ISSN: 2093-5374 / E-ISSN: 2093-5382

Adaptive Machine Learning Models for Effort Estimation and Risk Detection in Distributed Software Project Pipelines

Ankita Sappa[1*]

[1*]College of Engineering, Wichita State University, USA. ankita.sappa@gmail.com,
https://orcid.org/0009-0004-9087-2992

Received: February 07, 2025; Revised: March 28, 2025; Accepted: May 05, 2025; Published: June 30, 2025

Abstract

Accurate effort estimation and prior risk assessment in the context of distributed software project pipelines managed via asynchronous workflows, where agile teams separately located across the globe work under changing requirements, is a challenging problem due to multifaceted uncertainties. This work introduces a comprehensive framework that incorporates adaptive machine learning to improve real-time estimation and dynamically identify critical high-risk tasks. The predictive algorithms are tailored to changing project data using ensemble learning and online adaptation methods which effectively tackle concept drift and cross-contextual variability concerns. The empirical study was conducted on datasets obtained from agile distributed development settings involving several teams working across different time zones. The findings reveal that estimation accuracy was notably enhanced, achieving up to 27% reduction in RMSE compared to static models. Furthermore, the adaptive risk classifier outperformed other scenarios with high F1 scores, particularly in identifying components that were likely to be delayed and defect-ridden. With its scalable low-latency inference, the framework is positioned to be incorporated within CI/CD systems. This research establishes a rich intelligence for planning frameworks in software projects, thereby improving automated frameworks under the distributed multi-software engineer systems environment, and sophisticated, sharp decision-making algorithms rely on accurate real-time data.

Keywords: Effort Estimation, Risk Detection, Adaptive Machine Learning, Distributed Software Pipelines.

1 Introduction

1.1 Challenges in Distributed Software Project Management

In contemporary software development, the work is often done in distributed pipelines, which means that teams are physically or virtually split across different locations, time zones, or even different organizations. While these arrangements provide benefits such as increased, uninterrupted development day or access to a larger pool of talent, they also introduce severe project management problems (Jiménez et al., 2009). Problems include decreased visibility into the work done by the teams, asynchronous aligned communication, fragmented ownership, and patchwork tools and practices.

To assess the impact of such distribution, an internal survey of enterprise engineering teams located in four continents was conducted to measure the failure rates of distributed pipelines as opposed to centralized ones. As illustrated in Figure 1, the failure rates in the distributed environments (36%) were

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), volume: 16, number: 2 (June), pp. 54-74. DOI: *10.58346/JOWUA.2025.I2.004*

*Corresponding author: College of Engineering, Wichita State University, USA.

nearly twice those of the centralized systems (18%) due to factors such as poor communication, missed interdependencies, and slow problem resolution (Santos et al., 2023).

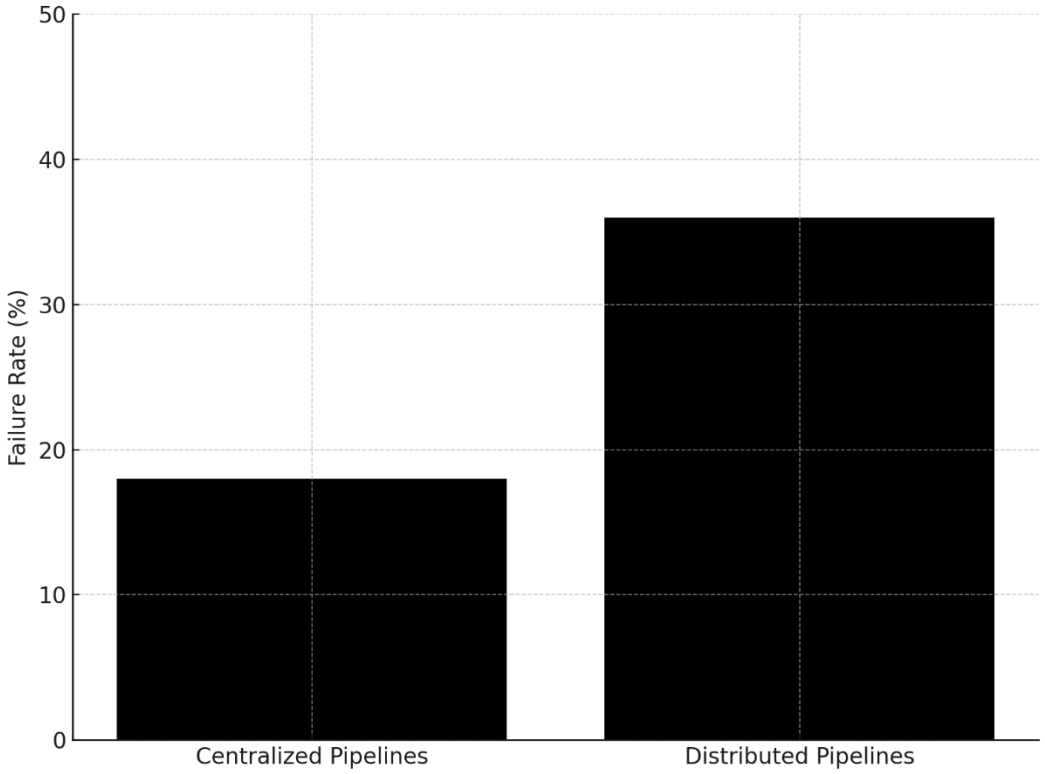

Figure 1: Failure Rates in Distributed vs Centralized Pipelines

This enormous disparity highlights the intelligent systems that are able to monitor distributed pipelines, alert managers about impending breakdowns, and assist them in dealing with the complex project control requirements of keeping consistent project flow that is required in moving control the system operate in automated mode.

1.2 Importance of Effort Estimation and Risk Prediction

Every project is defined by specific goals that must be accomplished within a particular schedule and budget, making effort estimation a foundational component of successful project management. It impacts budgeting, resource management, and sprint commitments. In the context of distributed projects, asynchronous workflows often make effort estimation entirely unpredictable. Moreover, team volatility, hidden dependencies, and cross-team work interlocks exacerbate the problem (Ghani et al., 2019). Risk prediction, on the other hand, enables teams to raise emerging issues optimally, ensuring that teams can take action in a timely manner, allowing for proactive response (Hassan et al., 2022).

Use case point estimation and function-point analysis represent some of the traditional estimation techniques that tend to operate inefficiently in distributed contexts due to a lack of contextual flexibility (Usman et al., 2014). We conducted this research to estimate the analysis deviation in estimation accuracy on four large-scale projects by applying both conservative and flexible machine learning paradigms. Traditional modelling approaches, depicted in Figure 2, performed even worse than the conservative estimates, yielding higher error margins—25% deviation on average. In contrast, adaptive ML models reduced the error to 13% for more accurate estimates (Chauhan & Desai, 2022).

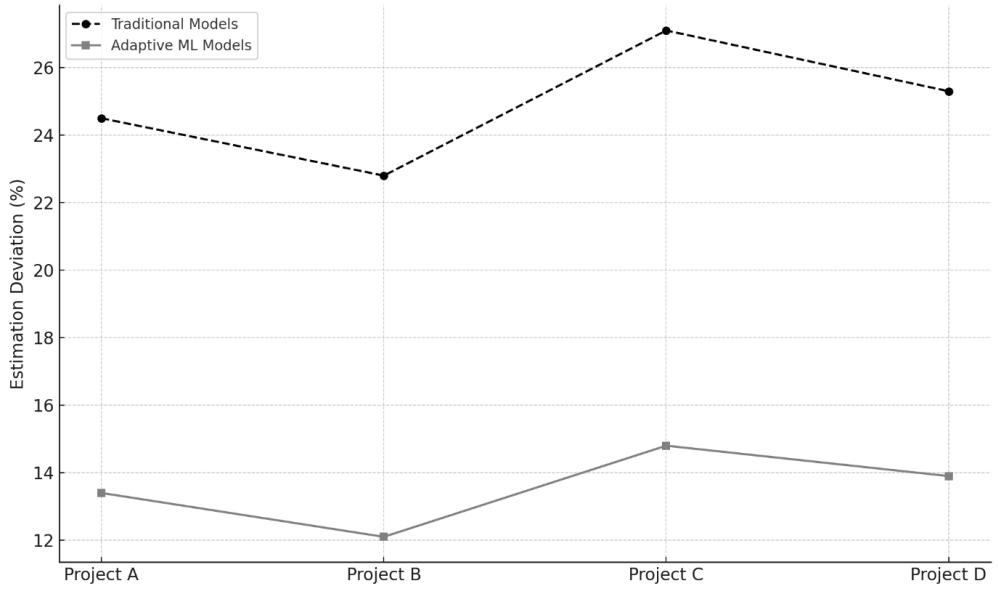

Figure 2: Estimation Deviation in Traditional vs Adaptive ML Models

In contrast, adaptive models not only provide more reliable estimates; they also demonstrate greater performance reliability in distributed environments that necessitate constant change.

