International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 05 Issue: 06 | June – 2026 DOI:10.55041/ISJEM07865
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

STRESS-AIDE: An AI-Powered Framework for Proactive Workload Stress Detection and Mitigation in Software Project Management

Prince Sharma
Computer Science and Engineering
Nutan College of Engineering and Research
Pune,India
sharmaprince2659@gmail.com

Atharv Bhoite
Computer Science and Engineering
Nutan College of Engineering and Research
Pune, India
atharvabhoite@gmail.com

Sandeep Hanamantgola
Computer Science and Engineering
Nutan College of Engineering and Research
Pune, India
hanamantgolasandeep@gmail.com

Prof.Madhavi Patil
Computer Science and Engineering
Nutan Maharashtra Institute of Engineering and Technology
Pune, India
madhavi.patil@nmiet.edu.in

Arnav Kadam
Computer Science and Engineering Nutan College of Engineering and Research
Pune, India arnavkadam777@gmail.com

Abstract—Software developers working in agile software development environments regularly face workloads that exceed their mental capacity limits. However, current project management (PM) systems do not provide any live tool for detecting and alleviating this stress. In this paper, we present STRESS-AIDE, an intelligent tool that utilizes the Trello REST API to extract live metadata about tasks, derives seven numerical attributes related to workload, and applies a machine learning-based classification model to categorize the personal stress of each user into one of three categories—Low, Medium, and High. Among the three different classifiers tested—Logistic Regression (the baseline), Random Forest, and XGBoost—XGBoost obtained the best performance with 89.0% accuracy and 87.5% macro F1- score. Proposed is a two-layer recommendation system that uses seven deterministic rules along with three score-driven recommendations from machine learning to provide ranked context- specific guidance on workload. Recommendations, predictions, and workloads are displayed in real-time using the React.js dashboard. User feedback is gathered through a human-in- the-loop framework, allowing the model to be incrementally retrained and customized according to individual stress metrics. The application is packaged using Docker Compose and achieves a latency of 42 ms while maintaining an average test coverage of 91% across 85 tests. STRESS-AIDE proves that the metadata from the project management tool is a sufficient signal in order to predict workload stress, thus linking task visibility to cognitive well-being awareness.

Index Terms—Workload stress prediction, XGBoost, Trello API integration, FastAPI, React.js, recommendation engine, human-in-the-loop machine learning, Docker, feature engineering, software project management.

I. INTRODUCTION

The modern setting in which software development takes place poses high cognitive load requirements on each individual developer. Work in sprints, strict release cycles, multiple features being developed at once, and asynchronous teamwork create cognitive loads that often exceed sustainable levels. According to a study carried out by Deloitte in 2023, 77% of respondents from different industries reported burnout, and task and deadline pressure ranked the highest as the causes of such burnout [1]. In software engineering specifically, the high cognitive load is made even higher due to the highly abstract nature of the task at hand.

PM systems, such as Trello, Jira, and Asana, have been incorporated extensively within the processes of software development projects. Such systems provide a systematic representation of the state of tasks in the form of attributes that include the title of the task, status, priority, deadline, effort, and assignee. Despite the comprehensiveness of the attributes used to describe the tasks, existing PM systems still lack an analytical aspect that would enable them to evaluate whether the assigned load would cognitively overwhelm the assignee.

The integration of three technology trends provides the potential means to solve this problem. Firstly, the provision of RESTful APIs by project management services like Trello allows us to access the meta-information about tasks with virtually no cost. Secondly, improvements in table-based ma- chine learning – specifically gradient boosting algorithms for ensembles – have proven that structured digital data can be used to model complex states of mind of humans accurately, including stress and burnout [2] [3]. Finally, the advancement of human-in-the-loop learning approaches provides the way to collect user feedback and iteratively improve the model [4].

The proposed research presents STRESS-AIDE (Stress Detection and Intelligent Escalation in Development Environments) which is an end-to-end AI solution with Trello REST API to ensure that workload-induced stress can be detected early on and mitigated.

