Learning Context-Aware Representation for Short-Term Software Task Completion Prediction

by Dong Wu

MS Computer Science, 2021, Georgia Institute of Technology

A Dissertation submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

August 31, 2026

Dissertation directed by

Michael Grenn
Professorial Lecturer of Engineering Management and Systems Engineering

The School of Engineering and Applied Science of The George Washington University certifies that Dong Wu has passed the Final Examination for the degree of Doctor of Philosophy as of August 31, 2026. This is the final and approved form of the dissertation.

Learning Context-Aware Representation for Short-Term Software Task Completion Prediction
Dong Wu

Dissertation Research Committee:

Michael Grenn, Professorial Lecturer of Engineering Management and Systems Engineering, Dissertation Director

Shahram Sarkani, Professor of Engineering Management and Systems Engineering, Committee Chair

John Fossaceca, Adjunct Professor of Engineering and Applied Science, Committee Member

Thomas A. Mazzuchi, Professor of Engineering Management and Systems Engineering & of Decision Sciences, Committee Member

Joseph Blackford, Adjunct Professor of Engineering and Applied Science, Committee Member

Amir Etemadi, Adjunct Professor of Engineering and Applied Science, Committee Member

©Copyright 2026 by Dong Wu
All rights reserved

Abstract of Dissertation

Learning Context-Aware Representation for Short-Term Software Task Completion Prediction

Accurately predicting whether a software development task is likely to be completed within a short horizon, here set to 7 days, is critical to agile project management. Classical estimation approaches and traditional machine-learning models often have difficulty representing the heterogeneous and unevenly timed evidence found in software tasks, including evolving text, structured metadata, and irregular update intervals. To address this problem, this dissertation evaluates a context-aware predictive model that combines textual, categorical, and numerical variables. While typical Transformers rely on static positional encoding, the proposed model uses feature-specific time-aware attention layers to capture the recency of irregular task updates. This mechanism is combined with multi-channel embeddings and hierarchical multi-head attention modules to model dependencies within and across feature modalities. Under the reported evaluation settings on 37 Apache Jira projects, and relative to the evaluated baselines, the proposed model achieves the strongest average performance in F1-score, precision, recall, accuracy, ROC-AUC, and PR-AUC for the 7-day prediction task. The results are supported by held-out project evaluation, project-level cross-validation, training-partition statistical comparison, chronological evaluation, ablation analysis, duration-stratified analysis of short-cycle and long-running tasks, and secondary validation on GitHub issues. The duration analysis keeps the 7-day prediction target fixed and shows that the proposed architecture reaches its strongest duration-specific performance on tasks completed between 3 and 7 days. Tasks completed within 3 days are harder because the issue tracker often contains too little evolving evidence before closure, while tasks extending beyond 21 days remain challenging because their histories are longer and noisier. Taken together, these analyses show that the time-aware attention mechanism

and hierarchical multichannel fusion design make a substantive contribution to predictive performance under the reported settings, while also clarifying that the model is best aligned with task lifecycles near the 7-day decision window. The GitHub validation shows that the model remains competitive beyond the Apache Jira setting, while also indicating that domain adaptation is needed before use in a substantially different ecosystem. Comparative efficiency analysis further distinguishes the full SBERT-including pipeline from the task-specific classifier with precomputed embeddings, showing that the latter is smaller and faster than the fine-tuned RoBERTa baseline under the reported setup. The model predicts whether a task will be completed within 7 days; for operational use, the same output can also be interpreted as a complementary delay-risk score. Overall, the results support the framework as a reliable research prototype for agile decision support, while also indicating that deployment in a new organizational environment would require project-specific validation, threshold calibration, and domain adaptation.

