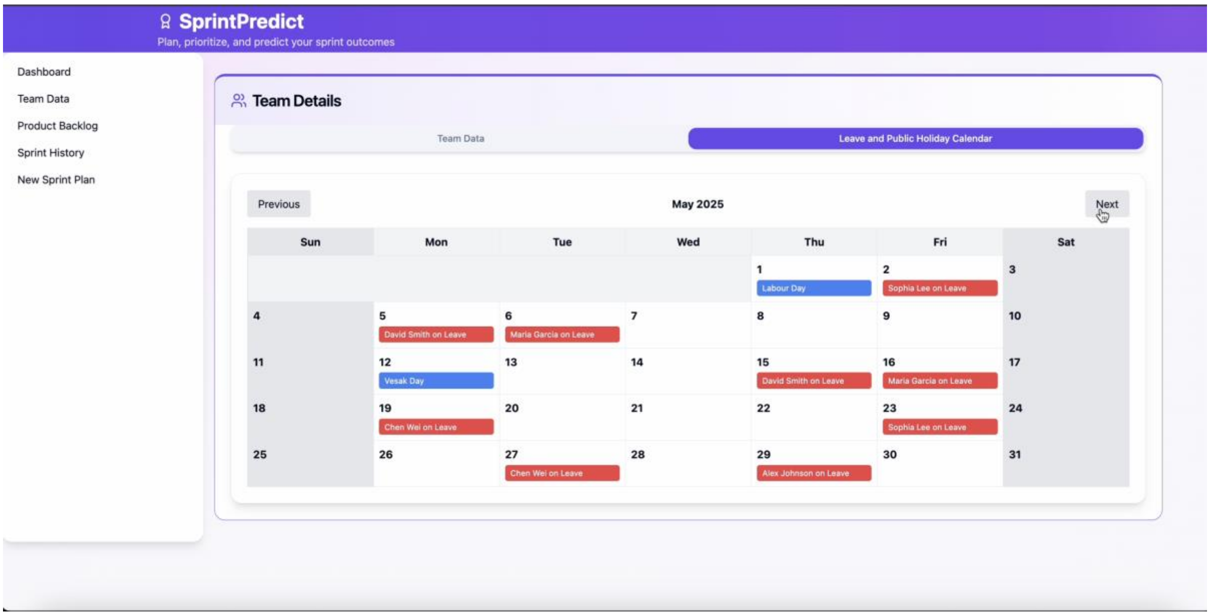

Figure 4-3: Leave and Public Calendar screen

4.3 Product Backlog

In the “Product Backlog” screen (Figure 4-4), users can click on “Sync from JIRA” button to pull relevant details such as priorities, story points, required skills, and dependencies. System will automatically sort it based on Priority, followed by Points from the highest to lowest.

The AI analyses these factors to determine which tasks are ready or blocked, ensuring that planning decisions meets the urgency and feasibility.