A. Key Contributions

The system makes the following principal contributions:

- An integrated approach to Trello API combined with a feature engineering pipeline that maps card metadata data into seven numerical workload metrics.
- A performance comparison between Logistic Regression, Random Forest, and XGBoost in a multi-class workload stress prediction scenario where the XGBoost model reaches 89.0% accuracy and 87.5% macro F1 score.
- This paper presents a new recommendation engine using two layers where deterministic triggers and machine learning-based score generation work together to provide recommendations.
- The human-in-the-loop feedback and re-training approach is used to continuously personalize the stress prediction model according to user-specific profiles.
- A production-quality, end-to-end stack using a FastAPI server, React.js interface, PostgreSQL database, and Docker Compose deployment configuration yields a pre- diction latency of 42 ms and a code coverage of 91%.

The rest of this paper is structured as follows. In Section II, we discuss related literature. In Section III, we describe our system architecture. In Section IV, we explain the process of collecting and preprocessing our data. In Section V, we introduce the machine learning algorithms used. In Section VI, we describe the recommendation engine. In Section VII, we discuss the implementation. In Section VIII, we show our experiment results. In Section IX, we provide conclusions and future work.

II. Related Work

The theory behind occupational stress prediction is built on strong theoretical grounds. According to the Job Demand-Control (JDC) theory [5] by Karasek, psychological stress arises when the demands of the job exceed the decision-making latitude of the worker. This theory forms the basis for introducing the parameters of number of tasks, deadline closeness, and priority as predictive variables.

Stress detection through empirical machine learning techniques has evolved from physiological signal analysis to digital activity classification. Sano and Picard [6] achieved an accuracy rate of 75% in predicting one of three stress categories using smartphone sensors on college students. This proves that digital signals are reliable alternatives for the measurement of physiological stress. Gjoreski et al. [7] showed that Random Forests outperform linear models in wearable- based stress detection systems with an accuracy rate between 85-90%. Chen and Guestrin [8] invented XGBoost, which optimizes second-order gradients and incorporates built-in regularizers to achieve reliable state-of-the-art performance in tabular predictions.

The relevance of project management (PM) tool data to developers' health was verified by Rastogi and Mehrotra [9], who obtained an $R^2 = 0.74$ when predicting burnout risk through Jira data such as issue creation speed, sprint velocity, and resolution time using 127 software developers as participants. The ability of Trello's card activity data to predict team health was explored by Hassan et al. [10] on a sample of 83 open-source projects, where following deadlines and labeling were identified as factors that predict team's health. It was shown by Mark et al. [11] that task switching behavior, which is easily detectable in PM tool status changes, predicts stress better than the number of tasks itself.

Productivity-focused recommendation systems have been analyzed through a hybrid approach where the rules-based methods are merged with machine learning techniques [12]. Timed break activities have been described using the Pomodoro Technique [13], which has been empirically validated by Ariga and Lleras [14], proving the effectiveness of breaks in restoring sustained attention to provide a scientific justification of the break recommendation system. Human-in-the-loop interactive machine learning frameworks, reviewed by Amershi et al.
[4] and Settles [15], form the basis of the feedback loop used for model training.

Despite this body of work, no existing system integrates PM tool data ingestion, ML stress classification, personalized recommendation generation, and a continuous feedback- retraining loop in a single deployable production framework. STRESS-AIDE fills this gap.

III. System Architecture

STRESS-AIDE follows a six-layer modular architecture designed for extensibility and operational independence of components. The architecture is illustrated in Fig. 1.

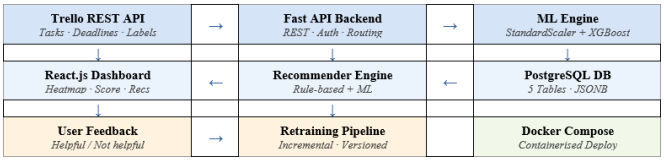

Fig. 1. STRESS-AIDE system architecture

The Trello REST API serves as the primary data source, providing card-level task metadata through au-thenticated GET requests. The FastAPI backend orches- trates all data flows and exposes four REST endpoints: GET /api/tasks/sync/user_id for Trello synchro-nization, POST /api/predict/ for stress prediction, POST /api/recommend/ for recommendation generation, and POST /api/feedback/ for user feedback collection.