Table of Contents

Abstract of Dissertation **iv**

1 Introduction **1**

1.1 Background and Motivation 1

1.2 An Interpretive Perspective on Data-Driven Estimation 1

1.3 Problem Statement 2

1.4 Research Objectives and Questions 3

1.5 Contributions 4

1.6 Organization of the Dissertation 4

2 Literature Review **6**

2.1 Practical Motivation for Software Effort Estimation 6

2.1.1 Schedule Overruns and Planning Risk 7

2.1.2 Coordination Overhead and Limits of Late Intervention 7

2.2 The Chronological Evolution of Software Task Estimation 8

2.2.1 Era 1 (1980s - 2000s): Expert Judgment and Parametric Models . . 8

2.2.2 Structured Judgment Support and Estimation Bias 8

2.2.3 Era 2 (2010s): Statistical Machine Learning 9

2.2.4 Era 3 (2018 - Present): Deep Learning and Representation Fusion . 11

2.3 Issue Trackers as Empirical Records of Work 12

2.4 Requirements Volatility and Specification Quality 14

2.5 Time-Aware Sequence Modeling 16

2.6 Dynamic Forecasting and Project-Control Literature 16

2.7 Project Memory, Cross-Project Learning, and Digital Accountability 17

2.8 Transferability, Benchmarking, and Organizational Context 19

2.9 Addressing Class Imbalance in SE Data 21
2.10 Large Language Models in Software Engineering 21
2.11 Evaluation Concerns in Predictive Software Engineering 22
2.12 Synthesized Taxonomic Literature Comparison 23
2.13 Systematic Gap Analysis and Dissertation Positioning 25

3 Theoretical Background **27**
3.1 Sequence Models for Task Histories 27
3.2 Models for Static and Sparse Inputs 28
3.3 Attention and Contextual Text Representations 29
3.4 Probabilistic Prediction and Calibration 30
3.5 Issue-Tracking Data as a Multimodal Process 31

4 Proposed Methodology **33**
4.1 Formal Problem Definition 33
4.1.1 Data Dictionary and Semantic Taxonomy 34
4.1.2 Prediction Semantics and Operational Interpretation 35
4.2 Feature Extraction and Embedding Pipeline 36
4.2.1 Dense Semantic Embedding (SBERT) 36
4.2.2 Sequence Windowing and Information Budget 36
4.2.3 Categorical and Numeric Normalization (Mathematical Feature Engineering) 37
4.2.4 Representation of Silent and Sparse Tasks 39
4.3 The Context-Aware Multi-Channel Architecture 40
4.3.1 Modeling Sequence Dynamics via GRU 40
4.3.2 Feature-Specific Time-Aware Attention 42
4.3.3 Interpretive Role of the Time-Aware Summary 43
4.3.4 Theoretical Bounds of the Logarithmic Penalty 44

4.3.5 Hierarchical Cross-Feature Attention for Context Fusion 45
4.3.6 Architecture Regularization: Cross-Head Decorrelation 47
4.3.7 Aggregation and Final Classification Output 48
4.4 Formal Computational Complexity and Gradient Analysis 49
4.4.1 Computational Complexity (Big-$\mathcal{O}$) Bounds 49
4.4.2 Gradient Flow and Decay Stability Analysis 50
4.5 Negative Results and Architectural Evolution 50
4.5.1 Failure 1: Naive Early Fusion 51
4.5.2 Failure 2: Time-Aware Multi-Head Attention without Decorrelation 51
4.6 Optimization Objective and Training Loop 51

5 Experimental Setup 53

5.1 Dataset Acquisition and Curation 53
5.2 Prediction Instances and Snapshot Construction 54
5.3 Feature Engineering and Distribution Analysis 55
5.3.1 Exploratory Data Analysis (EDA) 55
5.3.2 Project Heterogeneity within the Apache Ecosystem 57
5.3.3 Handling Class Imbalance 57
5.3.4 Multi-Collinearity Analysis via Variance Inflation Factor (VIF) . . 57
5.3.5 The Natural Language Processing Pipeline 58
5.3.6 Characteristics of the Dynamic Text (Comments) 59
5.4 Baseline Configurations 59
5.4.1 Traditional Machine Learning Baselines 59
5.4.2 Standard Deep Learning Baselines 60
5.4.3 Pretrained Transformer Encoder Benchmark 60
5.5 Evaluation Strategy 61
5.6 Hyperparameter Tuning Logs and Convergence Dynamics 63
5.6.1 Bayesian Optimization of the Topological Subspace 63

5.6.2 Analysis of the 3D Loss Surface Topography 64
5.6.3 Granular Hyperparameter Sensitivity Grids 64
5.6.4 Hardware Acceleration and Distributed Training (Data Parallelism) 66
5.6.5 Open Science and Algorithmic Reproducibility 67

