Adaptive Sprint Planning: A Hybrid MILP–RL Framework for Scaled Agile Projects [1]

Enoch Oghene-Mairo OMAJEH

Abstract

Sprint planning in large-scale agile projects is a complex decision-making process that involves balancing task priorities, team capacity, skill alignment, and evolving project conditions. Traditional approaches rely heavily on expert judgment or static optimization models, which often struggle to adapt to dynamic environments. This paper proposes a hybrid sprint planning framework that integrates Mixed-Integer Linear Programming (MILP) with reinforcement learning (RL) to generate feasible and adaptive task allocation strategies. The MILP model ensures optimal task allocation under capacity, skill, and dependency constraints, while the RL component learns from execution feedback to refine decisions across successive sprints in a closed-loop manner. The proposed approach was evaluated using a dataset comprising approximately 500 tasks and 12 team members, with historical sprint data used as a real-world baseline. Results show that the framework consistently outperforms traditional project management outcomes, achieving improvements in task allocation quality (+6.17%), workload balance (−27.78% variance), and delivery reliability (−57.14% task spillover), while maintaining near-real-time optimization performance (≈1.94 seconds). Furthermore, simulation results demonstrate that the RL component progressively improves allocation quality over successive sprints, increasing the normalized objective score from 0.86 to 0.93. These findings demonstrate that integrating optimization with adaptive learning enables not only high-quality initial sprint plans but also continuous performance improvement over time, making the approach well-suited for dynamic agile environments.

Keywords: *Adaptive Sprint Planning; Mixed-Integer Linear Programming; Reinforcement Learning; MILP-RL Framework; Hybrid Optimization; Agile Project Management; Task Allocation*

[1] How to cite this paper: Omajeh, E, O-M. (2026). Adaptive Sprint Planning: A Hybrid MILP–RL Framework for Scaled Agile Projects; *PM World Journal*, Vol. XV, Issue V, May.

1.0 Introduction

1.1 Agile Methodologies

Agile methodologies have become the go-to approaches for managing complex and rapidly evolving projects. This is because they enable teams to respond effectively to changing requirements by breaking down project development into several short development cycles that allows continuous feedback (Highsmith, 2009; Rigby et al., 2023).

However, when organizations adopt agile practices across large and geographically distributed teams, the challenge of coordinating work at scale become pronounced. Scaled agile frameworks, such as the Scaled Agile Framework (SAFe) and Large-Scale Scrum (LeSS) were developed to provide structural guidance for multi-team coordination, yet they continue to rely heavily on manual planning and human judgment.

1.2 Sprint Planning in Large-scale and Distributed Environments

Sprint planning is a critical activity in agile project management, during which tasks are selected, prioritized, and assigned to team members based on specific criteria (which include capacity, skills, and dependencies).

As the number of teams grows, this process becomes more and more complex due to heterogeneous team capabilities, inter-task dependencies, fluctuating workloads, and evolving project priorities (Larman & Vodde, 2016).

Traditional planning approaches, which are based on expert judgment and simple heuristics, are often insufficient to handle this level of complexity. And organizations using them frequently experience suboptimal workload distribution, skill mismatches, and resource bottlenecks (Larman & Vodde, 2016).

1.3 The Emergence of AI in Project Management

Recent advances in artificial intelligence show that AI has the potential to address the complexities inherent in scaled agile project management (Jiang et al., 2019; Menzies & Zimmerman, 2023). For example, Machine learning techniques have been applied to effort estimation, defect prediction, and backlog prioritization while reinforcement learning has shown promise in adaptive resource management and scheduling (Kocaguneli et al., 2012).

1.4 Research Gap

However, existing approaches remain largely fragmented. Optimization-based methods often depend on static models and fixed parameters, while learning-based methods usually lack explicit constraint handling and clear interpretability.

In scaled agile settings, sprint planning requires both strict constraint satisfaction and the ability to adapt as project conditions change. Pure optimization models are less effective when task complexity, team performance, or resource availability evolve over time. On the other hand, standalone reinforcement learning approaches struggle with slow convergence, infeasible solutions, and scalability in large planning spaces. To date, limited research has explored the systematic integration of mathematical optimization and reinforcement learning for adaptive sprint planning in large-scale agile projects.

This paper addresses this gap by proposing a hybrid framework that combines Mixed-Integer Linear Programming with reinforcement learning in a closed-loop feedback architecture for sprint planning in scaled agile projects. In the proposed approach, the MILP component generates feasible and near-optimal task allocation plans based on project constraints and objectives, while the reinforcement learning agent continuously learns from historical sprint outcomes and environmental feedback to update model parameters and decision policies. This integration enables the system to balance optimality, feasibility, and adaptability in dynamic multi-team environments.

To evaluate the effectiveness of the proposed approach, extensive experiments are conducted using real and simulated project datasets. The framework is compared against traditional heuristic-based planning methods and standalone optimization and learning approaches. Performance is assessed in terms of workload balance, sprint completion rate, schedule adherence, and scalability. The results demonstrate that the hybrid MILP–RL framework consistently outperforms baseline methods, particularly in large-scale and dynamic settings, and exhibits superior adaptability over successive planning cycles.

2. Related Work

2.1 Sprint Planning and Task Allocation in Agile Projects

Sprint planning is important to agile project management, as it enables teams to define short-term objectives and allocate work based on capacity and priorities. Early studies on agile planning

emphasized human-centric approaches, which rely on expert judgment, team negotiations, and velocity-based estimations to guide task assignment. These practices are effective in small, co-located teams but are less so in large-scale and distributed environments.