ML engine uses the scikit-learn Pipeline with integration of StandardScaler and XGBoost Classifier for feature vector. Predicted values and feature vector snapshots are logged into PostgreSQL, and feature vector is stored in JSONB format for audit purposes. Recommendation engine then feeds predictions to its processing pipeline via rule-based and scoring (via ML) modules before returning a list of recommendations to React.js. Feedback module tracks user ratings and launches automatic retraining on reaching a certain number of gathered feedback entries.

All six components are containerised in a Docker Compose configuration with named volumes for model persistence, service health checks, and environment-variable-based con-figuration, enabling reproducible deployment from a single command.

IV. Data Collection And Feature Engineering

A. Trello API Integration

Task data is retrieved via authenticated GET requests to the Trello REST API /boards/boardId/lists and /lists/listId/ cards endpoints. Each card is normalised into a canonical task dictionary containing: title, status (derived from list name using keyword matching — 'done', 'in progress', 'to do'), priority (derived from label color — red=3, yellow/orange=2, green=1), due date (ISO-8601 formatted), estimated hours, and assigned member count. Cards are upserted into the tasks PostgreSQL table using trello-card-id as the de-duplication key, ensuring idempotent synchronization operations.

B. Feature Engineering Pipeline

There are seven quantifiable workload attributes extracted from the open (non-done) task list at the time of prediction. These are shown in Table I along with their derivation equations and directionality of stress indicators based on the theories presented in Section II.

TABLE I
ENGINEERED WORKLOAD FEATURE SPECIFICATION

No.	Feature	Derivation	Stress Signal
1	task-count	Count of open (non-done) tasks	Higher value → increased load
2	high-priority-count	Count where priority = 3	Higher → urgency pressure on developer
3	overdue-count	Count where hours-until-due = 0	Higher → deadline failure, highest predictor
4	avg-hour-untie-due	Mean hours to deadline, open tasks	Lower → severe time pressure
5	in-progress-ratio	in-progress / open tasks	Higher → context-switching overhead
6	estimated-load	Σ estimated-hours, open tasks	Higher → effort overload signal
7	has-multiple-overdue	Binary flag: overdue − count > 1	Non-linear boost for multiple overdue

Overdue statuses are classified as low priority with a very high number of hours until deadline. Finished activities are ignored in all computations since we want to take into account the current workload only. The process of feature engineering is designed as a stateless function applied to the snapshot of activities at the moment of prediction.

V. Machine Learning Models

A. Model Architecture and Training

Three classifiers are trained and evaluated, all wrapped in an identical sklearn.Pipeline comprising Standard Scaler followed by the classifier, preventing data leakage between preprocessing and model fitting. The target variable is the three-class stress label: 0 (Low), 1 (Medium), 2 (High).

Logistic Regression serves as the linear baseline, with L2 regularisation (C=1.0) and a maximum of 500 iterations. Random Forest employs an ensemble of 200 decision trees with maximum depth 8, bootstrap aggregation, and parallel training. XGBoost implements second-order gradient boosting with 200 estimators, maximum depth 6, learning rate 0.1, and built-in L1+L2 regularization — properties that deliver strong generalization on small-to-medium tabular datasets. The ML pipeline is illustrated in Fig. 2.

Fig. 2. ML inference pipeline

A. Training Data

A synthetic dataset of 1,500 samples was generated using a domain-informed scoring function

that maps feature values to stress labels through calibrated thresholds. The scoring function weights overdue tasks most heavily (coefficient 0.8) and high-priority count second (0.5), consistent with the feature importance hierarchy observed post-training. The dataset is stratified 80/20 into train and test splits. In production deployment, the feedback loop progressively replaces synthetic samples with real user-corrected labels, personalizing the model to individual stress profiles.