6 Results and Analysis 68

6.1 Primary Evaluation I: Held-Out Project Performance 68
6.2 Statistical Significance and Effect Size Testing 70
6.3 Granular Cross-Sectional Analysis by Component and Severity 71
6.3.1 Performance by Architectural Component 71
6.3.2 Performance by Severity and Priority Level 72
6.4 Performance by Overall Task Lifecycle Duration 72
6.5 Sensitivity to Project Diversity (Robustness Analysis) 74
6.6 Cross-Ecosystem Generalization: Secondary Validation on GitHub 74
6.6.1 Secondary Dataset Demographics 74
6.6.2 Zero-Shot Transfer Results 75
6.7 Primary Evaluation II: Chronological Split 75
6.8 Probabilistic Calibration Analysis 76
6.9 Empirical Inference Latency and Environmental Impact 77
6.10 Data Efficiency and Learning Curve Analysis 78
6.11 Classifier-Only Efficiency with Precomputed Embeddings 79

7 Discussion and Implications 80

7.1 Systematic Feature and Module Ablation Study 80
7.1.1 Analysis of Feature Contributions 80
7.1.2 Validation of Architectural Innovations 81
7.1.3 How the Ablation Results Should Be Interpreted 81
7.2 Interpretability and Failure Analysis: Qualitative Case Studies 83

7.2.1 Case Study 1 (True Positive): A Qualitative Attention Pattern . . . 83
7.2.2 Case Study 2 (False Positive): Lexical Overreaction 84
7.2.3 Case Study 3 (False Negative): The Blindness of Silence 84
7.2.4 Case Study 4 (False Positive): Sarcasm and Non-Literal Language . 85
7.2.5 Case Study 5 (False Negative): Truncated Technical Context 85
7.3 Illustrative Deployment Scenario 85
7.3.1 Persona Definition and Workflow Scenario 85
7.3.2 Phase 1: Proactive Triaging 86
7.3.3 Phase 2: Evidence-Guided Follow-Up 86
7.3.4 Phase 3: Backlog Review 86
7.3.5 Illustrative Stakeholder Perspectives 86
7.4 Organizational Implications and Use Constraints 87
7.4.1 Use as Triage Support Rather Than Decision Automation 87
7.4.2 Threshold Selection as a Governance Choice 88
7.4.3 Human Oversight and Interpretive Limits 88
7.4.4 Adoption Costs and Data Preconditions 89
7.5 Threats to Validity 89
7.5.1 Construct Validity 89
7.5.2 Internal Validity 90
7.5.3 Statistical Conclusion Validity 90
7.5.4 External Validity 91

8 Conclusion and Future Work **93**
8.1 Summary of Research Outcomes 93
8.2 Avenues for Future Work 94
8.2.1 Ingestion of Direct Code Signals 95
8.2.2 Modeling Developer Social Topologies via Graph Neural Networks 95
8.2.3 Short- and Long-Duration Task Applicability 95

8.2.4 Pragmatic Tone and Speaker-Aware Modeling 95

A Standard GRU Formulation **97**

List of Figures

4.1 Illustration of the proposed context-aware architecture for predicting short-term task completion. The model uses different inputs in different channels, applies GRU + time-aware attention to dynamic sequences, and then uses a hierarchical cross-feature self-attention module designed with decorrelation regularization. 41

4.2 Cluster plots of correlation matrices across heterogeneous task features emphasize the need for structured rather than flat modal fusion. 46

6.1 Fold-level F1-scores from the 10-fold cross-validation procedure on the 30-project training partition used for statistical comparison. 71

7.1 Attention visualization for HADOOP-RW-32. The highlighted comments indicate inputs emphasized by the time-aware module in this example. . . . 83

List of Tables

2.1 Taxonomic Comparison of Representative Studies Relevant to Short-Horizon Software Task Prediction 24

4.1 Data Dictionary of heterogeneous feature space 34

5.1 Descriptive Statistics of the Combined Dataset (37 Apache Projects) 56

5.2 Evaluation Protocol Summary and Interpretation 62

5.3 Hyperparameter sensitivity grid A: F1 score for learning rate (η) and batch size (B) 65

5.4 Hyperparameter Sensitivity Grid B: F1 Score for attention head (H_{heads}) and decorrelation penalty (λ) 65

6.1 Average Performance on the Held-Out Test Set (7 Apache Projects) 68

6.2 Statistical Significance (Wilcoxon Signed-Rank with Holm-Bonferroni Correction) and Effect Size (Cliff's δ) 70

6.3 Cross-Sectional F1-Score by Jira Architectural Component 72

6.4 Cross-Sectional F1-Score and Recall by Task Priority 72

6.5 Performance by Overall Task Lifecycle Duration Under the Fixed 7-Day Prediction Task 73

6.6 Zero-Shot Transfer Evaluation: Apache-trained Models vs. React/Vue GitHub Issues 75

6.7 Probabilistic Calibration: Brier Scores on the Primary Test Set 76

6.8 End-to-End Computational Cost per Jira Task Update (Including On-the-Fly SBERT Embedding Generation) 78

6.9 Classifier-Only Computational Efficiency with Precomputed SBERT Embeddings 79

7.1 Ablation Study: Quantifying the Impact of Feature Modalities and Architectural Components (Evaluated on the 7 Test Projects) 80