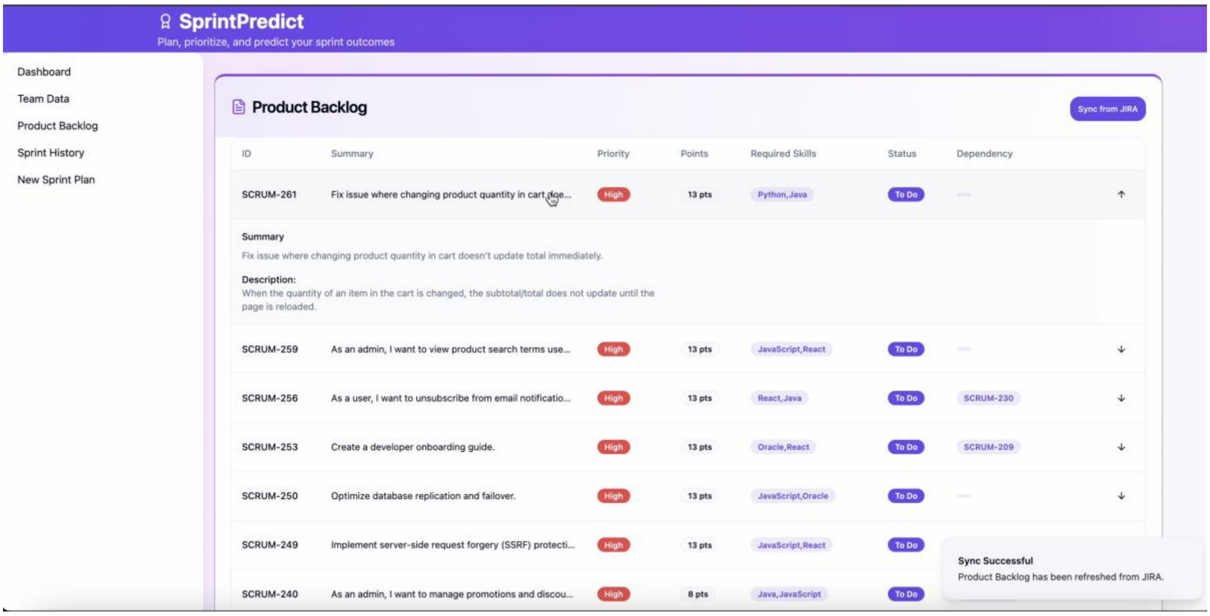

Figure 4-4: Product Backlog screen

4.4 Sprint History

Sprint History screen (Figure 4-5) provides a detailed look at previous sprints — showing team size, committed versus completed points, and success rates. This data feeds into our forecasting model to help predict future sprint performance and optimize planning.

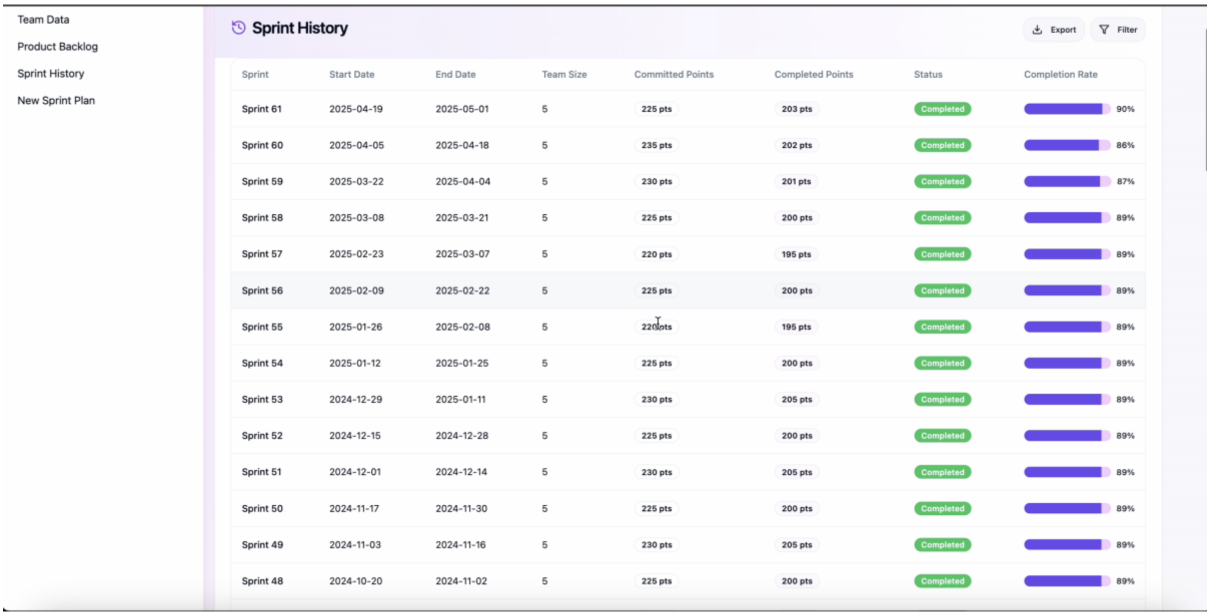

Figure 4-5: Sprint History screen

4.5 Sprint Planning

The Sprint Planning Dashboard (Figure 4-6) is where users define the sprint start date, duration, and backlog readiness. Once initiated, SprintPredict runs an optimization engine based on Mixed Integer Linear Programming (MILP). This model considers hard constraints like capacity, skill match, and task dependencies, along with soft objectives like minimizing context switches and balancing workload.