As a result, several researchers have investigated systematic approaches to improve sprint planning in scaled agile projects. Heuristic-based methods, such as priority ranking, skill matching, and workload balancing, have been proposed to support planning decisions (Mohan & Ahire, 2019; De Medeiros et al., 2020; Alshayban & AlZahrani, 2021; Ghanbari & Hoda, 2021). However, these approaches often lack adaptability.

More recent work has explored data-driven methods for sprint planning, which include predictive models for velocity estimation, effort prediction, and delivery risk assessment. Specifically, machine learning techniques are applied to historical project data to forecast sprint outcomes and identify potential bottlenecks (ForouzeshNejad et al., 2025; Prabhneet & Sharma, 2024; Elugbadebo, 2025). Despite these advances, most existing systems focus on isolated aspects of planning and do not provide integrated, adaptive solutions for large-scale agile environments.

2.2 Optimization-Based Approaches in Software Project Management

Mathematical optimization has been widely applied to resource allocation, scheduling, and planning problems in software engineering and project management. Mixed-Integer Linear Programming (MILP) is one of the most widely used because it is suitable for modeling complex decision problems involving discrete assignments, capacity constraints, and multiple objectives.

Several studies have formulated task allocation and scheduling problems as integer or mixed-integer programs, optimizing criteria such as project duration, cost, or workload balance. For example, optimization-based models have been used for staff assignment, release planning, and multi-project scheduling (Karam et al., 2017). These models provide strong guarantees of feasibility and optimality under fixed assumptions.

In agile contexts, optimization techniques have been applied to backlog selection, sprint planning, and release planning. Some approaches optimize story selection based on business value and capacity constraints (Golfarelli et al., 2014; Widodo & Sutabri, 2025), while others incorporate dependency and skill constraints planning (Dybå et al., 2011; Racheva et al., 2025).

However, most existing optimization-based methods rely on static parameters and assume relatively stable environments. When task complexity, team performance, or resource availability

changes over time, the effectiveness of static models deteriorates, as they lack mechanisms to adapt dynamically to changing project conditions (Azonuche & Enyejo, 2024). But we see that large-scale agile projects often exhibit high levels of uncertainty, making it difficult to maintain accurate optimization models. As a result, optimization-based approaches are typically complemented by manual adjustments, limiting their scalability and autonomy.

2.3 Reinforcement Learning in Software Engineering and Project Management

Reinforcement learning (RL) is a technique for sequential decision-making in dynamic environments (Sutton & Barto, 2018). It's commonly used in software engineering to problems like effort estimation, defect prediction, test case prioritization, resource management, and DevOps automation.

In project management and agile development, RL has been explored for adaptive scheduling, backlog prioritization, and workload balancing. Many studies model sprint planning as a Markov decision process, where agents learn policies to select tasks based on observed project states and rewards derived from delivery performance (Zhang et al., 2019; Zhao et al., 2022). RL-based systems are attractive due to their ability to learn from experience and adapt to changing conditions.

However, using RL for sprint planning has several challenges. First, there can be many possible states and actions in large-scale agile projects, which make learning slow and unstable. Second, RL agents can suggest solutions that are infeasible or unrealistic if constraints are not properly included. Third, RL models are often hard to interpret and may not seem reliable to practitioners. As a result, most existing RL-based approaches are limited to small-scale or simplified environments and have not been widely adopted in industrial settings.

2.4 Hybrid Optimization and Learning Frameworks

Hybrid approaches that combine mathematical optimization and reinforcement learning are a promising way to address complex decision-making problems. These frameworks leverage the strengths of both approaches, merging the rigorous constraint handling of optimization and the adaptability of learning-based methods.

In operations research and manufacturing, hybrid optimization–learning systems have been applied to scheduling, supply chain management, and energy systems (Azevedo, et al., 2024).

In software engineering, hybrid frameworks remain relatively underexplored. Some studies have integrated machine learning models with optimization techniques for effort estimation and release planning. However, few works have investigated closed-loop architectures in which reinforcement learning continuously adapts optimization models based on execution feedback.

Also, existing hybrid systems are often designed for offline optimization or limited-scale scenarios. Their applicability to real-time or near-real-time sprint planning in distributed multi-team agile projects has not been systematically studied.

2.5 Research Gap and Positioning of This Study

The review above shows that previous research has produced valuable contributions in agile planning, optimization-based scheduling, and reinforcement learning. But significant limitations remain.

First, most sprint planning approaches rely on static heuristics or isolated predictive models, lacking continuous adaptation mechanisms. Secondly, optimization-based methods provide feasibility and optimality guarantees but struggle in dynamic environments. Third, reinforcement learning approaches offer adaptability but face challenges related to scalability, constraint handling, and reliability. Finally, existing hybrid frameworks in project engineering are limited in scope and rarely implemented as integrated, closed-loop systems.

To the best of our knowledge, there is limited work on unified frameworks that systematically integrate Mixed-Integer Linear Programming and reinforcement learning for adaptive sprint planning in large-scale agile environments. In particular, the use of reinforcement learning as a feedback mechanism to continuously refine optimization-based planning models across successive sprints remains largely unexplored.

This study addresses these gaps by proposing a hybrid MILP–RL framework that combines rigorous constraint-based optimization with experience-driven learning in a closed-loop architecture. The proposed approach is designed specifically for distributed, multi-team agile projects and is empirically validated under realistic conditions. By doing so, this work advances research in intelligent decision support for scaled agile project management.

