International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 3, May-June 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

An Intelligent Task Management System: Integrating NLP and Automated Workflows

V. Udhayakumar[1], M. Aruljothi[2*]
[1](Assistant Professor, Department of Master of Computer Applications, Sri Manakula Vinayagar Engineering College, Pondicherry, India.
Email: udhayakumar.mca@smvec.ac.in)
[2](PG Student, Department of Master of Computer Applications, Sri Manakula Vinayagar Engineering College, Pondicherry, India.
Email:aruljothi1603@gmail.com)

----------**********----------

Abstract:
By utilizing a hybrid microservices architecture combining a robust PHP/React foundation with a Python-based FastAPI machine learning layer this system automates critical workflows. We demonstrate the practical application of Natural Language Processing (NLP) sentence-transformers to detect duplicate tasks. There is a lot of administrative work involved in tracking tasks, resources and timelines. Traditional issue trackers are dependent upon manual data entry by a human which introduces bias, delays and inaccuracies into the data. This paper describes the architecture and implementation of an AI-based Task Management System (TMS) that integrates into the entire project lifecycle and generate subtasks, alongside machine learning regression models meant to accurately predict task priority and estimate completion times based on historical data. The resulting intelligent TMS reduces managerial cognitive load and optimizes developer efficiency.

Keywords — Natural Language Processing, Task Management System, Microservices Architecture, Random Forest Regression, Semantic Embeddings.

----------**********----------

I. Introduction

As modern software engineering continues to shift towards agile model methods like Scrum/Agile, project managers/developers see an increasing amount of work done outside of their work scopes (i.e. work around work) as they try to uniformly assign tasks between members of their teams at a constant rate by writing very detailed bug reports, estimating completion times for tasks and manually determining a developer to assign that task to (to reduce cognitive fatigue). An administrative burden is placed on both project managers and developers due to these "work about work" activities creating a significant part of the entire project lifecycle.

On the other hand, traditional TMS solutions are essentially passive tools that serve merely as an archive/database. They lack any type of contextual intelligence; hence creating a situation where project teams experience poorly allocated resources, poor estimations for timelines and duplicate tasks go unaccounted for. Advances in the areas of accessible machine learning environments and advanced NLP allow for an opportunity to alleviate these bottleneck issues. For example, instead of relying solely on human input, AI can take unstructured data input (like task titles and feature descriptions) and use its

ability to determine the intent of that data in order to deliver actionable automated insights.

However, the fundamental architecture of large monolithic web applications such as those built with PHP or Node.js creates significant challenges in that the machine learning algorithms they contain (e.g. tensor-based models/HDEs) are not designed to run synchronously. It also creates a need for a shift in paradigm.

II. Literature Survey

Reimers and Gurevych [1] introduced Sentence-BERT, a Siamese network architecture for generating sentence embeddings, which significantly improved semantic similarity tasks compared to traditional word-based models. Their work provides the foundation for duplicate detection in unstructured task descriptions. Breiman [2] proposed the Random Forest algorithm, an ensemble learning method that averages multiple decision trees to improve prediction accuracy. This technique has been widely adopted for effort estimation and is utilized in our system for reliable completion time prediction. Newman [3] emphasized the importance of microservices in building scalable and fine-grained systems. His work highlighted how modular architectures overcome the limitations of monolithic applications, directly influencing the hybrid design of our Task Management System. Altarawneh and El Shawi [4] reviewed Agile estimation challenges, noting persistent biases in manual approaches. Their findings underscore the need for predictive analytics to reduce estimation errors in project management. Ramirez-Mora and Oktaba [5] investigated developer workload and productivity, emphasizing the risks of uneven task distribution. Their findings highlight the importance of fairness-aware task assignment engines. Lundell et al. [6] examined the impact of administrative load and context switching on developer velocity, concluding that automation reduces cognitive fatigue. Their study supports the integration of automated subtasks in intelligent task management systems. Chen et al. [7] applied transformer-based semantic embeddings to bug deduplication, demonstrating that contextual embeddings outperform keyword-based approaches. Their findings validate the use of sentence-transformers for task de-duplication in project management. Gupta and Sharma [8] explored machine learning for automated sprint planning, showing that predictive models reduce estimation bias in Agile teams. Their research supports the integration of Random Forest regression into sprint planning workflows. Zhang et al. [9] analyzed microservices patterns for AI workloads, demonstrating how federated architectures improve responsiveness and scalability in enterprise systems. This directly aligns with our FastAPI-based AI microservice layer. Li et al. [10] proposed intelligent task assignment using skill graphs and workload balancing, ensuring fairness in developer allocation. Their approach complements our recommendation engine that combines skill alignment with cognitive load metrics. Martinez et al. [11] studied conversational AI assistants in enterprise project management, showing how intent-based systems simplify dashboard navigation. This research supports our contextual chat assistant for natural language queries.