Chapter 1 - Introduction

1.1 Background and Motivation

Short-cycle delivery has changed the role of estimation. In Scrum and Kanban settings, teams need more than a planning number; they also need to know whether an open Jira task is likely to finish soon enough to keep the sprint stable [11, 70, 1, 4]. Small forecast errors are enough to disrupt dependency chains, coordination routines, and process-timing decisions [13, 11, 79, 5].

The main issue is the gap between planned work and observed work. Agile estimation research repeatedly points to the same sources of bias: hidden dependencies, requirement changes, and uneven experience [81, 70, 13]. These signals often appear first in comment threads, status transitions, and issue-tracking activity before they become visible in formal planning artifacts. However, classic estimation support still says little about high-volatility software tasks that can change several times inside a one-week forecast window.

This dissertation uses issue-tracker data as the main source of evidence for prediction. It asks a direct question: given the available task metadata and the current state of developer discussion, what is the probability that an open task will be completed within the next 7 days, and what complementary delay-risk score follows from that estimate? Related work in IT operations and site reliability engineering has asked similar questions about incident-mitigation time under changing operational traces [10].

1.2 An Interpretive Perspective on Data-Driven Estimation

Historically, teams estimate tasks through expert judgment or simple retrospective averages, such as team velocity [16, 17]. These methods remain useful in practice, but they are subjective and do not always adjust well when project conditions change rapidly [18, 19]. Data-driven estimation expands the modeling space to include linear predictors, tree ensembles such as random forests and XGBoost, and traditional neural architectures [21, 22].

Recent reviews point to the broader use of attention-based models and learned representations in software prediction research [25]. Even so, issue-tracking records combine evolving text, structured metadata, and irregular timing patterns in ways that require models designed for that particular mixture of evidence.

The shift from human judgment to machine learning changes the evidence base more than it changes the underlying estimation problem [16]. Formula-based models such as COCOMO used relatively static descriptors such as size or function counts. This dissertation uses a different source of evidence: the changing task record inside the issue tracker, including text, labels, and timestamps.

The claim is narrow. For the evaluated dataset and prediction horizon, the model that preserves heterogeneous features and irregular temporal structure performs better than the selected baselines. It does not define a general theory of software labor. Instead, it estimates completion probability from recurring patterns in developer communication and task metadata. Any stronger claim would go beyond what one-week issue-tracker evidence can support.

1.3 Problem Statement

Many existing prediction models still miss the same representation problem. They regard tasks as fixed records or flatten text, labels and numbers into an input block. This flattening hides the interaction between content and time.

When software-task prediction is treated as an evolving state problem, the limitations of these methods become more obvious. The main difficulties are:

1. **High Dimensionality and Heterogeneous Data:** Software tasks combine unstructured text (title, description, comments), categorical labels (type, priority, component), and numerical features (story points, estimated time). When these inputs are mixed too tightly, their nonlinear relationships are easily flattened or hidden.

2. **Irregular and Sparse Time Series:** Task progression appears as an irregular se-

quence in which comments and other updates arrive at unpredictable intervals. That timing matters. Standard recurrent models and Transformers with fixed positional encodings are not naturally designed for non-uniform temporal gaps [26].

3. **Need for Contextual Holistic Representation:** Useful prediction requires the task to be modeled as a joint configuration of text, metadata, and temporal state rather than as isolated feature blocks.

1.4 Research Objectives and Questions

To address these issues, this dissertation proposes a deep-learning framework for short-term software-task completion prediction based on context-aware representation learning. The goal is to design an architecture that can model heterogeneous and time-dependent task data while remaining practical enough for asynchronous decision-support scenarios.