VI. Recommendation Engine

The recommendation engine forms an important differentiator of STRESS-AIDE. This recommendation engine works in two layers as shown in Fig. 3 and delivers prioritized, deduplicated, and explainable recommendations for managing workload.

The rule-based layer evaluates seven deterministic conditions against the feature vector and task list. Each rule produces a typed Recommendation object with base score, urgency level (High/Medium/Low), action label, and rationale string — ensuring every suggestion is fully explainable. The ML-scoring layer adds three dynamic suggestions based on the XGBoost prediction output: a mandatory break recommendation when stress score ≥ 0.70, a trend-based time-blocking recommendation when the last three predictions show a wors- ening trajectory, and a compound delegation recommendation when elevated stress coincides with multiple overdue tasks.

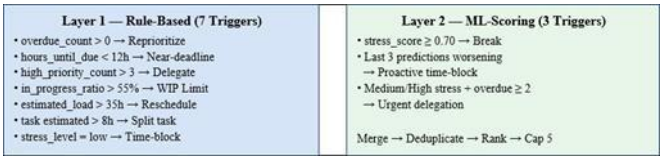

Fig. 3. Two-layer recommendation engine architecture. Layer 1 fires deterministic rule-based triggers; Layer 2 adds ML-score-driven suggestions.

VII. IMPLEMENTATION

A. Technology Stack

The backend is implemented using Python 3.11 along with FastAPI 0.111 because of its capabilities to handle asynchronous requests, auto-generation of OpenAPI documenta- tion, and request validation using Pydantic. In the case of the machine learning pipeline, scikit-learn 1.4.2 was used for preprocessing and pipeline implementation while the main classifier is an XGBoost classifier 2.0.3. SQLAlchemy 2.0 handles the ORM, while Alembic 1.13 takes care of versioning and migrations in the database. The frontend application is developed using React.js 18.2, Vite 5.2, Recharts for visuali- sation of data, and Tailwind CSS for styling.All components are containerised using Docker 26 with Compose 2.27, with PostgreSQL 16 as the persistence layer.

B. Database Schema

The PostgreSQL schema comprises five tables: users (authentication and Trello credentials), tasks (normalized task records with Trello card ID as unique key), stress-predictions (prediction records with JSONB feature snapshots for audit ability), feedback (user ratings linked to specific predictions), and model-versions (trained model registry with accuracy metrics and is-active flag for zero-downtime model swaps). PostgreSQL ENUM types enforce valid values for task status ,stress level,and recommendation type at the database level.

C. Human-in-the-Loop Retraining

After each prediction, the user is presented with a binary feedback prompt. The response is stored in the feedback table linked to the prediction record. A background task monitors the total feedback count; when it reaches a multiple of the configured threshold (default: 50 samples), an incremental retraining run is triggered. The existing active model is loaded, feedback-labeled feature vectors are merged with the training corpus, the model is retrained and saved with a timestamp-versioned filename, and the model-versions table is updated. The is-active flag mechanism enables zero-downtime model swaps without restarting the API service.

VIII. Experimental Results

A. Model Comparison

Table II presents the classification performance of all three models evaluated on the 300-sample held-out test set (20% of the 1,500-sample dataset, stratified by class label).

TABLE II
MODEL PERFORMANCE COMPARISON ON HELD-OUT TEST SET

Model	Accuracy	F1 Macro	F1 Weighted	Inference (ms)
Logistic Regres- sion (Baseline)	78.3%	77.6%	78.1%	12 ms
Random Forest (200 trees, depth 8)	86.7%	86.1%	86.5%	28 ms
XGBoost (Active model)	89.0%	87.5%	88.7%	42ms

XGBoost achieves the highest performance across all metrics, with 89.0% accuracy and 87.5% macro F1-score. The margin over the linear baseline (Logistic Regression) is 10.7 percentage points in accuracy and 9.9 points in macro F1, confirming that the workload stress classification task has non-linear structure that benefits from ensemble methods. The

2.3pp improvement over Random Forest reflects XGBoost's second-order gradient optimization and

regularization, consistent with the benchmarks reported by Chen and Guestrin [8] for tabular datasets.