3 Materials and Methods

3.1 Tools and Implementation Environment

i. Optimization framework: The MILP model was implemented using Google OR-Tools, specifically the CP-SAT solver. And it was chosen because of its high performance in solving large-scale combinatorial optimization problems with complex constraint structures. The solver was accessed through its Python API.
ii. Programming Environment and Libraries: The system was developed in Python (version 3.11). Supporting libraries included NumPy and Pandas for numerical computation and data manipulation, Scikit-learn for preprocessing and feature scaling, PyTorch for implementing the reinforcement learning policy network, and NetworkX for dependency graph construction.
iii. Data Integration: Task and sprint data were ingested through REST APIs from enterprise project management tools, including Jira and Azure DevOps. Skill profiles were stored in structured JSON schemas aligned with the system's skill ontology and automatically populated from repository activity and historical development data.
iv. Deployment and Orchestration: The optimization engine and RL components were containerized using Docker to ensure portability and reproducibility. Containers were orchestrated within the broader system using a microservice architecture deployed on a Kubernetes cluster.
v. Hardware Environment: Experimental evaluations were conducted on a cloud-based virtual machine configured with 4 vCPUs, 8 GB RAM, and Ubuntu 22.04 LTS, enabling consistent benchmarking of solver runtime and scalability performance.

3.2 System Overview

This paper proposes a hybrid sprint planning framework that integrates mathematical optimization with reinforcement learning to support decision-making in scaled agile projects. The framework is designed to generate feasible, high-quality sprint plans while adapting over time to execution feedback and changing team conditions.

At a high level, the system consists of three main components: (i) a data ingestion and feature engineering layer, (ii) a Mixed-Integer Linear Programming (MILP)–based sprint planning model, and (iii) a reinforcement learning (RL) component that enables adaptive refinement across sprints.

3.3 Problem Formulation (MILP)

The engine formulates sprint planning as a constrained optimization problem, designed to assign tasks to team members while maximizing overall sprint value and minimizing delivery risk.

Let the set of tasks be: T = $\{t_1, t_2, \ldots, t_n\}$

and the set of team members be: M = $\{m, m_2, \ldots, m_k\}$

The system aims to determine a task allocation that balances skill alignment, business value, and workload distribution.

The task allocation problem is expressed as a Mixed-Integer Linear Programming (MILP) model. The binary decision variable

$$x_{i\,j} = \begin{cases} 1, if\ task\ t_i\ is\ assigned\ to\ developer\ m_j \\ 0, otherwise \end{cases}$$

is defined for each task-member pair. Additional parameters in the model include:

i. e_i: estimated effort for task ti
ii. cap_j: available capacity of member mj
iii. $S_{i\,j}$: skill-match score between task ti and member mj
iv. d_i: priority or business value of task ti
v. Dep (i, k): equals 1 if task t_i depends on task t_k

The MILP objective is to maximize the combined value and skill alignment:

max $\sum_{i=1}^{n}\sum_{j=1}^{k}(\ d_i.s_{i\,j})\ x_{i\,j}$ ……equation 1

The formulation is subject to the following constraints:

i. Uniqueness: Each task must be assigned to exactly one developer: $\sum_{j=1}^{k} x_{i\,j} = 1 \quad \forall$i
ii. Capacity limits: Developer workloads must not exceed available sprint capacity: $\sum_{i=1}^{n} e_i\ x_{i\,j} \leq cap_{\,j} \quad \forall$j
iii. Dependency ordering: Precedence constraints ensure that tasks with dependencies adhere to temporal or sequencing requirements. If Dep (i, k) = 1, then Start $t_i \geq$ End t_k

iv. Binary decision variable: Each task assignment is modeled as a binary variable $x_{i\,j}$. This ensures that each task is either fully assigned to a single team member or not assigned at all, reflecting the discrete nature of task allocation in sprint planning: $x_{i\,j} \in \{0,1\}$

The optimization model was implemented using Google OR-Tools (CP-SAT). This was chosen for its ability to efficiently handle large-scale integer programming problems involving complex constraint sets. The engine was containerised and executed within a dedicated microservice configured with 4 vCPUs.

3.4 Reinforcement Learning Enhancement

To increase adaptiveness, the system integrates a reinforcement learning (RL) agent that models task allocation as a sequential decision-making problem.

At each timestep t, the system observes a state S_t, which includes team velocity, historical workload, and sprint backlog composition. The agent selects an action A_t, corresponding to assignment decisions, and receives a reward R_t inversely proportional to task delays, rework, or sprint spillovers.

The expected cumulative reward is optimized using policy gradient method:

$\nabla J(\theta) = E\,[\nabla_\theta \log \pi_\theta\ (A_t \mid S_t)\, R_t]$ ……..equation 2

where π_θ is the policy parameterized by θ. This allows the subsystem to learn from past sprints and dynamically improve allocations.

3.5 Data Pipeline

3.5.1 Type of Data and Sources

The Sprint Planning and Task Allocation Engine relies on structured input data to accurately model task assignments and optimize sprint outcomes.

The data sources are:

i. Task Management Systems: Jira and Azure DevOps task metadata, including task IDs, titles, estimated efforts, dependencies, and priorities.

ii. Historical Sprint Records: Completed sprint backlogs providing data on actual task durations, spillovers, and team velocity.

iii. Skill Matrices: Team member skills are primarily inferred automatically from development data (such as code repositories, commit histories, and pull request activity). Where available, additional sources such as CVs or self-reported skills may be incorporated to enrich the skill representation. These inputs are mapped to a standardized skill ontology.