The resulting plan, as seen in the "Results" screen (Figure 4-7), assigns tasks day by day, matching each developer's skillset and availability. The dashboard shows total tasks, story points, forecasted velocity, and the risk level for the sprint.

The "Risk Level" offers a concise yet powerful gauge of the team's overall sprint health (refer to code snippet below). For instance, an "Overcommitted / High Risk" label acts as an early warning, prompting the team to critically reassess their backlog and ensure a realistic scope. Conversely, indicators of being "Conservative / Under-planned" encourage the team to consider maximizing their potential and delivering even greater value.

For project managers especially, tracking this metric over time helps them to manage expectations effectively and a successful project delivery.

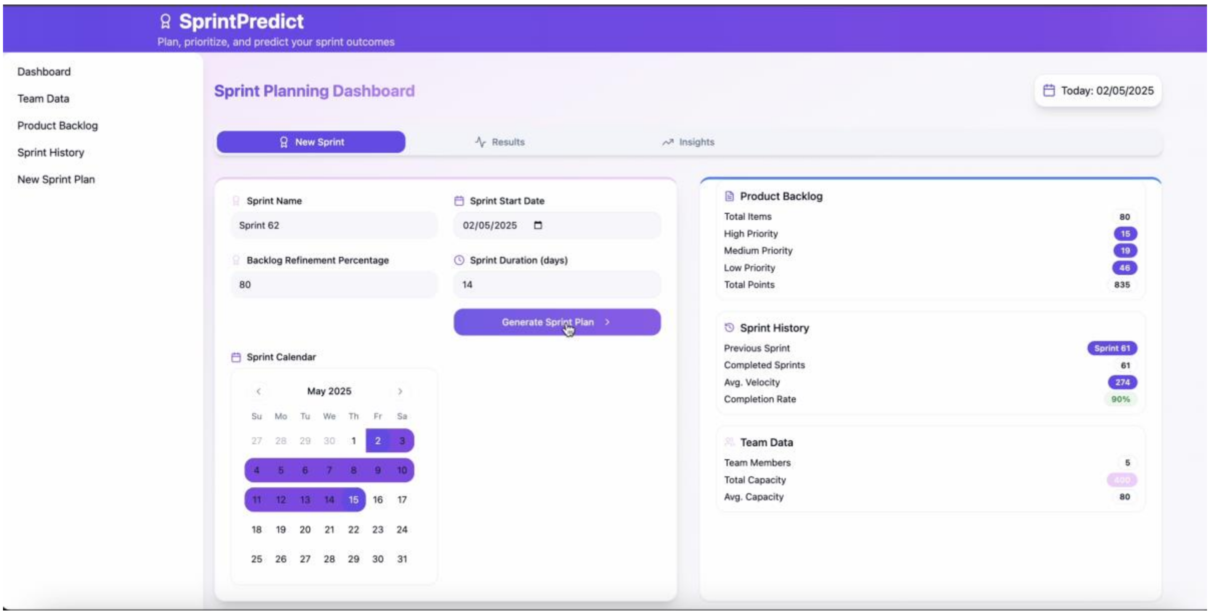

Figure 4-6: Sprint Planning screen

```
forecastedVelocity = Math.ceil(forecastedVelocity); //Convert forecastedVelocity to an integer
const differencePercentage = ((totalStoryPoints - forecastedVelocity) / forecastedVelocity) * 100;

    if (differencePercentage >= 20) {
      return "Overcommitted / High Risk";
    } else if (differencePercentage > 5 && differencePercentage < 20) {
      return "Ambitious / Stretch";
    } else if (differencePercentage >= -5 && differencePercentage <= 5) {
      return "Realistic / On Track";
    } else if (differencePercentage < -5 && differencePercentage > -20) {
      return "Conservative / Under-planned";
    } else if (differencePercentage <= -20) {
      return "Significantly Under-planned";
    }