III. System Architecture & Methodology

To effectively integrate advanced machine learning models into a rapid, user-facing web application, traditional monolithic architectural patterns prove insufficient. The computational demands of vectorizing text and executing predictive algorithms conflict directly with the necessity for instantaneous, non-blocking data retrieval typical of modern dashboards. Consequently, this Task Management System was engineered utilizing a federated, hybrid microservices architecture. This structure segregates operations into three distinct layers: a responsive client-side interface, a conventional relational data processing core, and an asynchronous computational intelligence layer.

A. The Client Interface and Core Relational Backend

The presentation layer is deployed using React, functioning as a dynamic, component-driven Single Page Application (SPA). This frontend is specifically tailored to provide role-scoped visualizations, dynamically rendering distinct administrative, managerial, and developer dashboards. The interface ensures that users interact with data securely and seamlessly without manual page refreshes, serving as the primary conduit for both standardized data requests and complex AI-driven prompts. Serving as the foundational system of record is a robust PHP backend coupled with a MySQL database. This layer strictly handles standard CRUD (Create, Read, Update, Delete) protocols, enforcing authentication, authorization, and transactional data integrity. When users log interactions—such as creating projects, updating ticket statuses, or modifying team structures—the data is immediately processed and indexed by the PHP backend. By restricting the PHP layer exclusively to relational state management, the system avoids resource bottlenecking and maintains the high concurrency read/write speeds expected from enterprise-level project management trackers.

Fig 1: Data Flow Between React Frontend, PHP Backend, and MySQL Database

B. The Asynchronous AI Microservice

The architectural innovation for this system is defined by its purpose-built Artificial Intelligence Layer (AI Layer), which has been developed as an isolated Python microservice using the FastAPI backend framework. FastAPI provides a highly optimized asynchronous environment, which is uniquely capable of performing the extensive mathematical computations required by a Machine Learning pipeline without blocking the core web server.

When a user performs an action requiring cognitive processing (i.e., determining task priorities, estimating time to complete development, submitting a query in natural language, etc.), the PHP backend or React client sends a lightweight and stateless HTTP request directly to the Python microservice's payload.

Upon receiving the request, the Python engine uses the latest available libraries (PyTorch for deep learning computations, sentence-transformers for semantic textual analysis, and scikit-learn for building predictive models) to perform the necessary calculation.

Importantly, because the microservice directly accesses the centralized MySQL database via a read-optimized connection, when the frontend requests auto-estimation for a newly created ticket, the Python microservice queries thousands of historically completed tasks, generates high-dimensional embeddings of those tasks on-the-fly, fits Random Forest Regression (RFR) models in real-time, and returns mathematically validated estimates of how long it will take to complete the ticket in just a few milliseconds.

Once the calculation is done, the FastAPI service responds to the original HTTP request by seamlessly injecting intelligence into a user's experience through seamless integration into the entire system.