To illustrate the structure of the input data, Table 1 shows a representative subset of the task metadata used as input for the engine.

Table 1: Representative subset of tasks used as input for the SPTA engine (full dataset ≈ 500 tasks).

Task ID	Title	Estimated Effort (hrs)	Priority	Dependencies	Skills Required
T101	Implement Login API	8	High	-	Python, REST, JWT
T102	Design Database Schema	12	Medium	-	SQL, ER Modelling
T103	Frontend Login Page	6	High	T101	React, CSS, HTML
T104	Unit Testing Login	4	High	T101	Python, Testing

Similarly, Table 2 shows an excerpt of the team skill matrix used to compute skill-match scores: Skill ratings are on a 1–5 scale, where 0 indicates no proficiency.

Table 2: Excerpt of team skill matrix used to compute skill-match scores for the SPTA engine.

Member	**Python**	**SQL**	**React**	**Testing**
M1	5	3	2	4
M2	2	4	5	3
M3	4	2	3	5

Dataset Statistics:

For the development and testing of the Sprint Planning and Task Allocation Engine, the dataset comprised approximately

i. Number of tasks per sprint: 500
ii. Number of team members: 12
iii. Average task effort: 7.3 hrs (min: 1 hr, max: 20 hrs)
iv. Average skill coverage per member: 3–5 skills
v. Dependency density: 0.18 (fraction of tasks with at least one dependency)

Processing Pipeline

The raw input data is processed automatically by the preprocessing pipeline before optimization. The pipeline performs the following transformations programmatically:

i. Missing Value Handling: Median imputation is applied for missing task effort estimates; missing skill ratings are set to zero.
ii. Feature Standardization: Numeric features (e.g., task effort) are normalized to a [0,1] range for RL feature representation.
iii. Dependency Extraction: Task dependency graphs are constructed from explicit links in Jira/DevOps or inferred from commit histories.
iv. Skill–Task Similarity Calculation: Cosine similarity between task skill requirements and member skills is computed using TF-IDF weighted vectors to generate the skill-match score $S_{i\,j}$ used in the MILP.

3.5.2: Automatic Data Extraction from Agile Toolchains

Teams are already producing the data the SPTA system needs as part of their normal development workflows. Therefore, the proposed system does not require additional manual data preparation or extra effort from team members.

Key data inputs such as task metadata, effort estimates, priorities, and dependencies are directly obtained from project management tools like Jira and Azure DevOps through their built-in APIs. These tools already store structured information about sprint backlogs, making it possible to automatically retrieve the data needed for optimization.

Developer skill profiles are also not manually created but automatically inferred from existing development data, such as code repositories, commit histories, and pull request activity. For example, a developer who frequently contributes to Python-related files or reviews backend code can be automatically associated with those skills. This approach uses lightweight data analysis techniques, avoiding the need for complex or time-consuming processing.

The feature engineering process (collecting, cleaning, and transforming raw data into the format required by the optimization and learning components) is therefore implemented as an automated preprocessing pipeline integrated into the sprint planning microservice, and runs automatically whenever planning is initiated. This pipeline executes in seconds and operates on data already available within the development ecosystem, ensuring that the approach does not disrupt the rapid planning cadence typical of agile environments.

3.6 System Workflow

Figure 1 illustrates the integrated optimization and learning framework of the sprint planning system.

The process begins with Data Ingestion (task metadata, team capacity, and historical velocity) and Feature Engineering (including skill-match scoring and dependency mapping). This structured data feeds the MILP Solver (CP-SAT), which calculates the Initial Optimized Sprint Plan. This plan is then executed in the real-world environment.

The Reinforcement Learning (RL) Agent observes the plan's actual performance and outcomes (such as delays and rework), and then generate a reward signal that is fed back to the optimization

engine. This Adaptive Feedback Loop allows the RL agent to continuously refine the parameters of the MILP model, dynamically improving allocation policies for subsequent sprints.