```

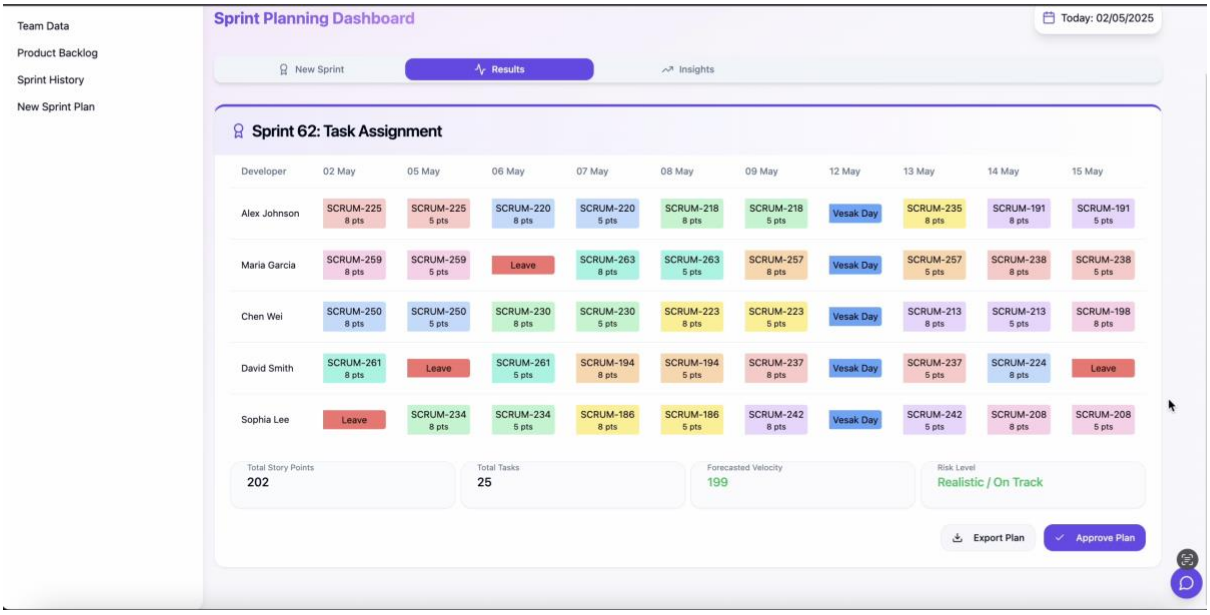

Figure 4-7: New Sprint Planning Result screen

Finally, the "Sprint Insights" screen (Figure 4-8) displays AI-generated recommendations. Using GPT-4o, the system analyses the final sprint plan and highlights unassigned tasks, possible bottlenecks, and suggestions for improvement. For instance, it may explain that a task could not be scheduled because the only available developer was already fully booked. All in all, this helps to fosters trust and understanding in the AI's suggestions.