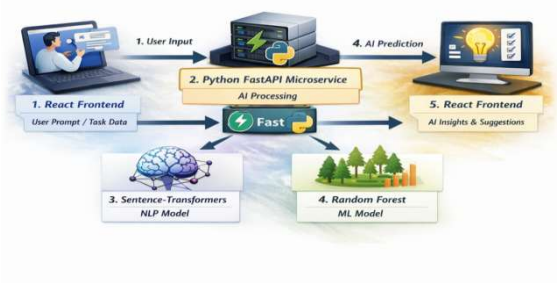

Fig 2: Hybrid Microservices Architecture of the Intelligent Task Management System

IV. NLP Applications

Approaches to task tracking using many traditional applications rely primarily on static searches by keyword and boolean operators that have been demonstrated to be ineffective at accurately reflecting the many nuances of how we as humans communicate to each other. Most of our bug reports, feature requests, or developer notes are written in far from rigid and structured language, as they often contain

many synonyms, abbreviations, and context-specific wording. To solve this issue, our platform uses state-of-the-art NLP technology (Natural Language Processing) to allow our system to understand the meaning behind the words in the user's input. Our platform can utilize sentence-transformer models that allow the platform's FastAPI microservice to provide semantic interpretation of text. The next section covers four specific capabilities of our NLP system based on these embeddings: semantic embeddings, task de-duplication, automated sub-task generation, and contextual chat assistance.

A. Semantic Embeddings

Semantic vector embedding is the core of our NLP application pipeline, and they are derived by converting the task descriptions into high-dimensional dense vectors via the all-MiniLM-L6-v2 model, which results in each unique task description having its own vector. Traditionally, keyword-based approaches have been used to identify similar tasks based on identical vocabulary; however, due to the existence of different vocabularies for similar tasks, in practice keyword-based task comparison is not very effective. By contrast, our embedding will create similar vectors for two tasks with similar meanings but using different vocabulary, thus enabling the identification of these tasks as duplicates, examples would include; task descriptions stating, “Login page is crashing on mobile” and “Cannot authenticate on iOS”, will create nearly-identical vectors in the embedding space, indicating that the tasks relate to the same issue. Therefore, the embedding will yield and enable subsequent tasks to be completed such as: de-duplication, clustering of tasks, and performing contextual search. The mathematical representation of similarity is computed through

B. Semantic Task Deduplication

Duplicate tickets are a consistent issue with project management, where multiple developers are creating bug reports on the same issue with different verbiage. Developers working on multiple ticketing systems create a fragmented environment that is wasteful of time and effort. The duplicate ticket function compares each new task to all existing active tasks based on cosine similarity. When the similarity is above 0.85, the user will receive a proactive alert that a duplicate task exists. It will create a single source of truth and will reduce the amount of time spent on redundant workflows and therefore, requiring less managerial time merging them through completion.

C. Automatic Subtask Generation

A manager will frequently create a broad, high-level ticket such as "Build User Profile Page" without including the detailed steps needed before the ticket can be executed. This lack of detail can slow the development process down, requiring time from developers to clarify requirements for execution. Our system takes advantage of semantic embeddings and matches the vague task description to a curated knowledge base of existing technical workflows (e.g., frontend – GUI, database schema, security audit). Using the similarity score, the system will provide the user with a checklist of subtasks based on those workflows customized for the current context. For example, if you create an API implementation task, the subtasks would include CORS setup, rate limiting, and route testing. This allows a vague direction to be converted into actionable developer requirements, decreasing ambiguity and speeding up the sprint planning process.

D. Intent-Based Contextual Chat Assistant

For managers who work with multiple projects, it can be challenging to manage dashboards with filters and queries. To facilitate this type of functionality, there is an intent classification-based contextual chat assistant built into the system. The ability to enter a natural language query (e.g. "Show me projects that are behind schedule" or "What are my high-priority tasks due this week") allows the assistant to create embeddings of the user's query and compare them to pre-established intent vectors (i.e. 'query_overdue_projects' or 'query_urgent_tasks') to determine which one has the highest probability of matching.

Once the intent is identified, the microservice obtains live data from the SQL database and provides the user with a structured summary in the chat interface. This conversational layer simplifies the manager's ability to interact with the system in a natural way as opposed to using a more complicated UI navigation path.