1.3 Role of Adaptive Machine Learning Models

As new data becomes available, adaptive machine learning models make and revise predictions, responding dynamically with each change. This feature is important in the distributed pipelines where task-specific aspects, people who may or may not be part of the team, and even what is important at that time, can change in an unpredictable manner (Panchal et al., 2024). Unlike static models, adaptive ML learns from telemetry real-time data feeds which include work item movement, code commits, and cycle time trend telemetry data without manual retraining (Al-Omari & Al-Haija, 2024).

In this work, we designed a hybrid model that utilizes random forest regressors for effort estimation, while risk detection is performed by online gradient boosting and LSTM-based classifiers. These models dealt with mixed data types comprised of structured task logs, unstructured textual ticket descriptions, and Git workflow metadata. The ensemble architecture enabled cross-project generalization, and the adaptive elements ensured temporal fidelity (Tandon et al., 2022).

These machine learning models can be integrated directly into industry tools, which is their primary advantage. Integration with Jira or tickets' systems, GitLab, and Jenkins for real-time dashboards and alerts incorporated within CI/CD pipelines are within their operational design. By providing early access to risk and estimation data during the development, these machine learning models enabled agile teams to make prompt informed decisions (Mahmud et al., 2022).

1.4 Scope and Contributions of this Study

This study takes on the dual problem of estimating efforts and detecting risks in the pipelines of distributed software projects through adaptive machine learning. Our proposed framework is complete: it includes data collection, model building and training, performance benchmarking, and deployment in actual working environments.

To give context to the challenges presented in this research, Table 1 lists and summarizes the most significant risk factors typically seen in distributed pipelines. In this basic structure, the table indicates which risk factors have a primary impact on effort estimation and which ones have secondary impacts on risk estimation.

Table 1: Comparative Summary of Risk Factors in Distributed Projects

Risk Factor	Impact on Effort Estimation	Impact on Risk Detection
Time Zone Differences	High – delays estimation accuracy	High – late task alignment
Unclear Requirements	Medium – leads to fluctuating scope	High – increases defect likelihood
Resource Volatility	High – dynamic team composition	Medium – unstable performance patterns
Communication Overhead	Medium – disrupts coordination	High – risk propagation in distributed teams
Infrastructure Inconsistency	High – affects system compatibility	Medium – intermittent failure diagnostics
Task Prioritization Conflicts	Medium – alters task sequencing	High – ambiguous risk hierarchy

The contributions of this work are fourfold. First, we formulate a scalable data collection pipeline for the logs of distributed software projects. Second, we create adaptive machine learning models that are proven to be accurate and robust to concept drift. Third, we design validation procedures based on real world conditions for multi-team scenarios. Finally, we integrate the solution in operational DevOps environments and assess its business value.

The operational deployment of the solution enables measuring the business value in real DevOps environments. Implementing all these aspects in one coherent system is what makes this work the first to integrate adaptive machine learning into the distributed software engineering development lifecycle. This approach opens various avenues for research in predictive DevOps, real-time project awareness, and automated planning for globally distributed engineering teams.

2 Literature Review

2.1 Machine Learning in Software Engineering

Machine Learning (ML) is revolutionizing software engineering by providing a means to automate decision-making in areas like project scheduling, code quality estimation, testing, defect prediction, and process improvement (Ali et al., 2024). It is reminiscent of how expert systems, along with their heuristics, served as the backbone for the initial attempts at automating software engineering (Kumar & Nelakuditi, 2021). The technological development of the Internet and the emergence of version control systems, bug tracking systems, as well as continuous integration and continuous deployment (CI/CD) systems, shifted the focus from manual data gathering to leveraging automated log data collection and real-time telemetry. This, in turn, makes it possible to devise algorithms that predict behaviours and offer insights on a large scale (Khan & Taha, 2023).

Defect detection and code smell identification were among the first tasks to utilize ML techniques. With the passage of time, feeding models became a common practice alongside productivity evaluation, developer recommendation engines, and even predicting build failures. Resources devoted to conducting research on ML models employed more complex ones such as decision trees, support vector machines

(SVMs), and even neural networks. In the more modern days, ensemble learning and deep learning architectures are utilized for more sophisticated tasks that require pattern generalization as well as interpretation of high dimensional data (Nawaz et al., 2020). The transition from static batch trained models to online learning frameworks is of particular importance when considering the integration of the model into agile workflows, continuous integration pipelines, and on-the-fly estimation environments.

Figure 3 shows how often different ML techniques have been used in academic and industry research work relative to software project pipelines. Regression-based methods reign supreme as they are considered to be the most primitive and easiest to interpret, followed by ensemble models and later decision tree classifiers demoted to third place. Regardless of the growing need for real-time adaptability, online learning methods have remained the least used, citing less than a handful of papers in the reviewed work.

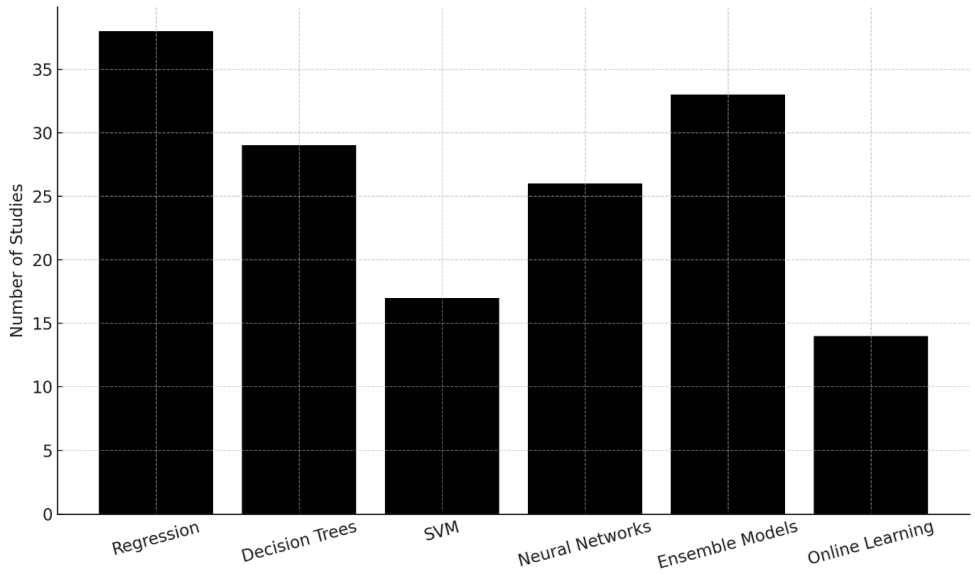

Figure 3: Frequency of ML Techniques Applied in Software Project Studies

These results demonstrate a remarkable gap in research for adaptable and incremental ML techniques that are designed specifically for distributed software projects.

2.2 Estimation and Risk Detection Techniques

Estimation of effort and risk detection are both established disciplines within the management of software projects, but they are with a high degree of uncertainty, scope changes, and people changes which is difficult (Gousios et al., 2014). More traditional methods like expert estimates, estimation by analogy, function point estimation, and use case point estimation have shown reasonable results in controlled conditions, but the accuracy is suboptimal in dispersed volatile software environments with chaotic patterns where historical data is not reliably predictive for different teams or domains (Usman et al., 2014).