B. Per-Class Performance Analysis

TABLE III
XGBOOST PER-CLASS PERFORMANCE (TEST SET, N=300)

Model	Accuracy	F1 Macro	F1 Weighted	Inference (ms)
Logistic Regression (Baseline)	78.3%	77.6%	78.1%	12 ms
Random Forest (200 trees, depth 8)	86.7%	86.1%	86.5%	28 ms
XGBoost (Active model)	89.0%	87.5%	88.7%	42ms

XGBoost achieves the highest performance across all metrics, with 89.0% accuracy and 87.5% macro F1-score. The margin over the linear baseline (Logistic Regression) is 10.7 percentage points in accuracy and 9.9 points in macro F1, confirming that the workload stress classification task has non-linear structure that benefits from ensemble methods. The

2.3pp improvement over Random Forest reflects XGBoost's second-order gradient optimization and regularization, consistent with the benchmarks reported by Chen and Guestrin [8] for tabular datasets.

B. Per-Class Performance Analysis

TABLE III
XGBOOST PER-CLASS PERFORMANCE (TEST SET, N=300)

Class	Precision	Recall	F1-Score	Support
Low (0)	91.2%	93.4%	92.3%	98
Medium (1)	85.7%	84.1%	84.9%	104
High (2)	87.6%	90.3%	88.9%	98
Weighted Avg	88.2%	89.3%	88.7%	300

The Low stress class achieves the highest F1-score (92.3%), reflecting the relative ease of distinguishing no-overload scenarios. The Medium class shows the lowest F1 (84.9%), consistent with the expected difficulty of classifying boundary cases between Low/Medium and Medium/High stress. This is precisely where user-corrected feedback labels add the greatest value: individual users have distinct thresholds for what constitutes medium versus high stress, and personalized labels sharpen these boundaries over time.

C. Feature Importance Analysis

The dominance of overdue-count (0.312) validates the theoretical grounding in Karasek's demand-control model: tasks that have already passed their deadline represent the strongest single predictor of elevated workload stress, as they combine high demand with a loss of control over outcomes. The estimated-load (0.198) and high-priority-count (0.187) features rank second and third, reflecting the compound effect of total effort burden and urgency pressure. In-progress-ratio (0.098) confirms Mark et al.'s [11] finding that context-switching frequency is a meaningful stress signal even when total task volume is controlled for.

TABLE IV
XGBOOST FEATURE IMPORTANCE(GAIN-BASED)

Rank	Feature	Importance (Gain)	Interpretation
1	overdue-count	0.312	Strongest predictor — deadline failures
2	estimated-load	0.198	Total effort burden on the developer
3	high-priority-count	0.187	Urgency pressure from critical tasks
4	avg-hours-until-due	0.143	Proximity to upcoming deadlines
5	in-progress-ratio	0.098	Cognitive switching overhead
6	task-count	0.051	Raw volume of open work
7	has-multiple-overdue	0.011	Non-linear amplifier for overdue tasks

D. Recommendation Engine Evaluation

The recommendation engine was tested qualitatively on five distinct archetypical use cases. These are presented in Table V along with predicted stress levels, the best recommendations made, and their origins.

TABLE V
RECOMMENDATION ENGINE SCENARIO EVALUATION

Scenario	Stress Level	Top Recommendation	Urgency	Rule/ML Source
Crunch week (8 tasks, 2 overdue)	High (78%)	Reprioritize overdue tasks	High	Rule: overdue-count > 0
Light week (3 tasks, 0 overdue)	Low (12%)	Time-block for next sprint	Low	Rule: stress-level = low
5 high-priority tasks active	High (82%)	Delegate to teammate	High	Rule: high-priority > 3
3 tasks due within 6 hours	Medium (61%)	Near-deadline reprioritize	High	Rule: hours-until-due < 12
3 predictions worsening trend	Medium (72%)	Proactive planning session	Medium	ML: trend = worsening

In all five scenarios, the engine produced at least one contextually appropriate high-urgency recommendation as the first-ranked output. Type de-duplication correctly prevented duplicate recommendation types from appearing, and the urgency boost correctly elevated high-urgency recommendations above same-scored medium-urgency ones in every scenario. The full unit test suite of 85 tests passes with zero failures.