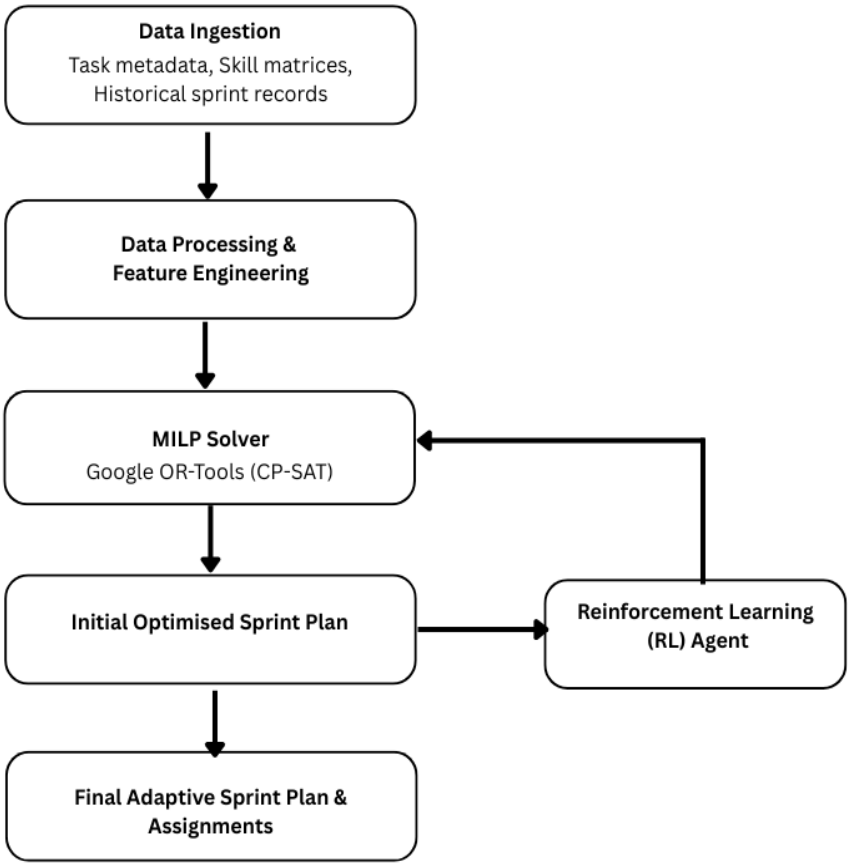

Figure 1: SPTA Subsystem Architecture and Workflow

4. Sprint Planning Model Evaluation

The Sprint Planning and Task Allocation (SPTA) engine was evaluated using the full dataset of 500 tasks and 12 team members described in Section 3.4. The evaluation focused on task allocation quality, workload balance, constraint satisfaction, and computational performance.

To demonstrate the practical benefits of the hybrid MILP–RL framework, its performance was compared against actual historical sprint outcomes managed by experienced project managers, serving as a real-world baseline.

4.1 Baseline Method: Historical PM Data

The baseline represents the actual task allocations observed in previous sprints of the organization (Vent Africa). Historical sprint records extracted from Jira and Azure DevOps, included:

- Task assignments and completion status
- Developer workload and capacity
- Task priorities and dependencies

Metrics were computed from these records to provide a realistic measure of traditional project management performance.

4.2 Performance Metrics

The following metrics were used for comparison:

- Constraint Satisfaction Rate: The percentage of assignments that respect capacity, uniqueness, and dependency constraints.
- Workload Variance (hrs^2): The degree of unevenness in task allocation across team members.
- Task Spillover Rate: The percentage of tasks not completed within the sprint.
- Solve Time (s): The time taken to generate the sprint plan.
- Normalized Objective Score: The aggregate measure of "quality of task allocation"

For each sprint, the Normalized Objective Score is derived directly from the MILP objective function, which sums the weighted skill-match scores multiplied by task priorities for the assigned tasks and normalizes by the maximum possible objective value if every task were perfectly matched. The same calculation is applied to historical baseline assignments using recorded task assignments from past sprints, enabling a consistent comparison between the hybrid MILP–RL system and traditional project management practices.

4.3 Comparative Results

To evaluate the practical benefits of the hybrid MILP–RL framework, historical sprint data from Vent Africa were used as a baseline. The hybrid system was then applied to the same tasks, team capacities, and dependencies recorded in these historical sprints. This allows a direct comparison between actual project management outcomes and the allocations generated by the proposed system.

Table 3: Comparison of Sprint Planning Metrics: Traditional Project Management vs. MILP-Based System

Metric	Historical PM Data	Hybrid MILP-RL system	Improvement (%)
Normalized Objective Score	0.81	0.86	6.17%
Constraint Satisfaction Rate	94%	100%	6.38%
Workload Variance (hrs^2)	16.2	11.7	27.78%
Task Spillover Rate	14%	6%	57.14%
Solve Time (s)	10,800-14,400*	1.94	99.98%

* Planning time for historical sprints was obtained through interviews with project managers and ranged between 3–4 hours **(**10,800–14,400s) per sprint.

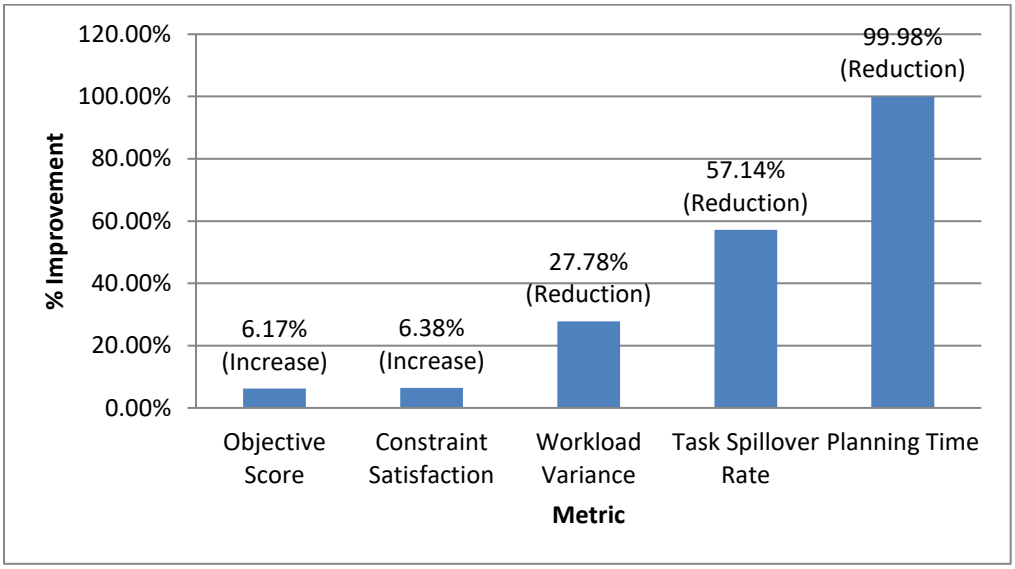

Figure 2: Chart showing percentage improvement of SPTA system over traditional planning approaches