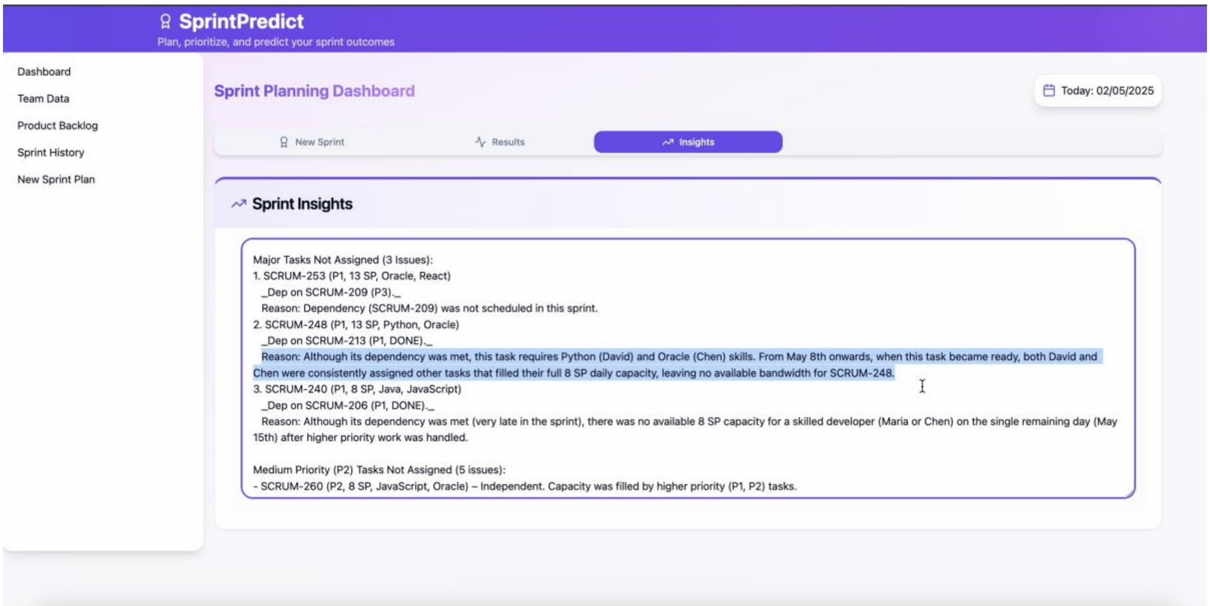

Figure 4-8: New Sprint Planning Insights screen

4.6 Sprint Assistant (Chatbot)

SprintPredict also includes an AI-powered chatbot — the Sprint Assistant — which allow users to perform sensitivity analysis by simulating the impact of various factors – such as adjustments to story points, team strength, and required skillsets… etc. By understanding these potential effects, the team gains valuable insights into the criticality of priorities and can proactively identify necessary remedial or preventive actions to ensure successful sprint outcomes.

The NLP model behind takes the user's input, adjusts the relevant parameters, and presents the resulting changes in sprint timelines, potential scope, or resource utilization. This might involve recalculating critical paths, simulating resource allocation, or projecting completion dates.

For example, how users might interact with chatbot (Figure 4-9):

- *"What happens to the sprint timeline if we underestimate story points by 20%?"*
- *"Show me the impact of adding two more high-priority bugs to the current sprint."*
- *"What if we increase the velocity target by 10%? How many more stories could we potentially include?"*

More importantly, our Sprint Assistant has the explainable AI (XAI) feature. It is able to articulate the reasoning behind its sprint-related decisions. For instance, when it suggests Jira assignments to specific developers, it doesn't operate as a black box. Instead, it can detail the factors influencing its recommendations. This explanation might include considerations such as the developer's historical velocity on similar tasks, their current workload and availability, their skill set alignment with the task requirements, and even team workload balancing to prevent bottlenecks.

All in all, XAI functionality empowers agile teams and project managers with valuable insights into the automated decision-making processes. By understanding the "why" behind the suggestions and calculations, users can validate the reasoning, identify potential biases or areas for refinement in the model, and ultimately make more informed decisions based on a clear and transparent understanding of the AI's contributions. This fosters a collaborative environment where AI acts as an intelligent partner, enhancing rather than obscuring the decision-making process.

For example, how users might interact with chatbot (Figure 4-9):

- *"Explain how you calculated the estimated completion date for this sprint."*
- *"What were the key factors in determining the recommended team capacity for the next sprint?"*