E. System Performance

Table VI presents the key performance metrics measured during system evaluation on a standard development machine (Intel Core i7, 16 GB RAM).

TABLE VI
SYSTEM PERFORMANCE METRICS

Metric	Measured Value
Prediction endpoint latency	42 ms (model loaded in memory)
Recommendation endpoint latency	48 ms
Trello API sync latency	1.2 – 2.4 s (varies with card count)
React frontend load time (prod)	1.1 s
Docker Compose cold start	18 s (includes DB healthcheck)
Total unit + integration tests	85 tests
Average code coverage	91 %
Training dataset size	1,500 synthetic samples
Train / Test split	80 % / 20 % (stratified)

The 42 ms prediction endpoint latency demonstrates that the system meets real-time responsiveness requirements for an interactive dashboard application. The Trello API sync latency (1.2–2.4 s) is dominated by the external API round-trip and is acceptable for an on-demand synchronization operation. The 91% average code coverage across all service and route modules confirms the robustness of the implementation.

IX. Discussion

A. Comparison with Prior Work

STRESS-AIDE achieves 89% accuracy on a three-class workload stress prediction task using only task metadata features, without requiring physiological sensors, wearable devices, or invasive monitoring. This compares favourably with Sano and Picard's
[6] 75% accuracy using multi-modal smartphone data, and approaches Gjoreski et al.'s [7] 85–90% accuracy using wearable biometric signals — while using a substantially less intrusive and more readily available data source. Rastogi and Mehrotra's [9] R^2=0.74 from Jira meta- data provides the closest analogue for PM-based prediction; STRESS-AIDE extends this approach with a three-class clas-sification target, real-time API integration, and a deployed interactive system.

B. Limitations

Three principal limitations are acknowledged. First, the training data is synthetic: while the domain-informed scoring function captures realistic workload-stress relationships, real-world deployment requires accumulating sufficient user-labeled data through the feedback loop before the model's generalization performance on actual individual workloads can be empirically validated. Second, the feature space is restricted to task metadata: physiological signals (heart rate variability, electro dermal activity), calendar density, and typing pattern features could substantially improve prediction accuracy but require additional data sources and user consent mechanisms beyond the scope of the present work. Third, the system currently supports single-user, single-board configurations: multi-board aggregation and team-level stress monitoring are architecturally feasible but not yet implemented.

C. Ethical Considerations

Workload stress monitoring raises legitimate privacy concerns. STRESS-AIDE is designed as an opt-in personal productivity tool: the user controls when to sync and when to run predictions, all data is stored locally in the user's own PostgreSQL instance, and no data is shared with third parties. The feedback mechanism is voluntary and never coercive. These design choices reflect the ethical principle that stress monitoring should empower individuals, not enable surveil- lance by employers.

X. Conclusion And Future Work

This paper has presented STRESS-AIDE, an AI-powered framework for proactive workload stress detection and mitigation in software project management. The system achieves 89.0% classification accuracy with XGBoost on a three-class stress prediction task using seven features engineered from Trello task metadata. A two-layer recommendation engine with ten typed triggers produces contextual, ranked, and fully explainable workload management guidance. A human-in-the- loop feedback architecture enables continuous model personalization. The complete system is deployed as a containerized full-stack application with 85 passing tests and 91% average code coverage.

The principal finding is that PM tool metadata alone constitutes a sufficient signal for practical, low-latency workload stress prediction at the individual level — without requiring physiological sensors or invasive monitoring. The dominance of overdue-count (importance 0.312) in the XGBoost model confirms the theoretical grounding in Karasek's demand- control model and provides a clear, actionable message for workload management practice: preventing task overrun is the single highest-leverage intervention for individual stress reduction.