To demonstrate the effect of reinforcement learning over successive sprints, the system was simulated across multiple iterations of the same task set. The first plan reflects the performance of the MILP solver without RL adaptation. For each subsequent iteration, "virtual sprints" are generated using outcome metrics (e.g., task spillovers, workload balance) derived from the simulated allocation. These outcomes serve as feedback to the RL agent, which updates its policy to favor allocations that lead to better results, illustrating how the hybrid system learns from prior outcomes to improve allocations over time.

Table 4: Sprint-by-sprint performance of the MILP–RL system showing progressive improvement over time

Metric	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6	Sprint 7
Normalized Objective Score	0.86	0.85	0.87	0.88	0.90	0.92	0.93
Constraint Satisfaction	100%	100%	100%	100%	100%	100%	100%

Workload Variance (hrs2)	11.7	11.2	10.8	10.5	9.3	8.6	8.12
Task Spillover Rate	6%	6.5%	5.8%	5.2%	3.9%	2.5%	2%
Solve Time (s)	1.94	2.09	1.34	1.90	2.28	2.71	2.01

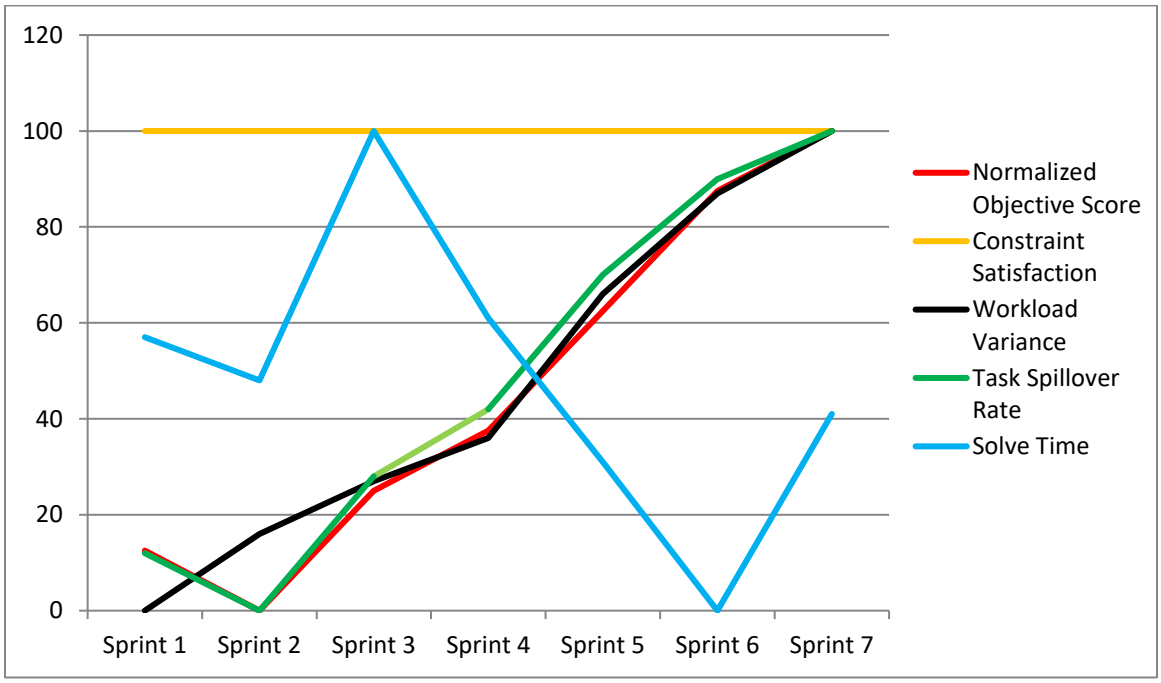

Figure 3: **Evolution of sprint planning metrics across iterations of the hybrid MILP–RL system** (values normalized to a 0–100 scale for visualization)

5. Discussion

5.1. Discussion of result

Table 3 above shows that the hybrid MILP–RL system outperforms the real-world baseline across all key metrics.

Constraint satisfaction improved from 94% to 100%, showing that the system respected all constraints, thereby eliminating conflicts observed with traditional methods. Workload variance was reduced from 16.2 to 11.7, showing a more balanced distribution of work across team members, thereby minimizing the risk of overloading. The normalized objective score increased from 0.81 to 0.86, showing that the system assigns tasks more intelligently, with allocations reflecting superior alignment of tasks to both business value and skill suitability.

Task spillovers decreased from 14% to 6% demonstrating improved sprint predictability, as more tasks are completed on time. Lastly, the system requires just 1.94 seconds to generate a plan, which is negligible compared to manual planning effort that takes 3-4 hours.

The percentage improvement analysis in Figure 2 above highlights the magnitude of gains achieved by the hybrid framework. The most significant improvement is observed in planning time, which is reduced by approximately 99.98%, showing a drastic reduction from several hours of manual effort to near-instantaneous plan generation. This enables real-time or near-real-time sprint planning, improving responsiveness in dynamic agile environments.