The focus of this work is driven by the following RQs:

- **RQ1:** How can irregular update intervals in software task communication be modeled to improve the prediction of task completion? Specifically, does a feature-specific time-aware attention mechanism outperform index-based positional encodings?

- **RQ2:** Does hierarchically combining the features through a multi-head self-attention layer improve prediction accuracy over simple feature concatenation?

- **RQ3:** How does the proposed framework balance predictive performance with computational efficiency when compared to standard sequence models and strong language-model-based baselines?

- **RQ4:** Can the attention-based components of the proposed framework provide qualitatively useful signals for interpretation in agile project management contexts?

1.5 Contributions

The main contributions of this dissertation are as follows:

- A multichannel architecture that processes unstructured text, categorical labels, and numeric features in parallel prior to fusion, preserving their distinct characteristics.
- A feature-specific time-aware attention mechanism that models irregular update timing through continuous elapsed time rather than index position alone.
- A hierarchical cross-feature multi-head self-attention mechanism, paired with a decorrelation loss, that models dependencies both within individual modalities and across them.
- An empirical evaluation on 37 open-source software projects, where the proposed framework delivers stronger performance than the selected baselines under the reported settings and the seven-day prediction horizon.
- A computational-efficiency analysis, along with qualitative case studies, that illustrates decision-support value while identifying practical conditions for deployment in new organizational settings.

1.6 Organization of the Dissertation

The rest of this dissertation is organized as follows:

- **Chapter 2** reviews the literature on software effort estimation, requirements volatility, issue-tracker analytics, time-aware sequence modeling, and related predictive software engineering work.
- **Chapter 3** condenses the technical background required for the proposed architecture.

- **Chapter 4** develops the mathematical formulation and architectural design of the model.
- **Chapter 5** details the dataset, preprocessing pipeline, baselines, and evaluation procedures.
- **Chapter 6** reports the quantitative findings, statistical comparisons, and robustness analyses.
- **Chapter 7** interprets those findings, presents qualitative case analyses, and examines validity threats and deployment implications.
- **Chapter 8** closes with the main findings and directions for future work.

Chapter 2 - Literature Review

Software engineering outcome prediction has progressed from parametric estimation models to statistical learning and, more recently, deep representation learning. This chapter reviews the literature most closely related to short-horizon software-task prediction and organizes it into eight overlapping categories: expert and agile effort estimation, requirements volatility, issue-tracker analytics, project memory and organizational learning, dynamic project forecasting, transferability and benchmarking, time-aware sequence modeling, and recent work on large pretrained language models. The review also identifies the specific gap addressed by this dissertation: repeated prediction from evolving issue-tracker state rather than one-time estimation from static planning artifacts.

2.1 Practical Motivation for Software Effort Estimation

The motivation for predictive modeling in software engineering has both practical and theoretical implications [11, 70, 2, 14]. Recent peer-reviewed work continues to treat effort and duration estimates as planning artifacts that support release scope, coordination, and process improvement rather than as isolated numerical outputs [4, 5]. Practitioner sources raise the same operational issue, emphasizing that agile delivery depends on repeated planning and review routines while broader project performance remains sensitive to scope, budget, and schedule pressure.

In a contemporary agile environment, these planning decisions are revisited repeatedly through backlog refinement, sprint planning, sprint reviews, and daily coordination, so even modest forecast errors can alter short-term work allocation [4, 11, 78, 15] [14, 28]. The value of forecast support in this setting is therefore less about producing a single duration estimate and more about improving situational awareness for work that has already begun.

2.1.1 Schedule Overruns and Planning Risk

When software work runs past its expected schedule, the effect rarely remains local to the delayed task. Recent research on inaccurate estimation and software project duration continues to show that schedule error persists in agile settings and is shaped by changing requirements, team experience, and hidden dependencies [81, 70, 13]. Practitioner evidence points in the same direction, continuing to associate scope creep and weak fit-for-purpose project management support with budget and schedule loss [6, 5]. In such conditions, an underestimated blocker can force sprint-level reprioritization and amplify coordination load across related tasks [11, 79]. Automated estimation is therefore not aimed at eliminating uncertainty. It is aimed at surfacing earlier signals of short-horizon completion risk.

2.1.2 Coordination Overhead and Limits of Late Intervention

Recent software engineering research also shows that schedule pressure is not reliably alleviated by adding contributors, because complex projects accumulate their own communication and coordination dependencies [78]. Empirical research in global software engineering shows that coordination overhead consumes substantial developer time, and uneven access to key personnel can become a real bottleneck [79]. Longitudinal evidence from large-scale agile projects points in the same direction: when many teams work in parallel under changing requirements and technical uncertainty, inter-team coordination remains a persistent source of friction.