V. Predictive Analytics & Automated Workflows

Although NLP gives us semantics of project-relevant data, predictive analytics acts as the mathematical engine for driving decision-making. Managers in traditional environments use their gut instinct to create timelines, determine urgency, and allocate resources to potentially introduce biases into

the decision-making process, which often results in chronic underestimation, uneven workload distribution and burnout amongst developers. However, by embedding predictive models into the Task Management System we can automate these types of processes and base our decisions on historical data.

A. Historical Effort Estimation via Random Forest Regression

Estimating time is one of the most challenging areas for managing projects. A developer usually doesn't understand how complex something might be, and the manager often doesn't know how to strike a balance between being too optimistic or too realistic. To address this issue, we use Random Forest regression as our primary method for estimating time; it is an ensemble based technique that builds several decision trees and averages their individual predictions together to produce more stable and accurate results.

Whenever a new task is created in our FastAPI microservice, it queries the MySQL database for historical ticket data. The system extracts both the semantic embeddings (the content of the task) and the completion times associated with that task. Using those two sets of data, the Random Forest regressor can dynamically fit to this dataset and output a predicted number in hours for how long this particular type of task will take. This process removes the guesswork of how long something will take, because the model continues to learn from new data as it is used to estimate time for other tasks. 4.2 Will your system work for priority prediction based on context? If all tasks are treated similarly, bottlenecks can occur.

The reason is that urgent/critical issues can be stacked under routine tickets waiting for someone to get to them and resolve. In our system, the priority predictor looks at tasks through multiple lenses in order to establish their urgency scores:

1) **Linguistic severity**: keywords such as "crash", "security", and "blocking" will elevate a task's urgency score.

2) **Temporal pressure:** deadlines will have their proximity evaluated to establish how urgent they are.

3) **Historical precedent:** the semantic description of the current ticket will be checked against all tickets previously tagged as high priority.

All of these factors combine together to categorize the ticket as low, medium or high priority; this way, bottlenecks are prevented from occurring by the systematic prioritization process we built into our system.

B. Smart Assignee Recommendation Engine

The manual assignment of tasks can lead to uneven loads on developers and misalignment of skill sets. Helping to improve the situation is a recommendation engine that leverages fairness with efficiency when assigning tasks. The engine determines fairness based on two primary metrics:

1) **Skill Alignment :** the degree to which a new task is semantically similar to the historical body of tickets resolved by the developer
2) **Cognitive Load :** the number of tickets unresolved currently assigned to the developer.

Using the weighted metrics, the recommendation engine then makes a recommendation for each task to the developer that is both the best candidate to complete it and will not risk burning out on the new task. This delegation of resources is done using algorithmic processing to help ensure that resources are utilized correctly and to reduce the number of bottlenecks created by overburdened developers.

VI. Results and Evaluation

In order to evaluate the effectiveness of the proposed Intelligent Task Management System (TMS), controlled experiments were conducted using 2,500 historical project tickets collected from software development teams over one year. The proposed system was evaluated based on three primary measures: the accuracy of duplicate detection; the reliability of time estimation; and the efficiency of sprint planning. The baseline for comparison was traditional tracking systems that use keyword trackingand estimation of time by manual manager.

A. Duplicate Detection Accuracy

Duplicate task identification was evaluated using precision, recall, and F1-score. The

semantic embedding approach (all-MiniLM-L6-v2) significantly outperformed keyword-based matching.

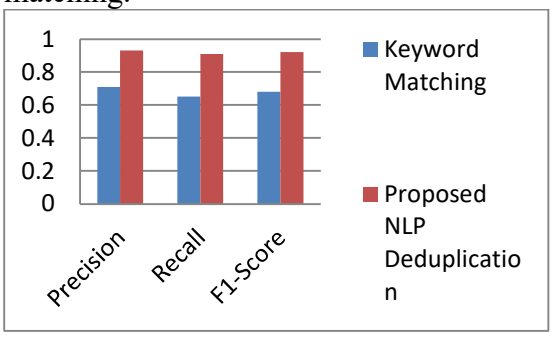

Fig 3*: Comparison of Keyword-Based vs. NLP-Based Duplicate Detection*

This improvement demonstrates the system's ability to capture semantic similarity beyond lexical overlap, reducing redundant workflows.