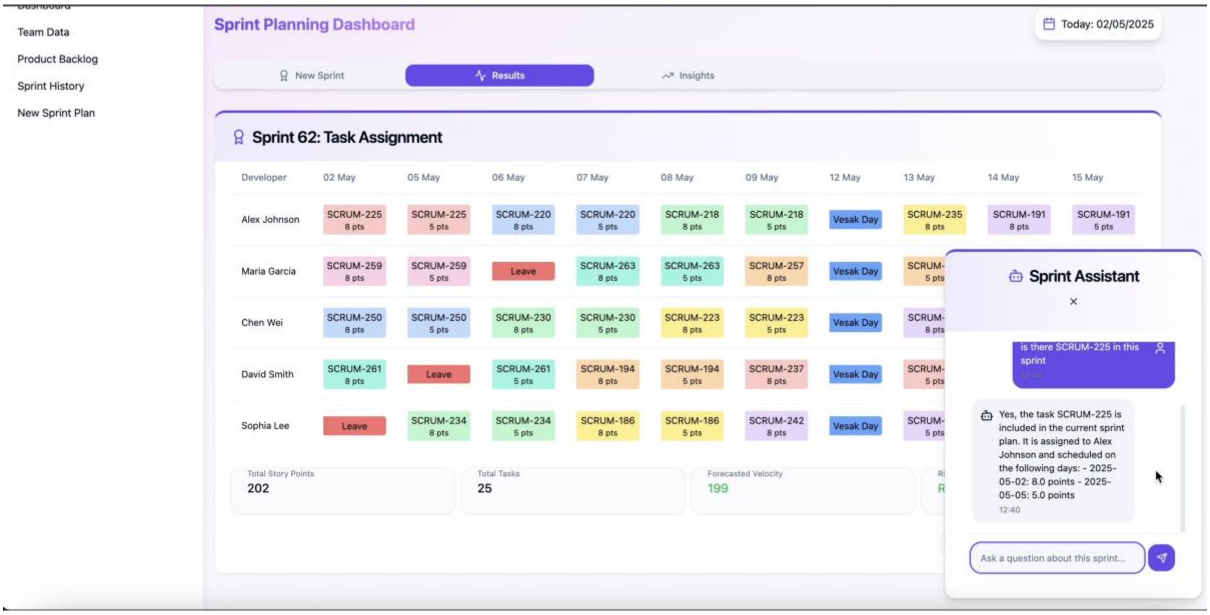

Figure 4-9: Sprint Assistant (Chatbot) screen

4.7 Limitations

4.7.1 AI Limitations

Limitation 1: Data Dependency and Bias

Our SprintPredict ML model learns from historical data. For a newly formed team, there is limited historical data in Jira that the AI can leverage hence, due to incomplete data, it may struggle to provide meaningful recommendations. Or, there could also be inaccurate or biases in the data which the AI could amplify some of these biases in its reasoning.

For example:

A team that previously focused on speed over quality, leading to many rushed tickets indicated as "Resolved" but later reopened. The ML model learns that resolved tickets with the shortest turn-around time equates to successful completion.

Limitation 2: Ethical / Accountability Considerations

When AI system makes a flawed recommendation that leads to project delays or project termination, it is challenging to justify who exactly is responsible – the AI engineers, the users who followed the AI's recommendation or the AI itself?

4.7.2 Real World Limitations

Limitation 1: Life's Unforeseen Circumstances

Our system assumes all, if not most of the team members to be consistently available throughout the sprint. However, in reality, teams might face unexpected events such as family emergencies, illness, sudden resignations or re-assignment to other teams. When such absences occur, it disrupts the sprint planning / forecast that had been carefully planned. According to (Storey, 2020), AI systems typically struggle to readjust meaningfully. It also lacks the emotional intelligence and contextual awareness to sensitively and flexibly adapt in such cases (Dignum, 2019).

Limitation 2: Human's Emotion

Though the AI system may provide an objectively sound recommendations on the sprint, some team members may express distrust or resist algorithmic decisions. One reason is because when algorithm errs, people perceive it as a fundamental flaw in the logic, leading to a stronger negative reaction (Dietvorst, Simmons, & Massey, 2014) It can also be challenging to regain users' trust even after the AI system is improved or correct. User might remain sceptical and ignore AI's recommendation, hence not maximising the full benefits of the AI feature.

5 Conclusion

This project has been an enriching journey for our team. We have gained valuable hands-on experience in area such as knowledge reasoning, machine learning, natural language processing, and web application development. We not only grasped the theoretical concepts but also successfully applied them into a functional system.

5.1 Future Improvements

While we acknowledge that there is no perfect system in the world, but there is certainly room for improvements in our system.

5.1.1 Robustness of AI

Improvement 1: Graph Neural Networks (GNNs) for Jira Dependency Management

There are often complex dependency relationships in a Jira task, such as (a) task dependencies, where a task relies on the completion of the other, or (b) resource dependencies where tasks are dependent on specific resources, or (c) schedule dependencies where task timeline are interlinked (Atlassian Community, 2024).

However, presently, our system mainly supports (a). Hence, exploring GNNs might help to model the other dependencies more extensively and help the AI model understand the various cascading impacts better. Thus, achieving a more optimal sprint planning result.

Improvement 2: Refine Regression Model to capture trends in Unplanned Leaves

Although in our current regression model, SprintPredict is able to adjust the sprint forecast based on the unplanned leaves, more refinements can be done to make the regression model more robust.

One method is to explicitly model and predict unplanned leaves. Instead of reacting to leaves *after* they happen, an estimate of its likelihood can be proactively incorporated and anticipate the impacts *before* the sprint begins.