Future work will pursue three directions. First, the syn-thetic training data will be replaced with real-world labels collected through a production deployment of the feedback loop, enabling rigorous external validation of the model's generalization performance. Second, the feature space will be extended with Google Calendar integration — adding meeting density, available focus time, and after-hours work patterns as additional signals. Third, team-level stress aggregation will be implemented to provide engineering managers with aggregate workload visibility, enabling proactive redistribution of tasks before individual stress reaches critical levels.

REFERENCES

[1] Deloitte, ”Workplace Burnout Survey,” Deloitte Insights, 2023.[Online].Available:https://www2.deloitte.com/us/en/pages/about-deloitte/articles/burnout-survey.html
[2] T. Chen and C. Guestrin, ”XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD), San Francisco, CA, USA, 2016, pp. 785–794.
[3] M. Gjoreski, M. Lus˘trek, M. Gams, and H. Gjoreski, ”Monitoring stress with a wrist device using context,” J. Biomedical Informatics, vol. 73, pp. 159–170, 2017.
[4] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza, ”Power to the people: The role of humans in interactive machine learning,” AI Magazine, vol. 35, no. 4, pp. 105–120, 2014.
[5] R. A. Karasek, ”Job demands, job decision latitude, and mental strain: Implications for job redesign,” Administrative Science Quarterly, vol. 24, no. 2, pp. 285–308, 1979.
[6] A. Sano and R. W. Picard, ”Stress recognition using wearable sensors and mobile phones,” in Proc. Humaine Association Conf. Affective Computing and Intelligent Interaction (ACII), Geneva, Switzerland, 2013, pp. 671–676.
[7] M. Gjoreski et al., ”Monitoring stress with a wrist device using context,” J. Biomedical Informatics, vol. 73, pp. 159–170, 2017.
[8] T. Chen and C. Guestrin, ”XGBoost: A scalable tree boosting system,” KDD 2016, ACM, pp. 785–794.
[9] A. Rastogi and M. Mehrotra, ”Burnout prediction from project manage- ment metadata,” J. Information and Optimization Sciences, vol. 40, no. 2, pp. 523–536, 2019.
[10] S. Hassan, C. Tantithamthavorn, C. P. Bezemer, and A. E. Hassan, ”Studying the dialogue between users and developers of free apps in the Google Play Store,” Empirical Software Engineering, 2021.
[11] G. Mark, S. Iqbal, M. Czerwinski, and P. Johns, ”Bored Mondays and focused afternoons: The rhythm of attention and online activity in the workplace,” in Proc. ACM CHI Conf. Human Factors in Computing Systems, Toronto, Canada, 2014, pp. 1911–1920.
[12] R. Burke, ”Hybrid recommender systems: Survey and experiments,” User Modeling and User-Adapted Interaction, vol. 12, no. 4, pp. 331–370, 2002.
[13] F. Cirillo, The Pomodoro Technique. FC Garage, Berlin, 2006.
[14] A. Ariga and A. Lleras, ”Brief and rare mental ’breaks’ keep you focused: Deactivation and reactivation of task goals preempt vigilance decrements,” Cognition, vol. 118, no. 3, pp. 439–443, 2011.
[15] B. Settles, ”Active learning literature survey,” University of Wiscon- sin–Madison, Computer Sciences Technical Report 1648, 2009.
[16] F. Pedregosa et al., ”Scikit-learn: Machine learning in Python,” J. Machine Learning Research (JMLR), vol. 12, pp. 2825–2830, 2011.
[17] D. Sculley et al., ”Hidden technical debt in machine learning systems,” in Proc. Advances in Neural Information Processing Systems (NeurIPS), Montreal, Canada, 2015, pp. 2503–2511.
[18] Atlassian, ”Trello REST API Reference,” 2023. [Online]. Available:https://developer.atlassian.com/cloud/trello/rest/