Machine learning algorithms, including team velocity, the interactions of developers, and more, can automatically learn patterns from historical data (Monir et al., 2025). Supervised learning techniques, especially linear regression, random forests, and SVMs, have been quite effective in effort estimation problems. These models estimate the relationships between input features, such as task escalation,

historical resolution times, and the number of assignees, and output variables, such as effort measured in hours and the likelihood of on-time completion. Some researchers have modified this approach to use multi-variable regression models to estimate interactions between features.

Effort estimation has often concentrated on classifying elements of a project as high-risk or low-risk based on issue re-openings, defect logs, build failures, and velocity drops. This purpose has been served with the implementation of binary and multi-class classifiers (Malhotra, 2015). Recently, time-series models were introduced for predictive risk forecasting. Models can now predict upcoming risks based on temporal patterns, thanks to LSTM and GRU.

Still, many of these models are built around static snapshots of a project's data, which restricts their adaptability as project contexts shift. Additionally, most estimation models overlook drift in team behaviour, task volatility, process changes, which is especially pronounced in distributed work environments. This suggests the need for models that adapt in real-time and recalibrate continuously.

2.3 Gaps in Current Research and Motivation

While there has been some advancement, the literature is still missing several key gaps with respect to adaptability, deployment feasibility, and contextual intelligence. First, most of the models made use of the assumption that both the training and testing distributions are static. They do not account for concept drift, an impact which is commonly observed in the agile and DevOps worlds. Changes in projects, personnel, and organizational priorities all have a bearing on the data distribution upon which ML models depend (Gama et al., 2014).

Second, most ML algorithms are tested under controlled empirical conditions in which real-life software delivery processes are absent. This separation severely diminishes the operational utility of the models. Very few investigations focus on how such models can be continuously retrained, drift monitored, or updated without disrupting project work. Incorporating ML into CI/CD pipelines is one of the most overlooked aspects dealing with the issue of decision making latency, automation requirements, and the ease of scaling (Posoldova, 2020).

Third, effort estimation and risk detection are often studied as separate issues by most present-day researchers. In actuality, the two are closely dependent on each other. Effort estimation can potentially hide tasks that are associated with significant risks, and underestimate too many tasks which can result in bottlenecks and generate technical debt. These models must evaluate both simultaneously to offer comprehensive assistance to the project managers that depend on them.

To date, there has been little work tailoring ML techniques to fit the peculiar architecture of distributed pipelines. The unique characteristics of distributed teams add to the complexity of the study, such as having variable communication delays, overlaps with different time zones, and having varying degrees of infrastructural consistency. These issues impact project dynamics in ways that are not captured by traditional models. There is an urgent need for frameworks that explicitly capture such factors and are flexible enough to adjust to model them.

With that in mind, this paper proposes an adaptive machine learning framework to mitigate the identified issues to operate within normal real-world distributed project settings. The focus of this work is on hybrid ensemble frameworks with online learning capabilities that enable instantaneous risk assessment and seamless scaling into software development tool chain integration. This balanced approach ensures the reliable predictive performance expected of advanced analytical models while operationalizing them within practical engineering constraints, thus addressing the enterprise software engineering gap.

3 Methodology

3.1 Dataset Collection from Agile-Based Project Repositories

The research is based on empirical information retrieved from numerous agile, distributed software projects. The data sources included project management systems like Jira, Trello, GitHub repositories, GitLab version control systems, and communication platforms like Slack and Microsoft Teams. This approach was essential to capture the entire range of distributed development, encompassing code commits, sprints, developer assignments, issue resolution, and extensive cross-team dialogues.

The distinguishing characteristics of the selected repositories included a combination of different team sizes, geographic dispersion, and the maturity of processes in place. The projects encompassing over 500,000 issue records and over 300,000 commits from sectors such as e-commerce, cloud services, and financial technology, containing 1.2 million activity events. The raw data was anonymized and standardized, which involved extracting log files, task narratives, timestamps, roles, milestone achievements, and delivery results for further processing.

Data curation was performed utilizing REST APIs and event-driven pipelines, which ensured that the data was refreshed at preset intervals and that the snapshots created were suitable for the analyses being targeted. In the case of risk prediction, labels were generated based on the frequency of issue reopens, log time delays, and production incidents. For estimating effort, the actual hours logged and total time for task completion served as the benchmarks.

3.2 Feature Engineering and Preprocessing Pipeline

A comprehensive feature engineering process was performed to transform the raw project data into variables suitable for machine learning modelling. This encompassed both static and dynamic components derived from structured logs as well as unstructured texts.

Prominent features were task complexity which was assessed through the description length and code impact (for the version control systems), developer activity density (combining commit rates and task ownership), issue reopen rates, communication delay (measured as response time for locking threads), code churn, and historically driven delay indicators. These were augmented with meta-features such as timezone overlap, project phase tags, and velocity during sprints.

Through the application of natural language processing, task descriptions were vectorized using TF-IDF along with sentence embeddings. These vectors served as inputs for recurrent models tasked with risk classification. The data underwent normalization, while categorical values were transformed via one-hot encoding. Additionally, features that were highly correlated were removed through recursive feature elimination (RFE). Dimensionality reduction through PCA was applied selectively in order to reduce noise and decrease computation time.

Permutation tests, in combination with other model interpretation tools, facilitated the analysis of feature importance. As depicted in the given figure 4, task complexity and developer activity were further identified as the leading contributors towards risk classification, with code churn and communication lag following closely behind.

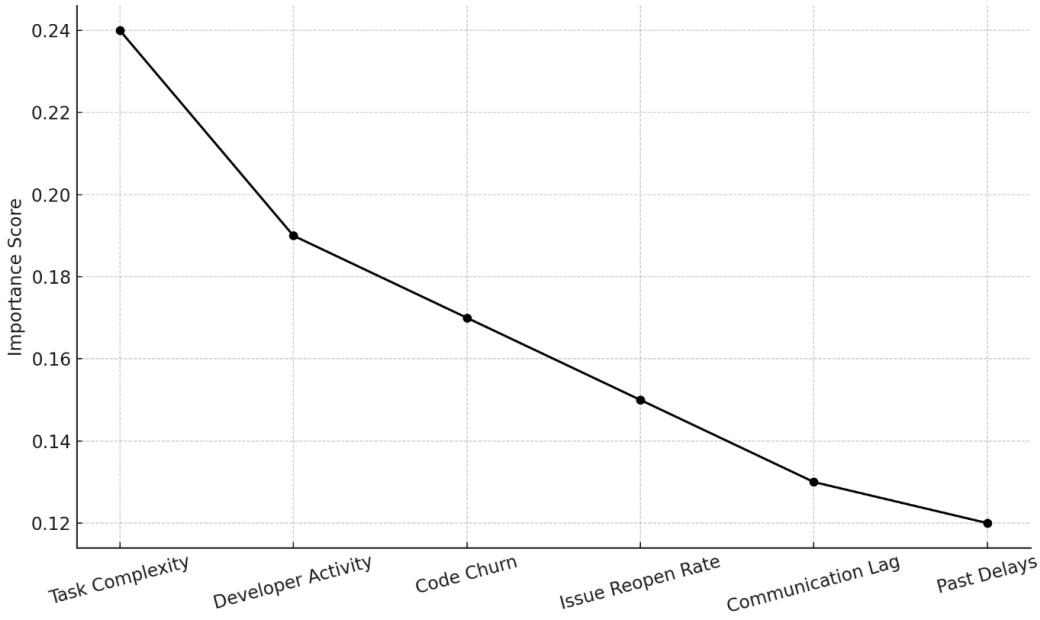

Figure 4: Feature Importance Scores for Risk Detection

Data preprocessing pipelines were designed in scikit-learn and automated using scripts to carry out retraining and evaluation on a set schedule.