For example, the current main model adjusts the forecast after unplanned leaves occur. But, we can introduce a different regression model to feed predicted number of unplanned leaves into the main model as an input. The main model can then use this prediction, along with other factors, to generate a more informed initial sprint forecast.

Now, the main regression model now has an anticipatory variable. It can learn the historical relationship between predicted future leaves and actual sprint capacity/velocity, leading to more accurate initial forecasts.

Improvement 3: Human-in-the-Loop Learning for Common Sense Checking

Another improvement is for the AI model to learn / observe how project managers / scrum masters manually adjust the AI's initial Jira prioritizations. Then apply reinforcement learning to learn from these human corrections and refine the AI's future recommendations, leading to a more robust planning. And more importantly, it is to validate AI's decision with human as the "common sense checker".

Improvement 4: Explainable AI

Last but not least, present users with two different proposed project plans or prioritizations generated by the AI and ask them to choose the one they prefer. This direct feedback can be used to train the model to align better with human preferences.

5.1.2 User Interface / User Experience

Improvement 1: Customizable Interface

Implement features that allow users to drag and drop widgets (e.g. pie charts, line/bar/scattered graphs.. etc), resize panels, and select information to display on the dashboard. This way, a well organised and personalised dashboard help users to remain focus and complete tasks faster. System shall also store and automatically reload user's preferences ensuring that users do not need to reconfigure the interface each time they use it. All in all, with customisation and convenience, it can promote a more positive usability experience overall.

Improvement 2: Intelligent Input Assistance

Implement features like auto-completion, smart suggestions, and error prevention to make data input and query formulation more efficient and less error-prone. Users would feel more confident interacting with a system that assist than obstructs them.

6 Acknowledgments

We would like to thank Dr. Gary and Dr. Liya for their professional advices in ensuring that our project was headed in the right direction from the start. And to our MTech course mates for bouncing off ideas together. Their discussions offered feedback that we might have overlooked. Last but not least, to our fellow Group 7 peers, thank you for your dedication as we juggle our individual work and family commitments to complete this practice module successfully.

7 Works Cited

Google Gemini. (2025). Retrieved from https://gemini.google.com

Paul Scerri, P. J.-M. (2025). Retrieved from https://projects.iq.harvard.edu/files/teamcore/files/2002_15_teamcore_scerri_workshop2002.pdf

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Bell Telephone Laboratories, Incorporated.

Ai task allocation in 2025. (2025). Retrieved from callin: https://callin.io/ai-task-allocation/

Scrum Organisation. (2020). What is Scrum.

Xu, S. (27 March, 2025). A Guide to Open-Source Embedding Models.

Atlassian Community. (11 April, 2024). *A Guide to Effective Jira Dependency Management*. Retrieved from Atlassian Community: https://community.atlassian.com/forums/App-Central-articles/A-Guide-to-Effective-Jira-Dependency-Management/ba-p/2667688

Storey, M. e. (2020). The Future of Software Development and the Role of AI. *ACM Future of Software Engineering Track*.

Dignum, V. (2019). *Responsible Artificial Intelligence: How to Develop and Use AI in a Responsible Way.* Springer.

Dietvorst, B., Simmons, J. P., & Massey, C. (2014). Algorithm Aversion: People Erroneously Avoid Algorithms After Seeing Them Err. *Journal of Experimental Psychology*.

8 Appendices

8.1 Appendix 1: Project Proposal

Please refer to GitHub https://github.com/harshitinfiniteinsight/sprintpredict.git

8.2 Appendix 2: From Theory to Practice – MR, RS, CGS

Course Title	How it is applied in SprintPredict system?
Machine Reasoning (MR)	Knowledge Acquisition, Knowledge Modelling, Sprint Optimisation with hard and soft constraints For more details, please refer to section.
Reasoning Systems (RS)	Knowledge Modelling, Supervised Machine Learning, Random Forest Regressor Algorithm. For more details, please refer to section.
Cognitive Systems (CGS)	Large Language Model, Natural Language Processing, Chatbot, Explainable AI. For more details, please refer to section.