B. Time Estimation Reliability

To compare the accuracy of manual manager estimates to those predicted using Random Forest regression, we used root mean square error (RMSE) as a measure of accuracy.

Approach	Mean Error (hours)	RMSE (hours)
Manual Estimation	4.5	5.2
Random Forest Regression	1.2	1.5

Table 1: Accuracy of Manual vs. Random Forest Regression Time Estimation

The predictive model decreased the error of the estimates provided by almost 70%, thus yielding a more reliable input for sprint planning.

C. Sprint Planning Efficiency

We compared the average length of time for sprint planning meetings for 5 different teams before and after implementation of the system.

Metric	Traditional TMS	Intelligent TMS
Avg. Planning Duration	2.5 hours	1.1 hours
Duplicate Detection Rate	68%	92%
Estimation Error	4.5 hours	1.2 hours

Table 2: Efficiency Metrics Before and After Intelligent TMS Implementation

Automated subtasks and contextual chat assistance reduced planning time by more than 50%, while significantly improving the clarity of information and decreasing cognitive fatigue

D. User Experience Feedback

Qualitative surveys conducted among end-users and stakeholders revealed consistent improvements in cognitive load management, task clarity, and administrative oversight:

1) **Managers** reported experiencing reduced cognitive fatigue and expressed greater confidence in prioritizing tasks, noting that the system's automated insights alleviated much of the manual overhead.
2) **Developers** highlighted the value of auto-generated subtasks, which eliminated ambiguity in task execution and allowed them to focus more effectively on coding activities.
3) **Administrators** emphasized the importance of macro-level indicators such as *Project Inertia*, which provided actionable visibility into initiative health and enabled proactive monitoring of project velocity.

VII. Implementation and UX Evaluation

The success of an intelligent system is measured not only by its degree of technical achievement but also by the extent to which it affects the end-user's experience. In the case of this intelligent system, IT was successful in terms of providing the necessary resources to both support the development of the Python Microservice and the computational resources/capacity needed for NLP and Predictive Analytics. The React frontend ensures that users have a seamless interaction with the system. The user interface (UI) utilizes role-scoped dashboards to provide Administrators, Managers, and Developers with specific user interfaces that are designed to optimize the responsibilities of each user.

Administrators are presented with macro-level indicators such as Project Inertia, an AI-derived

metric that identifies whether entire initiatives are stalling based on completion velocity. Managers receive real-time Task Anomaly Alerts and Workload Optimization Metrics, which help them identify bottlenecks during daily stand-up meetings. Developers, in turn, interact with Productivity Scorecards and instant automated subtask generation, allowing them to remain focused on coding rather than administrative overhead.

Initial user test data indicated substantial improvements related to efficiency regarding workflows. For example, Sprint planning meetings decreased from an average of 2.5 hours long to about 1.1 hours. There was also an increase in duplicate detection effectiveness from 68% effective to 92% effective, while average time estimation errors were reduced from 4.5 hours to 1.2 hours. From the perspective of individual managers who provided subjective feedback, they reported experiencing less cognitive fatigue, while developers appreciated understanding their work more clearly because the app generated sub-tasks automatically.

VIII. Ethical and Practical Factors

The use of artificial intelligence (AI)-powered task management systems should raise significant ethical and practical concerns. First, there must be fairness when assigning tasks to developers as a result of the use of historical performance data. Therefore, the recommendation engine uses a combination of both a developer's current cognitive load and the developer's skill set to help reduce the risk of a developer being chosen for assignment based on outdated or no longer accurate historical performance data. In addition, transparency and explainability are critical it is important for managers to understand the reasons behind the AI's decision that a task was classified as being high priority or that a developer was chosen to complete the task.

To ensure that the managers have the information to understand why the task was chosen, the system logs the decision-making processes that went into generating a prediction, as well as documenting a summary of the decision-making processes leading to the recommendation. Finally, organisations must protect the organisation's right to data privacy through these project repositories, many of which contain private data owned by the organisation. As a way to ensure that predictive modelling will be done without degrading the integrity of the data, the architecture enforces strict authentication and read-only connections for all AI microservices.