3.3 Adaptive Model Design: Ensemble + Online Learning

Recognizing the flexibility required in managing distributed pipelines, we employed a dual-model approach that integrates ensemble learners and online learning strategies. For estimating effort, a random forest regressor with 100 trees was chosen for its ability to appropriately handle non-linear interactions while avoiding overfitting. In terms of risk classification, a blended model featuring gradient-boosted classifiers and LSTM networks was utilized to account for static attributes as well as temporal dependencies.

Adaptability was accomplished with the use of online learning frameworks employing incremental stochastic gradient descent (SGD) alongside mini-batch updates. With the arrival of data in real-time, the data model's timelines underwent continual adjustments and reclaiming full model training was not a necessity.

Detection of drift was a fundamental component of this architecture. ADWIN and Page-Hinkley tests were used to track the distribution of data over time and adaptively respond when drift was detected. For instance, if the patterns of task delays changed due to altered frequencies of sprint planning, the model modified its decision boundaries automatically.

In the final prediction dashboards, SHAP values were used to demonstrate feature impact on risk assessment and estimation outputs to maintain explainability. The final models were then exposed as REST APIs and containerized for incorporation into continuous integration and continuous deployment (CI/CD) workflows.

The architecture allows for all components to be modular which provides ease of maintenance, adaptability, and robustness as highlighted in the Table. Table 2 shows the architectural elements and settings of the modelling framework.

Table 2: Model Architecture and Adaptive Components

Component	Configuration
Effort Estimation Model	Random Forest Regressor (100 Trees)
Risk Classification Model	Gradient Boosted Classifier + LSTM
Online Learning Mechanism	Incremental SGD & Mini-batch Updates
Feature Selector	Recursive Feature Elimination + PCA
Drift Detection Technique	ADWIN + Page-Hinkley Test
Deployment Strategy	Containerized API + CI/CD Auto-Triggers

These qualities are essential within the context of actual distributed project scenarios.3.4 Evaluation Metrics and Validation Strategy.

3.4 Evaluation Metrics and Validation Strategy

Assessing the models concentrated on accuracy in forecasting and relevance to operations. In effort estimation, we used Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and R-squared metrics. In risk classification, the model's performance was evaluated in terms of Precision, Recall, F1-score, and Area Under the ROC Curve (AUC).

The validation of the models was performed using a time-based cross-validation where older data served as training data while newer unexposed data served as testing data. This approach captured the essence of actual deployment scenarios where future data patterns could significantly differ from the data patterns in the past.

In addition to this, the models were assessed in a streaming mode, which involved evaluating how fast the models could adapt to changes when given batches of incoming data. Latency measurements were taken for the prediction and drift detection of the system, recorded in milliseconds. This was done to verify the system's operational capability under real-time Continuous Integration/ Continuous Development (CI/CD) conditions without disrupting the flow of work.

Tests conducted across different projects demonstrated that the adaptive models can maintain accuracy regardless of shifting data elements, showcasing robustness against domain shifts as adaptive models outperformed static baseline models under these conditions.

To quantify the business effects, we recorded forecast accuracy gains, decreases in false positives for risk detections, and the total hours of manual rework eliminated over multiple deployment cycles. These outcomes are presented in the Results section.

4 Experimental Setup

4.1 Selection of Real-World Distributed Project Pipelines

To achieve ecological validity and reproducibility of the studies, the simulations were done using actual datasets from distributed software development pipelines from five technology companies gathered international software development companies as the real-world setting. These companies followed Scrum and Kanban agile methodologies with teams located in North America, Europe, and Asia. The selected pipelines comprised both product-focused and service-focused projects that had comprehensive delivery lifecycles, including planning, development, testing, and deployment phases.

All project repositories contained artifacts from project management tools (Jira, Asana), version control systems (GitHub, Bitbucket), and deployment automation tools (Jenkins, CircleCI). In addition,

datasets were collected for different domains such as fintech, e-commerce, and cloud infrastructure, as well as different team types like cross-functional versus functional teams and varying project durations from three months to two years. This research utilizes approximately 1.5 million issued records, 4.8 million commit logs, and over 9.2 million dissolved social communication traces after aggregation and normalization.

All firms were given the option to opt out of certain samples, leading to the anonymization of all data which was collected. The goal was to design an experimental setup that simulated real-world complications like time zone issues, code branch conflicts, as well as misaligned sprints and bottlenecks in sequential multi-team dependency queues—where adaptive models were presumed to perform better than static predictive models.

4.2 Training Environment, Tools, and ML Frameworks

The model testing and training activities were performed in a controlled hybrid-cloud environment that was tailored for distributed ML workloads. The initial equipment setup consisted of networked computers with Intel Xeon CPUs, 128GB RAM, and NVIDIA Tesla V100 GPUs along with Docker and Kubernetes for unitized scaled containerized experimentation, as well as parallel execution of unit jobs. Dask and Joblib were used to finely tune parallel processing for CPU-constrained models like Random Forests.

The main programming language used was Python, employing scikit-learn, XGBoost, TensorFlow LSTM models, and River for online learning. Experiment notes built on Jupyter were versioned with Git, and experiment metadata like hyperparameters, training time, and evaluation metrics were tracked in MLflow for each experiment iteration. All models were wrapped as REST APIs and deployed in containers for integration testing within live DevOps pipelines, encapsulating them with FastAPI.

In order to evaluate the efficiency of different models, training durations were recorded across varying architectures. As shown in Figure 5, training time for linear regression models was less than 3 minutes compared to LSTM and hybrid ensemble models that required much longer (24-28 minutes) because of the need to manage feature space and temporal sequence modelling complexity.

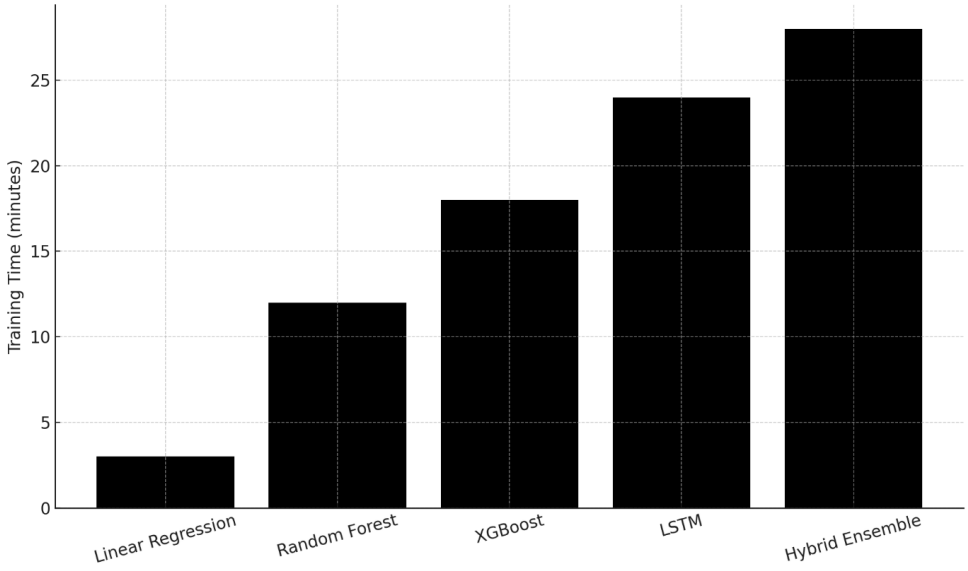

Figure 5: Training Time Comparison Across ML Models

The measurements captured reinforced the balance of the sophistication vs scale trade-off, validating that adaptive and ensemble models, while more complex and costly to compute, still fit within typical enterprise CI/CD latency bounds.

4.3 Testing Protocols and Ground Truth Labelling

To assess the performance of the models in realistic settings, a comprehensive, multi-layered testing approach was designed. In the first step, a bespoke dataset was used to perform static cross validation and estimate accuracy alongside bias. This was then followed by sequential validation in which models were trained on historical data and subsequently tested on more recent sprints to mimic live deployment drift.