8.3 Appendix 3: Installation and User Guide

Please refer to GitHub https://github.com/harshitinfiniteinsight/sprintpredict.git

8.4 Appendix 4: Individual Project Report

Please refer to GitHub https://github.com/harshitinfiniteinsight/sprintpredict.git

8.5 Appendix 5: AI Assistant's (Chatbot) Response

Q1: "What happens to the sprint timeline if we underestimate story points by 20.

SprintPredict's Response: If story points are underestimated by 20%, it means that each task will actually require 20% more effort than currently planned. Since the sprint capacity is fully utilized (utilization rate of 1.0 for all developers), this will directly impact the sprint timeline as follows: - The actual required effort for the selected 202 story points becomes: 202 × 1.2 = 242.4 story points. - Developer capacity remains unchanged, so the scheduled workload (tasks and story points) will now exceed what can be completed within the sprint duration. - As a result, not all planned work can be completed within the current sprint timeline; some tasks will likely spill over into the next sprint, or the sprint would need to be extended beyond the planned dates.

Q2: Show me the impact of adding two more high-priority bugs to the current sprint. Story point 8 and requiring Python skill set.

SprintPredict's Response: Adding two high-priority bug tasks, each with 8 story points and requiring Python skills, will increase the total story points scheduled from 202 to 218, and the total tasks from 25 to 27. Since all developers are currently at 100% utilization, these new tasks will push workload beyond current allocations, or require reassignment or overtime. The best candidates for Python tasks based on typical team skill distribution (not specified in the plan) might be Alex Johnson or Chen Wei, but both are fully booked

Q3: What if we increase the velocity target by 10%? How many more stories could we potentially include?

SprintPredict's Response: Currently, the sprint plan has: - total_story_points_selected: 202 If we increase the velocity target by 10%, the new target would be: - 202 × 1.10 = 222.2 story points (rounded to 222 for planning) This means you could potentially include up to: - 222 (new target) - 202 (current) = 20 additional story points To estimate the number of additional stories (tasks) that could be included: - The average story points per selected task is: 202 / 25 = 8.08 - 20 additional story points / 8.08 ≈ 2.5 So you could potentially add 2–3 more stories (depending on their size).

Q4: Explain how you calculated the estimated completion date for this sprint.

SprintPredict's Response: The estimated completion date for this sprint is determined by the provided list of sprint working days and the allocation of tasks within those dates in the developer_daily_schedule. Here's how the calculation is derived based on your sprint plan: - The sprint_working_days are: ["2025-05-02", "2025-05-05", "2025-05-06", "2025-05-07", "2025-05-08", "2025-05-09", "2025-05-12", "2025-05-13", "2025-05-14", "2025-05-15"] - The developer_daily_schedule shows all task assignments for each developer on these dates. - The last date with scheduled work for any developer is *2025-05-15: - Alex Johnson, Maria Garcia, Chen Wei, and Sophia Lee all have tasks scheduled on 2025-05-15. - David Smith's last scheduled task is on 2025-05-14, but since other developers have work on 2025-05-15, the sprint completion is determined by the latest scheduled work. Therefore, the estimated sprint completion date is **2025-05-15*.

Q5: What were the key factors in determining the recommended team capacity for the next sprint?

SprintPredict's Response: Based on the current sprint plan data, the key factors that determined the recommended team capacity for the next sprint likely included: 1. Individual Developer Sprint Capacity Inputs: Each developer's maximum available capacity was set at 80 units. 2. Utilization vs. Daily Sum Capacity: Each developer's scheduled points exactly matched their available daily capacity, resulting in a utilization rate of 1.0 for all developers. This suggests the team operated at full available capacity. 3. Story Points Scheduled: The total story points scheduled (202.0) were distributed to match the sum of developers' capacities (sum of daily capacities: 56 + 44 + 55 + 34 + 39 = 228, but actual scheduled = 202, so some buffer remains). 4. Workload Imbalance: The workload imbalance deviation was 22.0, indicating some unevenness in distribution but within an acceptable range. 5. Sprint Working Days: The plan considers 10 working days, affecting how much work can be allocated. 6. Soft Constraint Metrics: Metrics such as context switches (32) and late completion penalties (963.0) influence the optimal workload to avoid overloading or inefficiency.