Another substantial improvement is seen in task spillover rate, as it is reduced by approximately 57.14%, indicating a marked increase in delivery reliability. Workload variance also saw good improvement, as it's reduced by 27.78%. Moderate improvements are observed in the normalized objective score (+6.17%) and constraint satisfaction (+6.38%). As can be seen, while the traditional planning approach was already achieving reasonably good performance in task allocation quality and constraint adherence, the hybrid MILP-RL system still improves on it.

Overall, these results show that the hybrid MILP–RL system delivers significant operational gains in key areas critical to agile project success.

Table 4 shows the effect of the reinforcement learning (RL) agent on sprint planning. As can be seen, the RL agent gradually improves allocation over time. The Normalized Objective Score increases from 0.86 in the initial sprint without RL adaptation to 0.93 by the final iteration,

reflecting better quality allocations. Workload variance and task spillover rates generally decrease over the successive sprints, showing that the system becomes better at allocations that balances workloads and ensures timely completion of tasks.

Some metrics (such as task spillover and normalized objective score) worsen temporarily in Sprint 2. This reflects the stochastic nature of early learning, where the RL agent is exploring different allocation strategies before gradually improving over successive sprints.

By the final iteration, all metrics converge to improved values compared to the first system-optimized sprint without RL adaptation. This trend is further illustrated in Figure 3, where the normalized metrics provide a clear visual representation of the improvement trajectory across successive sprints.

The constraint satisfaction rate remains consistently at 100% across all iterations, indicating that the MILP solver enforces core planning constraints from the start. Solve times vary slightly between 1.34 s and 2.71 s across the simulated sprints, reflecting normal fluctuations in task distributions and solver behavior. So, the adaptive learning component does not significantly affect computational efficiency, and planning remains near-instantaneous.

5.2 Implications for Scaled Agile Practice

The results suggest that hybrid planning systems can significantly enhance decision support in large, distributed agile programs. The model's ability to generate optimized plans quickly while continuously learning from execution data can reduce manual coordination overhead and improve planning consistency across teams.

Also, the interpretability of optimization outputs addresses a common concern associated with purely learning-based approaches. And this makes the framework more acceptable for real-world adoption where transparency is essential.

The microservice-based implementation further indicates that the approach can be integrated into existing agile tool chains without requiring substantial infrastructure changes.

5.3 Limitations

Despite the promising results, some challenges remain.

- First, the evaluation was conducted using a single dataset and simulated execution feedback, which may not fully capture the variability of real industrial environments.
- Secondly, the reinforcement learning component was evaluated conceptually rather than through long-term deployment, limiting the ability to measure cumulative learning effects.
- Thirdly, the current model assumes reliable effort estimates and skill representations (but these may vary in practice).
- Finally, as the system relies on historical and inferred data, there is potential for bias in task allocation (such as reinforcing existing skill patterns). The system's use of automatically extracted data from multiple sources and the RL component's operation of continuously updating its policy based on observed outcomes can reduce this risk. However, such biases cannot be completely eliminated.

5.4 Comparison with Existing Literature

The results of this study align with previous research that show that optimization techniques can improve task allocation and workload balance in software project environments, and that reinforcement learning provides mechanisms for adaptive decision-making. However, as highlighted in Section 2, existing studies typically address these capabilities in isolation, either through static optimization models or standalone learning-based approaches.

The empirical findings demonstrate that the proposed hybrid MILP–RL framework achieves both high allocation quality and operational feasibility under realistic constraints, while also enabling adaptive refinement across sprints. This integrated performance directly addresses the limitations identified in the literature, particularly the lack of frameworks that simultaneously ensure feasibility, interpretability, and adaptability in large-scale agile planning.

5.5 Research Directions

To strengthen this framework, future research should consider:

- Incorporating multi-objective optimization to explicitly balance value, risk, and technical debt
- Exploring deep reinforcement learning methods for richer state representations

6. Conclusion

This study proposed a hybrid sprint planning and task allocation framework that integrates Mixed-Integer Linear Programming (MILP) with reinforcement learning (RL) to support decision-making in large-scale agile environments. The aim was to overcome the limitations of existing approaches by combining constraint-based optimization with adaptive learning from execution feedback.

The results show that the framework consistently outperforms traditional project management approaches across key sprint planning metrics (such as task allocation quality, workload balance, and delivery reliability) while maintaining near-real-time computational performance. The optimization component achieved feasible and high-quality allocations under complex constraints, while the RL feedback mechanism progressively refined planning decisions across successive sprints. Together, these findings highlight the practical value of combining optimization and adaptive learning to improve both immediate planning outcomes and long-term performance in dynamic agile environments.

This work contributes to literature by introducing a unified, closed-loop MILP–RL architecture for agile project planning. It addresses key research gaps identified in prior studies, especially the lack of integrated systems that simultaneously ensure feasibility, scalability, and adaptability. From a practical perspective, the framework offers a scalable decision-support tool for distributed and multi-team agile projects, with potential to improve planning accuracy, reduce manual effort, and enhance predictability.

However, the evaluation relied on historical and simulated data, and therefore may not fully capture real organizational dynamics. Future work should focus on real-world deployment studies, multi-objective optimization extensions, and explainable RL techniques to improve transparency and practitioner trust.

Overall, the findings indicate that hybrid optimization–learning approaches provide a promising foundation for adaptive and intelligent sprint planning, bridging the gap between static planning models and the dynamic nature of modern agile software development.