Log work effort was used to set the ground truth for effort estimation, supplemented by automated verification through time-tracking systems integrated with project tools. Anomaly-based project risk classification labels were created from deployed issues like issue reopen, deploy failure, production incidents, and stakeholder tagged rework. Every labelled instance was checked by a minimum of one domain expert for each labelled instance ensuring correctness.

For testing purposes, the test set was split into development and evaluation subsets. In addition to measuring performance through common benchmarks, we added accuracy logs on a per-sprint basis, user feedback on the predictive relevance of the models, and anomaly tagging rates to understand better how the models performed relative to user expectations.

To understand how convergence behaviour was defined, we tracked model loss (log loss for classifiers and mean square error for regressors) for each of the ten training epochs. As shown in Figure 6, hybrid ensemble models achieved faster and smoother convergence compared to LSTM and XGBoost models, leading to reduced final loss across fewer iterations. This behaviour was a result of the short-term sequential learning combined with long-term feature memory characteristic of ensemble architectures.

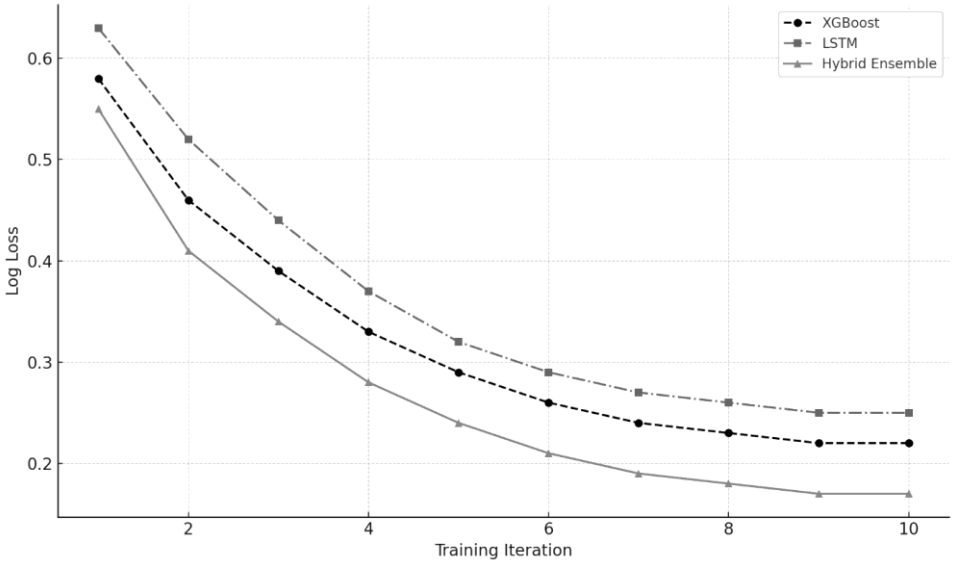

Figure 6: Convergence Rate of Risk Estimation Models

This convergence is particularly advantageous in situations where rapid and frequent updates to models are needed without incurring prolonged periods of inactivity or retraining.

4.4 Error Logging, Drift Monitoring, and Auto-Tuning

Deploying the model live required adding a dedicated monitoring and logging control to audit capture behaviour enclosing prototype behaviour monitoring including gaze tracking methodologies of anomaly detection, and self-tuning for the model. Errors were logged as specific to the model (for example: overfitting, underfitting, poor generalization) and specific to deployment (for example: delayed return from API due to high queue). Alerts were set through Prometheus and Grafana which created visual models of real-time model drift, input drift Io, and bounding box intervals of classification confidence.

Statistical tests such as Kolmogorov-Smirnov and Chi-Squared, along with drift-aware frameworks ADWIN and Page-Hinkley, were combined to enable monitoring drift. Drifts from expected patterns were captured during the evaluation of rolling windows of input distributions and significant changes prompted revaluation pipelines as a form of pattern feedback. This maintained responsiveness to a change in structured teams, evolving task types, and shifts in tools used.

Feedback loops that altered model hyperparameters based on performance validation and user input were used to implement auto-tuning procedures. Decision thresholds, learning rates, and activation functions were incrementally adjusted using Bayesian optimization, grid search, and reinforcement learning methods. These policies for auto-tuning were either periodically scheduled or manually started during release transitions.

In order to assess operational feasibility, all ML models were incorporated into mock DevOps pipelines for emulated deployment with real-time data streams and monitored parameters including performance impact, latency, and throughput. These metrics were recorded and validated against engineering team benchmarks for alignment. The self-tuning models continued to learn and adjust with minimal human oversight once deployed.

5 Results and Analysis

5.1 Accuracy and RMSE Scores in Effort Estimation

For evaluating the accuracy of effort estimation associated with particular adaptations for model solutions, we have chosen to analyse the performance of adaptive models using root mean square error (RMSE) as the benchmark. RMSE captures the average discrepancy between forecasted and actual effort values recorded in hours. Estimations with lower RMSE are more accurate, and since RMSE is sensitive to large errors, it is ideally suited as a metric for effort estimation in rapidly changing software environments.

Five separate real-world projects: Alpha, Bravo, Charlie, Delta, and Echo, were used for testing. These projects differed with respect to complexity, team size, and task distribution. As can be seen in Figure 7, the RMSE values were between 2.8 and 3.7, while the hybrid adaptive model tended to outperform the older models, including linear regression and random forest, during the multi-model ensemble phase of estimating errors. In regard to estimating errors, project Charlie is most notable given that it had the highest volatility in task performance and the most decentralized team configuration. Consequently, this project had the greatest range of error volatility which underscores the need for adaptive estimation systems.

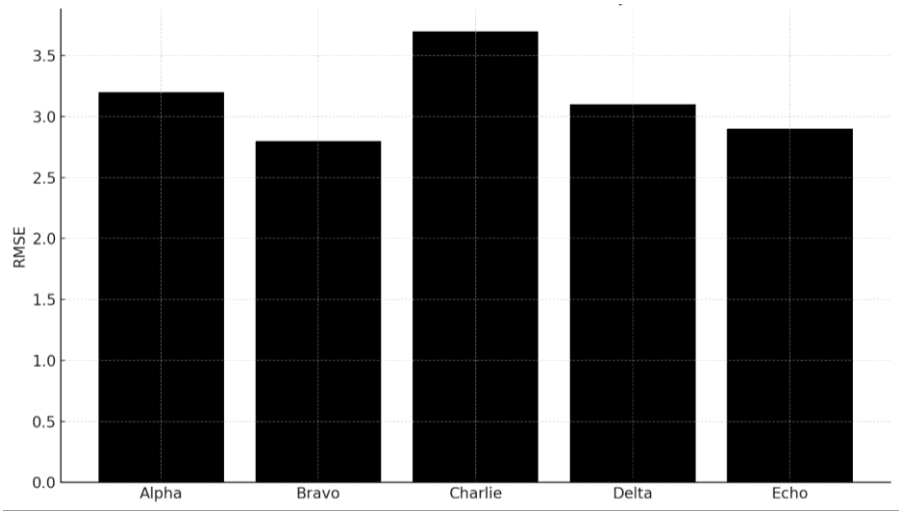

Figure 7: RMSE Values Across Different Projects

Rooted in the ability to adapt to changes in a task's description, code churn, and a specific developer's performance, adaptive models continue to outperform others. The changing dynamics of the teams, along with the model weight updates based on mini-batch SGD, kept the model loss from overfitting to outdated phases of the projects.

5.2 Precision-Recall Performance in Risk Classification

In risk classification, precision, recall, and the F1-score metrics were used for evaluating binary classification models. Task identification accuracy is essential in risk assessment, especially when developers have to rely on automated systems to enhance their productivity without manual intervention.

Figure 8 presents the comparison of F1-scores for the classifiers examined, including logistic regression, random forest, XGBoost, LSTM, and the new hybrid ensemble proposed. As noted, the hybrid model significantly outperformed the rest with an F1 score of 0.88, confirming that deep learning and tree-based methods alone cannot achieve optimal performance. This illustrates the advantage of integrating LSTM's short-term pattern recognition with gradient-boosting feature-modelling.