This also explains why early risk detection is important. The signal is useful only if it appears before the task has visibly stalled. If the signal arrives early enough, the team can still adjust scope, re-prioritize work, request targeted review, or re-examine architectural options instead of relying on passive staffing changes.

As project-management practice and machine-learning methods have developed, the literature on software task estimation has also expanded.

2.2 The Chronological Evolution of Software Task Estimation

2.2.1 Era 1 (1980s - 2000s): Expert Judgment and Parametric Models

Early work relied heavily on parametric models such as COCOMO and on expert-judgment practices such as planning poker [16, 9, 60]. Those approaches set much of the field's early direction, but later reviews continued to return to the same weakness: in agile environments, requirements and implementation details can change during the sprint, which leaves estimation exposed to subjectivity and limited adaptability.

As software development became more agile, attention moved away from estimating long-duration projects and toward predicting the completion of shorter tasks or user stories [27, 3]. That shift mattered. It favored empirical, data-driven approaches over fixed parametric formulas.

The older methods still matter as an intellectual baseline for two reasons. First, they treated estimation as a coordination problem rather than a purely numerical problem. Second, they kept uncertainty communication in view. Modern machine-learning formulations inherit both concerns, even though the underlying modeling machinery now looks very different.

2.2.2 Structured Judgment Support and Estimation Bias

Follow-up research shows that expert evaluation is not a single stable cognitive behavior [69]. Usman et al. reported in a multiple-case study that checklist-supported estimates can improve task evaluations by prompting teams to state assumptions, scope boundaries, and risk factors that might otherwise remain implicit. This finding is important because it redefines some estimated errors as ignored considerations, not just a weak numerical technique.

Jørgensen and Halkjelsvik [71] further show that software-effort estimates are susceptible to sequence effects, meaning that the order in which candidate tasks or comparison examples are presented can shape later judgments. This result matters here because one-

shot planning estimates are vulnerable not only to missing information, but also to framing effects that build as examples are encountered. A system that updates forecasts as evidence accumulates can therefore address a real weakness in the estimation process rather than simply automating fixed expert judgments.

2.2.3 Era 2 (2010s): Statistical Machine Learning

As repositories and issue trackers became easier to mine, researchers turned to statistical learning methods such as Random Forests, Support Vector Regression, and XGBoost [80, 82, 28, 29]. When structured metadata, such as priority, component, or story-point fields, dominates the input, these models often outperform manual baselines.

One strength of this line of work is methodological discipline. Tabular models make feature effects easier to inspect, preprocessing alternatives easier to compare, and reproducible baselines easier to establish. Their main limitation appears elsewhere: heavy dependence on manual feature engineering constrains how much semantic content can be recovered from issue descriptions and conversational updates.

Work in the *Journal of Systems and Software* pushed this line toward agile, task-level settings. Dragicevic et al. [68] proposed a Bayesian-network model for task effort estimation in agile software development, illustrating the appeal of probabilistic dependency modeling when task information is incomplete or uncertain. Conoscenti et al. [70] then combined data analytics with developer feedback to identify reasons for inaccurate estimates in agile practice, suggesting that estimation error is often a socio-technical process phenomenon rather than the consequence of a single missing numerical predictor.

These studies link classical effort estimation with modern predictive analysis, but they are still closer to planning-time support than to continuous task-state prediction. Most operate on task descriptors or retrospective project evidence rather than on a temporally ordered record of descriptions, comments, reassignments, and status changes collected while work is underway.

Recent peer-reviewed journal work has continued to narrow the gap between early-phase planning and data-driven estimation in agile contexts. Pospieszny et al. [81] showed that machine-learning models can be used to estimate both software effort and software duration, reinforcing the practical point that schedule forecasting and effort forecasting should be treated as related but distinct targets. More recent studies by Rosa and Jardine in the *Journal of Systems and Software* examine estimation under government acquisition constraints where familiar agile measures such as story points are often unavailable at the moment decisions must be made [20, 2]. Their work compares alternative early sizing measures and proposes functional-initiative-based representations to support early agile effort estimation before mature delivery artifacts exist.