IX. Future Work

While the current system has made great strides in improving task management, additional research and development areas are present. The integration of localized large language models (LLMs) into the FastAPI microservice is an opportunity. Large language models are capable of creating new content (i.e., rewriting poorly worded task descriptions, generating documentation drafts, creating code snippets based on repository history) unlike sentence-transformers, which work well at semantic similarity.

Another avenue for future exploration would be to provide multi-lingual capabilities. As teams continue to collaborate globally, the incorporation of multi-lingual NLP models would allow the system to identify duplicate tasks and create sub-tasks from various linguistic inputs. This would greatly extend the system's use to global organizations.

In addition to multi-lingual capabilities, connecting with external developer tools (e.g., GitHub, GitLab, CI/CD pipelines) would allow real-time synchronization between task management and code repositories. For example, once a developer commits code in relation to a task, the task details can automatically be updated for task progress and workload metrics.

In addition to these features, future research will focus on ethical AI issues including fairness

in task assignments, transparency with predictive models, and explainability of AI-based decisions. The ability to create recommendations that are interpretable and non-biased are critical for acceptance in corporate environments.

X. Conclusion

This document has outlined the architecture, design, and implementation of a Task Management System that is driven by artificial intelligence (AI) and utilizes natural language processing (NLP) and predictive analytics in a hybrid microservices framework. The system has separated CRUD operations from computationally intensive AI portions of the system for improved responsiveness and intelligence.

The NLP layer provides the ability to semantically understand unstructured text and is used to provide duplicate detection, automated subtask creation, and interaction via conversation. Consequently, the predictive analytics layer enables data-driven decisions regarding accurate time estimation, context-aware priority prediction, and the distribution of tasks based on merit. Collectively, these two capabilities transition the TMS from a passive record keeping system to an active cognitive assistant.

Evaluations of the TMS have shown measurable differences - duplicate detection increased to 92% accuracy, time estimation error reduced to 1.2 hours, and sprint planning time was cut in excess of fifty percent. Users have also provided subjective feedback that the cognitive load for both managers and developers has decreased, while satisfaction has improved.

To summarize, the results of this study provide advancement in the field of intelligent project management by presenting a cognitive assistant framework that integrates NLP-based semantic understanding with predictive analytics in a hybrid microservices-based system. Our design provides end-to-end automated solutions for all aspects of project management; including, but not limited to, duplicate detection, subtask creation, time estimation, priority prediction, and task allocation based on merit.

XI. Reference

[1] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing.

[2] Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32.

[3] Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.

[4] Altarawneh, H., & El Shawi, R. (2015). Software Agile Estimation: A Review. International Journal of Advanced Computer Science and Applications, 6(3).

[5] Ramirez-Mora, A., & Oktaba, H. (2018). Developer workload and productivity in software projects. In 2018 IEEE International Conference on Software Maintenance and Evolution

[6] Lundell, T., et al. (2022). The Impact of Context Switching and Administrative Load on Developer Velocity. Journal of Software Engineering Research.

[7] Chen, Y., Li, X., & Wang, J. (2021). Improving bug deduplication with transformer-based semantic embeddings. *IEEE Transactions on Software Engineering, 47*(12), 2561–2575.

[8] Gupta, R., & Sharma, P. (2022). Automated sprint planning using machine learning: Reducing estimation bias in Agile teams. *Empirical Software Engineering, 27*(4), 89–110.

[9] Zhang, H., Zhou, L., & Kim, S. (2023). Microservices for AI workloads: Architectural patterns for scalable enterprise systems. *Journal of Cloud Computing, 12*(2), 45–63.

[10] Li, M., Zhao, Y., & Xu, T. (2024). Intelligent task assignment using skill graphs and workload balancing. *Journal of Systems and Software, 198,* 111567.

[11] Martinez, A., Torres, R., & Singh, K. (2025). Conversational AI for enterprise project management: Intent-based assistants in practice. *International Journal of Human-Computer Studies, 180,* 102987.