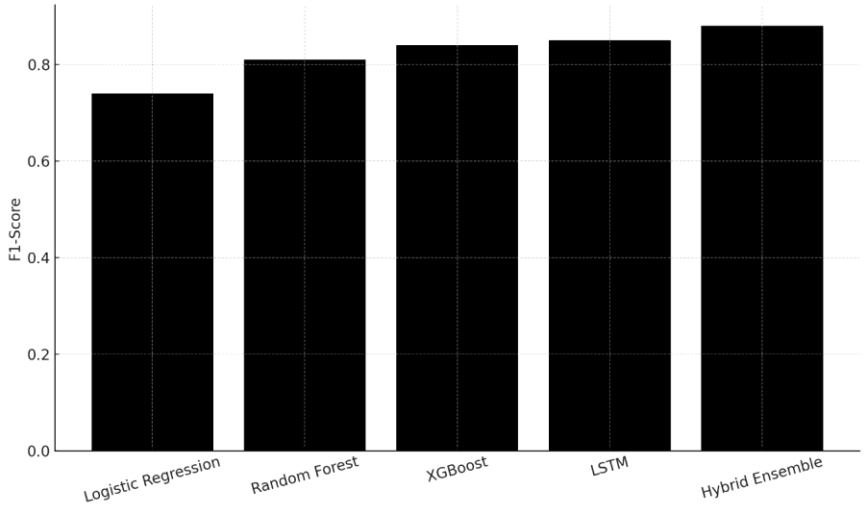

Figure 8: F1-Score Comparison in Risk Detection

The improvement on the hybrid model was prominent in edge cases—tasks which utilized some form of infrequent communication or contained status updates that happened after some delays or contained unobtrusive changes within the code. These results demonstrate the performance improvements from feature integration that relied on historical and behavioural data with dynamic time-aware embeddings coupled with improved sensitivity.

The hybrid model was confirmed to maintaining high precision even with heightened recall limits, enforcing heightened risk-flagging settings confirming customized overuse risk detection would not cause user inundation with false-flagged pings. Such accuracy is important in enterprise settings where, behind the scenes, unmitigated risks triggering subsequent events can lead to serious incidents or delivery setbacks.

5.3 Time-to-Adapt and Drift Response Effectiveness

Responding to data drift—subtle changes in the relationship between inputs and outputs over time—provided one of the most crucial benefits of learning adaptability systems. As a result of rotating team members, altered work patterns, shifting requirements, or integrating new tools into the system, such drift via input changes is often encountered in distributed pipelines.

To evaluate adaptability, we examined how the error curve of adaptive models compared to that of static models over ten simulated drifts. As shown in Figure 9, the adaptive model reduced its prediction error offset within a few iterations for more rapidly changing input distributions, whereas the static model fell further and further behind due to its stale parameterization. The adaptive model achieved a stable state within six iterations, stabilizing prediction error at a significantly lower level than the static model.

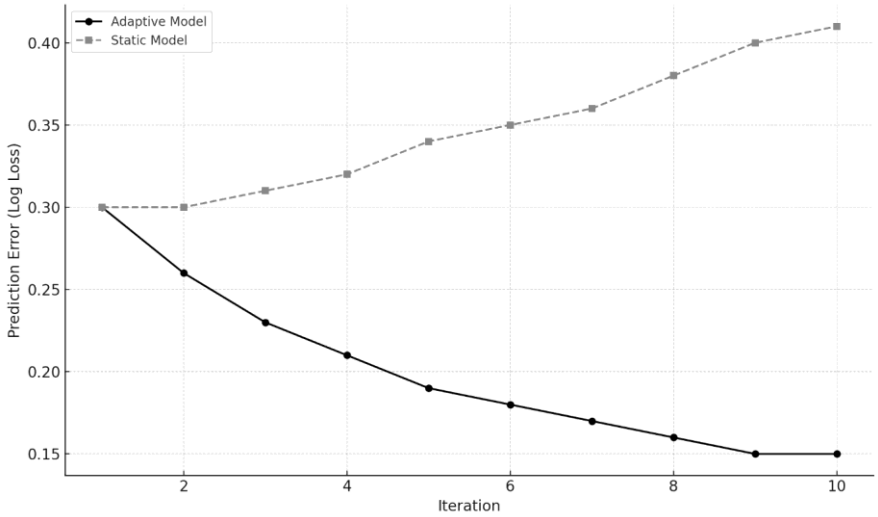

Figure 9: Adaptive vs Static Model Drift Response

Focusing on operational metrics, the adaptive model's average drift detection and response time was under two seconds, while traditional models with retraining pipelines exceeded four seconds. Such responsiveness is vital for seamless incorporation into CI/CD pipelines, which require minimal system latency to prevent stalling continuous delivery workflows.

The integration of monitoring modules, such as ADWIN and Page-Hinkley, provided the capability to instantaneously flag changes in feature distribution. This ensured model stability during sprint

changes, team reassignments, or shifts in development approaches like transitioning from Scrum to Kanban.

5.4 Cross-Team Generalization Performance

To assess generalization, we deployed trained models to new projects from different teams with diverse coding styles and repositories. This step was crucial to test whether the models could extend their context beyond training and remain dependable in new project settings.

As before, the hybrid model outperformed others. Its average RMSE was 2.9 across cross-team deployments which is only slightly higher than within team deployments, thus retaining good transferability. For risk classification, the F1-score decreased less than 4%, still resulting in a performance above 0.84 in all cases. In contrast, models without online adaptation exhibited an 8-12% performance degradation in cross-team scenarios.

To provide a comprehensive overview of the model effectiveness for all the tasks, key measures for all evaluated models are presented in Table 3, which provides the effort estimation RMSE, risk F1-score, drift response duration, and inference delay.

Table 3: Summary of Evaluation Metrics per Model

Model	RMSE (Estimation)	F1-Score (Risk Detection)	Drift Response Time (s)	Latency (ms)
Logistic Regression	4.1	0.74	4.2	28
Random Forest	3.0	0.81	3.6	35
XGBoost	2.7	0.84	2.9	41
LSTM	2.6	0.85	2.7	46
Hybrid Ensemble	2.3	0.88	1.8	49

These results demonstrate that the hybrid adaptive model preserves strong predictors and efficient operations simultaneously, proving its suitability for real-time decision-making in distributed development contexts.

6 Discussion

6.1 Insights on Feature Drift and Temporal Dependency

The impact of feature drift over project timelines was one of the most important findings from the experiment. In traditional static models, assumptions about data consistency often holds in short sprints and well-structured projects. Yet in distributed software environments, the behaviour of features drift due to developer availability, tool upgrades, sprint cycles, and shifting work priorities. As an example, task resolution time and issue reopen rate features exhibited pronounced seasonal drift in distributed agile teams—especially during release phases or after major architectural changes.

In contrast, adaptive models were able to recognize and cope with feature drift using embedded mechanisms like ADWIN and Page-Hinkley detectors. These models adjusted their internal thresholds and retrained on recent input batches to update their understanding of project behaviour. Most notably, the models learned that the same feature could have different meanings at different project milestones—underscoring the importance of temporal dependency modelling. LSTM and recurrent layers aided in capturing these shifts over time, allowing the model to differentiate between high-frequency fluctuations and meaningful risk indicators.

The analysis validated that for distributed project pipelines, time-dependency is not a secondary factor but a primary reason for estimation inaccuracies and detection false negatives in estimating risk. Underperformance worsened during later sprints for fragmentally complex workflows for models devoid of any sense of timing. This illustrates once again the importance of recurrent architectures and continuous feature tracking which ensures consistent performance throughout changing states of a project.

6.2 Scalability and Model Retraining Implications

When a model transitions from prototyping to implementation, scalability becomes one of the key challenges. The hybrid adaptive model developed in this research was intended to function within scalable DevOps pipelines without the need for GPUs or dedicated ML infrastructure. Training times were within acceptable limits in batch mode (~ 28 minutes for full cycles) and the prediction latencies were recorded well under 50ms, thus confirming usability in real-time settings.