Recent work in *Information and Software Technology* and *Expert Systems with Applications* also suggests that context can matter as much as model class [13]. Alsaadi and Saeedi studied overall workload estimation for novice agile teams at the user-story level, while Harish Kumar and Srinivas proposed a hybrid rule-based and recurrent-network method for software effort estimation in project-management environments [40]. Despite their architectural differences, these studies point to a shared conclusion: software effort remains difficult to estimate because project characteristics interact nonlinearly and evaluation conditions vary with team maturity, delivery setting, and data quality. This supports the dissertation's focus on learning from evolving task evidence rather than treating static planning artifacts as sufficient.

Contemporaneous retrospective work makes this point clearer [19]. In an updated systematic review of ensemble effort estimation, Cabral et al. conclude that combined learners often improve robustness, but that reported benefits remain highly sensitive to dataset composition, preprocessing choices, and target definition. In other words, better estimation accuracy depends not only on choosing stronger algorithms, but also on managing heterogeneity in the evidence source. This result matters to this dissertation because it shows that model diversity alone does not eliminate the need for careful evaluation under project-to-

project variation. A method may appear strong under one corpus design but remain fragile when transferred to a different issue-tracking environment.

2.2.4 Era 3 (2018 - Present): Deep Learning and Representation Fusion

The third era centers on deep learning and representation fusion [30, 83, 84, 31, 32] [42, 41].

At the same time, there is still a common limitation in this literature: many models treat task states as static snapshots or combined heterogeneous inputs, without explicitly modeling the evolution of issue threads during active work.

As representation-learning systems became more common, the prediction unit changed as well. The problem is no longer limited to one-time effort estimates obtained from planning artifacts; in many newer studies, the model is asked to re-evaluate task status each time new evidence enters the record. This shift is particularly relevant to issue trackers, where comments, reassignment events, and workflow transitions typically contain information that cannot be observed at the time of task creation.

Recent peer-reviewed work also shows that the text itself contains software-specific structure that general bag-of-words models or static sentence embeddings do not always recover well. Gomes et al. [33] compared BERT-based and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS projects and showed that richer semantic representations can improve predictive modeling when longevity-related signals are embedded in narrative reports. This finding is encouraging, but caution is still necessary: performance depends on the label definition and vocabulary patterns of the target project. Giray et al. [21, 8] likewise report in software-prediction review work that deep learning holds clear promise for defect-oriented prediction, while also documenting recurring problems in data preparation, interpretability, and cross-study comparability. These observations support the dissertation's emphasis on disciplined evaluation rather than assuming that a deeper model can solve the problem by itself.

Work in complex and intelligent systems further demonstrates that software language modeling benefits from domain-aware representations [39, 94]. Tang et al. show that combining syntactic information with semantic enhancement improves software knowledge-entity extraction, indicating that software artifacts contain structured technical concepts that may be missed by generic labeling. Wang et al. similarly use graph convolutional networks for inter-version defect prediction and show that software artifacts can be correlated across versions rather than treated only as isolated text samples [53]. These studies are not direct solutions to short-term task completion prediction, but together they reinforce a key modeling premise of this dissertation: software work records are semantically rich, structurally heterogeneous, and only partially captured by flat text encodings or one-time feature vectors.

2.3 Issue Trackers as Empirical Records of Work

Issue trackers sit somewhere between two weaker extremes. They retain the explicit structure typically lacking in informal conversations, but they also preserve more process detail than static planning spreadsheets or repository-level commit summaries. Typical issue records combine initial problem statements, classified metadata, status and ownership revisions, and evolving discussions among developers, reviewers, and project coordinators. This combination makes the issue tracker well suited for studying short-term work progress, not just one-off project planning.

From a research standpoint, two attributes are especially important. The first is temporal continuity: intermediate task states remain visible, so prediction can be repeated as work progresses rather than deferred until closure. The second is accessibility. Public issue trackers such as Jira and Bugzilla have supported a substantial empirical literature because task content and workflow history can be collected at scale [58, 84]. These two attributes are central to this dissertation, which frames the prediction problem around evolving task state rather than one-time estimation at project inception.

At the same time, issue trackers are not neutral tools. Teams differ in how often they comment, how they use priority tags, when they close tasks, and how much coordination occurs outside the tracker. Some projects document technical discussion extensively in comments, whereas others rely more heavily on external chat, meetings, or code-review systems. For this reason, issue-tracker data should be understood as a rich but partial representation of project activity. This limitation does not make the data unusable, but it does make cross-project evaluation and validity discussion particularly important.