Author Declaration on AI Use

I certify that I only used AI to help with language editing, grammar correction, readability, and structural clarity when writing this manuscript. No discoveries, interpretations, or conclusions were unique because of AI.

As the author, I am solely responsible for all research concepts, analytical effort, arguments, and intellectual contributions. AI was just utilized to assist with the writing, and I maintain the work's originality and academic integrity.

References

1. Alshayban, A., & AlZahrani, A. (2021). Improving task allocation in agile teams with workload balancing mechanisms. *International Journal of Advanced Computer Science and Applications*. 12(6), 214–222.

2. Azevedo, B. F., Rocha, A. M. A. C., & Pereira, A. I. (2024). Hybrid approaches to optimization and machine learning methods: A systematic literature review. *Machine Learning*, 113, 4055–4097. https://doi.org/10.1007/s10994-023-06467-x

3. Azonuche, T. I., & Enyejo, J. O. (2024). Exploring AI-powered sprint planning optimization using machine learning for dynamic backlog prioritization and risk mitigation. *International Journal of Scientific Research and Modern Technology*, 3(8), 40–57. https://doi.org/10.38124/ijsrmt.v3i8.448

4. De Medeiros, D. F., Mendonça, M. E., & de Souza, C. R. B. (2020). Task assignment in agile teams: A heuristic approach using skills and workload. *Empirical Software Engineering*. 25(5), 4041–4073.

5. Dybå, T., et al. (2011). Conceptual scheduling model and optimized release scheduling for agile environments. *Information and Software Technology*. https://doi.org/10.1016/j.infsof.2011.04.004

6. Elugbadebo, O. (2025). *Machine Learning-Driven Sprint Planning: Enhancing Predictability in Agile Development*. (unpublished manuscript).

7. ForouzeshNejad, A. A., Arabikhan, F., Gegov, A., Jafari, R., & Ichtev, A. (2025). Data-Driven Predictive Modelling of Agile Projects Using Explainable Artificial Intelligence. *Electronics*, 14(13), 2609.

8. Ghanbari, H., & Hoda, R. (2021). Investigating task allocation patterns in distributed agile teams. *Information and Software Technology*.

9. Golfarelli, M., Boschetti, M. A., Rizzi, S., & Turricchia, E. (2014). A Lagrangian heuristic for sprint planning in agile software development. *Computers & Operations Research*. 47, 70–81.

10. Highsmith, J. (2009). *Agile project management: Creating innovative products* (2nd ed.). Addison-Wesley Professional.

11. Karam, A., Attia, E.-A., & Duquenne, P. (2017). *A MILP model for an integrated project scheduling and multi-skilled workforce allocation with flexible working hours*. In 20th IFAC World Congress,

Volume 50, Issue 1 (pp. 13964–13969). IFAC-PapersOnLine. https://doi.org/10.1016/j.ifacol.2017.08.2221

12. Kocaguneli, E., Menzies, T., & Bener, A. (2012). Software effort estimation with optimism and pessimism: A literature review. *IEEE Software, 29*(5), 20–27.

13. Jiang, Y., Lu, X., & Zhang, Y. (2019). Artificial intelligence application in software project management: A review. *IEEE Access*, 7, 51566–51583.

14. Larman, C., & Vodde, B. (2010). *Large-scale Scrum: More with LeSS*. Addison-Wesley.

15. Menzies, T., & Zimmermann, T. (2023). Software analytics in the era of large language models: Opportunities and threats. *Communications of the ACM,* 66(11), 82–91.

16. Mohan, C., & Ahire, A. (2019). Multi-criteria decision making for agile sprint planning. *Journal of Software: Evolution and Process*. 31(6), e2169.

17. Prabhneet, K., & Sharma, A. (2024). Agile Effort Estimation Using Machine Learning – A Systematic Review. *IITM Journal of Management and IT*. 15(1), 45–58.

18. Racheva, Z., et al. (2025). BPriS: Disciplined agile delivery planning method based on work items list pattern applied to prioritized semantically coupled software functions. *Applied Sciences*, 15(9), 5091. https://doi.org/10.3390/app15095091

19. Rigby, D. K., Berez, S., & Noble, R. (2023). Agile at scale: New evidence on transformation performance. *Harvard Business Review*, 101(4), 102–113.

20. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction (2nd ed.).* MIT Press.

21. Widodo, D. A., & Sutabri, T. (2025). Optimizing sprint planning in agile methodology using greedy algorithm. *International Journal Scientific and Professional*, 4(2), 536–540. https://doi.org/10.56988/chiprof.v4i2.86

22. Zhang, D., Dai, D., He, Y., Bao, F. S., & Xie, B. (2019). *RLScheduler: An automated HPC batch job scheduler using reinforcement learning*. arXiv. https://arxiv.org/abs/1910.08925

23. Zhao, Y., Wang, Y., & Li, X. (2022). Adaptive task scheduling using deep reinforcement learning in software project management. *Information and Software Technology*, 142, 106732.

About Author

Enoch Oghene-Mairo Omajeh

Port Harcourt, Nigeria

Enoch Oghene-Mairo OMAJEH is a Project Planner and a PhD candidate in Information Systems Engineering in the University of Port Harcourt, Nigeria (expected in 2027).

With vast experience leading project execution and digital transformation, his interest lies in the intersection of Project Management and IT, particularly Artificial Intelligence. His focus is to contribute to the development of intelligent systems that support project decision-making, enhance execution efficiency, and improve overall project delivery performance. He can be contacted at eomajeh@gmail.com