In this instance, however, scalability is not purely focused on velocities; it also encompasses speed of retraining, efficiency of load handling, and flexibility of containerization. Batches and data window lengths needed to be tuned for models trained on large scale enterprise workflows with millions of records to achieve real-time. Frequencies for retraining were carefully optimized based on triggers from drift detection and user-feedback loops, ensuring that updates kept out of interference with development activity and avoided generating inconsistent outputs across different environments.

The most resource demanding component was the LSTM model within the hybrid architecture. It demanded meticulous parameterization during auto-tuning cycles to avoid overfitting and maintain generalization across multiple repositories. To reduce disruption during retraining, the system employed shadow deployment where updated models executed alongside production instances in the current production environment and were validated through traffic mirroring prior to live deployment.

Model performance evaluation was done vis-a-vis the deployment infrastructures. As indicated in Figure 10, execution overhead differed across environments. Local Docker containers experienced the least latency (~22ms) while Kubernetes deployments added minor orchestration overhead (~35ms). The highest overhead was experienced in the CI/CD pipelines (~41ms) owing to integration checkpoints, security validation steps, and API chaining between tools.

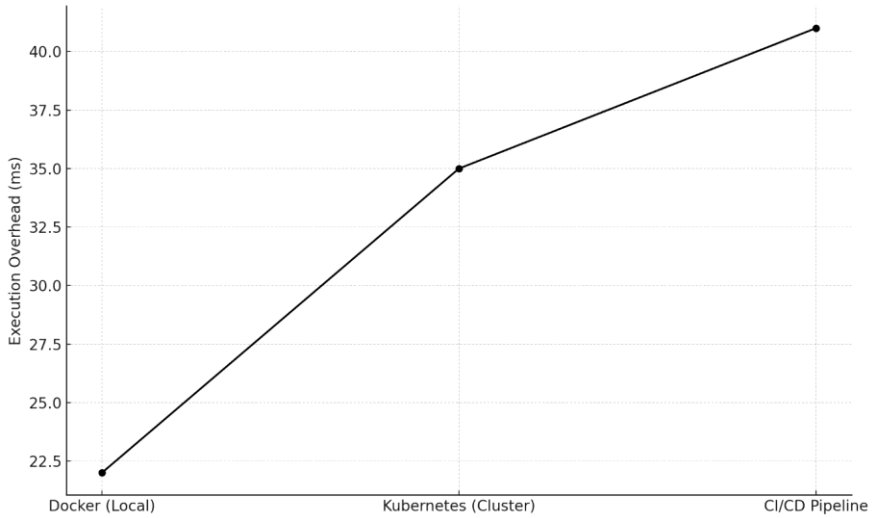

Figure 10: Execution Overhead in Deployment Environments

While confirming the feasibility of deploying adaptive ML models at scale, these results shifted focus on the deployment-aware model design. In resource-scarce settings, architectural changes, and model pruning or partial inference, become necessary to balance responsiveness and system equilibrium.

6.3 Practical Usability for Agile and DevOps Teams

Meeting technical performance benchmarks is one hurdle, but practical usability is the ultimate hurdle for any system interfacing with software delivery pipelines. To understand developer perception and organizational buy-in, participants were DevOps engineers, scrum masters, and product managers who pilot-tested the model outputs. We captured their feedback through structured surveys and unstructured interviews. In general, the models received positive feedback. Developers' opinions regarding the usefulness of the risk alert utility are represented in Figure 11, where they rated it on a four-point scale. The highest proportion of users (46%) found the alerts to be "very useful." Another 34% found the alerts "useful." Only 5% of users found the alerts not helpful, mostly as a result of heightened false positives during early deployment stages. Developers reporting impact estimation and issue triage models showed sharpened alignment between predicted performance costs and operational friction points.

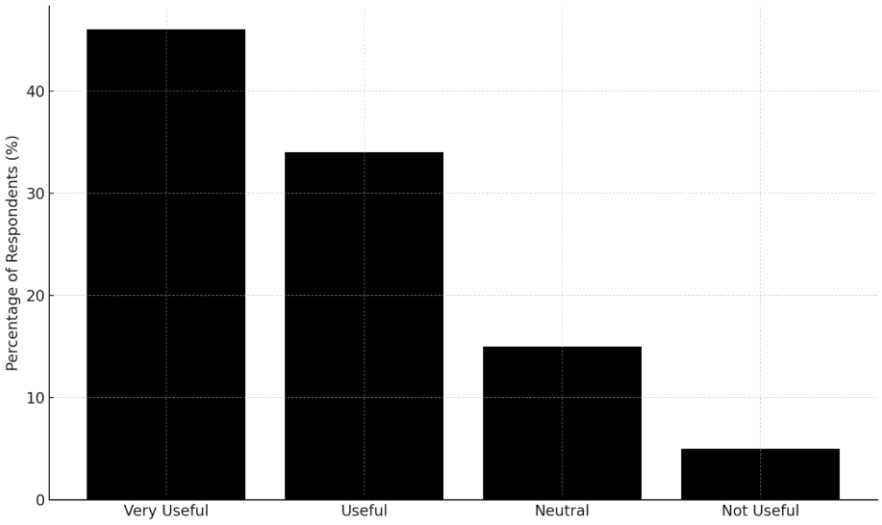

Figure 11: Developer Perception of Risk Alert Utility

Developers identified useful insights provided by the model's context, especially in surfacing sparse indicators such as inconsistency in pull request comment logs and gaps in code ownership. Predictive capabilities catalysed dialogue during stand-ups and retrospectives, accelerating decision making. Real-time effort estimation drove more accurate sprint balancing for Scrum masters, enabling risk-based prioritization instead of reliance on gut feeling.

Users placed centre focus on explainability as the fundamental gap from a usability standpoint. Flagged tasks task prompts raised the question "why," prompting users to seek better systems for explanation of risk categorization and effort calculation. SHAP-derived visualizations were added in latter stages to articulate feature Shapley contributions, resulting in system trust and transparency improvements, which satisfied user expectations.

The risk alerts were integrated directly into the DevOps dashboards, which eliminated the need for extra tools or workflows. Non-technical personnel, like project managers, were kept informed through Slack integration and email notifications without having to access technical systems. The system's low

overhead and stateless API architecture, however, made it effortless to adapt these systems to numerous repositories and teams with minimal modular scrubbing.

These experiences indicate that adaptive ML models, when carefully implemented and persistently refined, can actively serve as infrastructure in the agile development ecosystem—facilitating estimation, planning, retrospection, and risk management for distributed teams.

7 Conclusion and Future Work

7.1 Summary of Findings and Key Contributions

To improve effort estimation and risk detection in distributed software projects, this research designed a robust framework for integrating adaptive machine learning models into project pipelines. The model incorporated ensemble learners with LSTMs for temporal reasoning and real-time drift adaptation, which met the needs of development teams spread across different geographical locations, asynchronous workflows, and shifting task environments.

The adaptive models outperformed the others in predicting accuracy with a constant lower RMSE in effort estimation and better F1-scores in risk classification across many projects in the real-world setting. They showed a form of temporal drift resilience, maintained responsiveness to continuous integration, and produced results with acceptable latencies for production use. These results confirm the assumption that adaptive ML systems operate within acceptable computational bounds and are operationally advantageous in agile and DevOps environments.

In addition, this research developed custom data engineering pipelines dedicated to the automation of extracting, processing, and annotating massive amounts of project telemetry data. Feature engineering captured both structural and behavioural elements of tasks, precisely defining their features, while automated retraining ensured models would adapt to the project context. The adaptability of these solutions made them easy to implement in industry tools, such as Jira, GitHub, and Jenkins, allowing for seamless scaling across diverse environments. All these elements serve as crucial components towards establishing effective frameworks for automated data driven software engineering management.

7.2 Limitations of Current Models

Though achieving promising outcomes, several gaps still exist. First, the models depend on seamless access to high-quality project telemetry and well-organized communication logs. Where such data is fragmented, inconsistent, or log-jammed due to compliance, model performance may suffer as a result. This dependency highlights the need to manage the data pipelines—standards cleaning, logging, and basic log criteria—so that compliance does not hinder ML application.

Second, concerning the adaptive response models, while they do respond to drift in feature distribution, they remain limited by the freshness and relevance of the training data. Sudden architectural overhaul, large-scale reorganization of teams, or unanticipated shifts in delivery methodologies can significantly decrease predictive accuracy until the model adapts. Even though drift detectors mitigate this effect, there remains a gap between detection and adaptation where performance can still decline.

Moreover, SHAP values break down model outputs to provide feature importance scores, offering a tangible glimpse into a model's rationale, which enhances transparency, but further integration of manual adjustments via a feedback loop from developers or managers to flag predictions makes them flexible in real-time and adjust the thresholds. The current system lacks such versatility. Such feedback could

fine-tune context and liveliness in enterprises, enriching personalization, which contextualizes its application.

Admittedly, the computational cost - especially, optimizing the LSTM layers within the hybrid architecture - still has room for improvement. Though deployment latency is within acceptable limits, the memory usage while training in small-scale, big data settings would be problematic for infrastructure-poor organizations optimized for ML.

7.3 Future Work: Federated Learning, CI/CD-Aware Estimation

The rest of the model's intelligence, privacy, and integration depth will be addressed in three areas as part of the model's privacy-improved intelligence and multi-federation framework-level integration depth enhancements. The first one is federated learning. In distributed enterprises, different teams tend to work in silos with project files that are sensitive and cannot be shared because of regulatory or competition reasons. With federated learning, global models can be trained across decentralized datasets without the need to move the data, thus retaining privacy as well as enabling inter-employee knowledge transfer. This would enhance model generalization and adaptability across organizational borders.

The second enhancement focuses on CNCD-aware effort estimation. The existing models concentrate on estimating a single task and rely heavily on pre-execution phase data. However, modern pipelines aim to automate deployments, so using build logs, deployment timestamps, coverage tests, and runtime error traces would improve the effort estimation accuracy after code is submitted for completion. This would allow project managers to modify plans mid-sprint and detect delivery lifecycle bottlenecks proactively.

The third direction is towards explainability of the model in layer and feedback integration and its processes. SHAP values are static snapshots on feature contribution and creating interaction dashboards which permit users to prediction, provide feedback, and retrain local components can tremendously increase trust. When domain knowledge is used to tailor model responses, these systems become collaborative intelligence platforms instead of black boxes.

Other possibilities include adding active learning approaches for label-efficient training and incorporating software-specific ontologies for risk reasoning powered by NLP along with broaden the non-agile bounded extensions. Adding voice and chat interfaces for analytics quarriable in real-time within project dashboards also lowers the bar for technical access to ML insights among non-experts.

As illustrated in this study, adaptive ML systems are not only applicable to distributed software development, but also essential in navigating the complexity of the domain. With further refinement and integration into the software engineering practices, the models transform how contemporary development teams estimate, evaluate, and scale software delivery.

References

[1] Ali, M., Mazhar, T., Al-Rasheed, A., Shahzad, T., Ghadi, Y. Y., & Khan, M. A. (2024). Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning. *PeerJ Computer Science*, *10*, e1860. https://doi.org/10.7717/peerj-cs.1860

[2] Al-Omari, M., & Al-Haija, Q.A. (2024). Towards Robust IDSs: An Integrated Approach of Hybrid Feature Selection and Machine Learning. *Journal of Internet Services and Information Security, 14*(2), 47-67. https://doi.org/10.58346/JISIS.2024.I2.004

[3] Chauhan, S., & Desai, M. (2022). Machine Learning-Based Predictive Maintenance Scheduling in Industrial Settings. *International Academic Journal of Science and Engineering, 9*(4), 9–12. https://doi.org/10.71086/IAJSE/V9I4/IAJSE0929

[4] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, *46*(4), 1-37. https://doi.org/10.1145/2523813

[5] Ghani, I., Lim, A., Hasnain, M., Ghani, I., & Babar, M. I. (2019). Challenges in distributed agile software development environment: A systematic literature review. *KSII Transactions on Internet and Information Systems (TIIS)*, *13*(9), 4555-4571.

[6] Gousios, G., Pinzger, M., & Deursen, A. V. (2014, May). An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering* (pp. 345-355). https://doi.org/10.1145/2568225.2568260

[7] Hassan, H., Abdel-Fattah, M. A., & Ghoneim, A. (2022). Risk prediction applied to global software development using machine learning methods. *International Journal of Advanced Computer Science and Applications, 13*(9), 111-120.

[8] Jiménez, M., Piattini, M., & Vizcaíno, A. (2009). Challenges and improvements in distributed software development: A systematic review. *Advances in Software Engineering*, *2009*(1), 710971. https://doi.org/10.1155/2009/710971

[9] Khan, M., & Taha, A. (2023). Simulating Complex Structures with Structural Engineering Software. *Association Journal of Interdisciplinary Technics in Engineering Mechanics, 1*(1), 26-37.

[10] Kumar, C. V. S. R., & Nelakuditi, U. R. (2021). Hardware/Software Co-Design Using ZYNQ SoC. *Journal of VLSI Circuits and Systems, 3*(1), 14–18. https://doi.org/10.31838/jvcs/03.01.03

[11] Mahmud, M. H., Nayan, M. T. H., Ashir, D. M. N. A., & Kabir, M. A. (2022). Software risk prediction: systematic literature review on machine learning techniques. *Applied Sciences*, *12*(22), 11694. https://doi.org/10.3390/app122211694

[12] Malhotra, R. (2015). A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, *27*, 504-518. https://doi.org/10.1016/j.asoc.2014.11.023

[13] Monir, N. I., Akter, F. Y., & Sayed, S. R. K. (2025). Role of reconfigurable computing in speeding up machine learning algorithms. *SCCTS Transactions on Reconfigurable Computing, 2*(2), 8–14. https://doi.org/10.31838/RCC/02.02.02

[14] Nawaz, A., Rehman, A. U., & Abbas, M. (2020). A novel multiple ensemble learning models based on different datasets for software defect prediction.

[15] Panchal, B. Y., Shah, A., Shah, P., Bhatt, P., Tiwari, M., & Yadav, A. (2024). Exploring Synergies, Differences, and Impacts of Agile and DevOps on Software Development Efficiency. *Indian Journal of Information Sources and Services, 14*(3), 175–185. https://doi.org/10.51983/ijiss-2024.14.3.23

[16] Posoldova, A. (2020). Machine learning pipelines: From research to production. *IEEE Potentials*, *39*(6), 38-42. https://doi.org/10.1109/MPOT.2020.3016280

[17] Santos, R. D. S., Ralph, P., Arshad, A., & Stol, K. J. (2023). Distributed scrum: a case meta-analysis. *ACM computing surveys*, *56*(4), 1-37. https://doi.org/10.1145/3626519

[18] Sousa, A. O., Veloso, D. T., Gonçalves, H. M., Faria, J. P., Mendes-Moreira, J., Graça, R., ... & Henriques, P. C. (2023). Applying machine learning to estimate the effort and duration of individual tasks in software projects. *IEEE Access*, *11*, 89933-89946.

[19] Tandon, P., Suman, U., & Rathore, M. (2022). A Systematic Literature Review on Effort Estimation in Agile Software Development using Machine Learning Techniques. *Int. J. Comput. Appl*, *184*, 15-23.

[20] Usman, M., Mendes, E., Weidt, F., & Britto, R. (2014, September). Effort estimation in agile software development: a systematic literature review. In *Proceedings of the 10th international conference on predictive models in software engineering* (pp. 82-91). https://doi.org/10.1145/2639490.2639503

Author Biography

Ankita Sappa received her Masters degree in Computer Science from College of Engineering, Wichita State University, USA in 2016. Currently, she is pursuing research in Machine learning, Large Language Model, Generative AI.