The above literature also indicates that issue trackers can externalize project knowledge at multiple levels of abstraction. The initial summary typically expresses expected needs or defect symptoms, comments capture evolving diagnostic reasoning, and field updates encode workflow outcomes from those discussions. Studies on bug longevity, entity extraction, and cross-version defect prediction [33, 39, 53] indicate that these layers carry complementary rather than redundant information. A practical implication is that short-horizon prediction should not rely only on opening issue text or on static categorical fields. Instead, as task records develop, prediction should use the interaction among narrative content, event sequence, and project metadata.

Recent requirements engineering work has also clarified the relationship structure of issue repositories [75]. Lüders et al. analyzed public Jira repositories and found substantial heterogeneity in how link types are used, the completeness of link data, and which link categories can be reliably predicted from issue text. Their BERT-based model performs particularly well for structural relationships such as Epic and Subtask, while more general links such as Relate remain ambiguous and repository dependent. For this dissertation, the lesson is broader than link prediction itself. Issue trackers encode task status partly through relationships between issues, and those relationships are governed by local documentation conventions. Ignoring this heterogeneity may cause a prediction model to confuse repository-specific linking practice with a general signal of task difficulty or progress risk.

This observation connects to a second strand of literature that treats project records as

recoverable organizational memory [12]. Abdellatif et al. developed a method for automatically recalling software lessons learned and showed that project managers can retrieve previously documented experience from textual repositories to support current decision-making. Their setting differs from short-horizon task prediction, but the implication is closely related: narrative project records are more than historical archives. They can also function as operational memory when new work resembles prior episodes. This reinforces the dissertation's use of issue histories as evolving evidence sources. Comments, summaries, field changes, and linked issues do not merely document completed work; they preserve fragments of prior reasoning that can inform new predictions when interpreted in context.

2.4 Requirements Volatility and Specification Quality

Any task completion model also inherits the uncertainty of the requirements process [67]. In Requirements Engineering, McGee and Greer analyze the causes and effects of requirements changes and show that volatility depends not only on the complexity of the implementation, but also on the dependencies between the source of the change and the requirements. Their findings are important here because task delays often reflect changing dynamics upstream rather than just downstream coding efforts.

More recent evidence suggests that many of the input artifacts remain noisy and incomplete in practice. Franch et al. [72] found that practitioners still rely primarily on natural-language specifications supported by general-purpose templates and tools, while ambiguity, inconsistency, and incompleteness remain common challenges. This observation supports a core modeling choice in the present dissertation. If software work continues to be documented largely in natural language and semi-structured records, then contextual text modeling is not an optional embellishment; it is one of the few ways to capture information that is operationally important but poorly normalized.

Recent work in requirements engineering has also shown that natural-language struc-

tures carry information about downstream process behavior [73]. Frattini et al. report the prevalence of causal relationships in requirements artifacts, develop an automatic detector for such relationships, and show that causal-relationship representation is related to lead time and volatility in the requirements life cycle. For this dissertation, the focus is not the detector itself but the broader result: wording carries process information. Text representations are more than descriptive noise surrounding a task. They can encode signals about ambiguity, expected behavior, and the number of iterations a requirement may go through. As the task narrative becomes more specific, this finding strengthens the rationale for contextual issue-text representation and repeated prediction.

Abbas et al. [74] extend this point by examining whether similar requirements correspond to similar software in an industrial setting. Studying railway requirements and their linked implementations, they found a moderately positive relationship between requirements similarity and software similarity, with BERT-based representations outperforming more traditional alternatives. Just as importantly, practitioners in their study emphasized that similarity judgments depend on trace links, domain knowledge, and test evidence in addition to textual resemblance. For this dissertation, that result is useful in two ways. It supports contextual text modeling as a realistic way to detect downstream process implications from upstream language, but it also cautions against treating text alone as a complete proxy for implementation effort. Multimodal evidence remains necessary.

Together, the requirements literature complements the issue-tracker literature in an important way. Issue histories record how tasks evolve after they enter the workflow, while requirements-engineering studies explain why the input state is often unstable to begin with. The combination strengthens the case for repeated prediction from evolving task state instead of a single estimate generated at task creation.

2.5 Time-Aware Sequence Modeling

Temporal dynamics are central to prediction problems based on sequential updates. Standard Transformer architectures [26] and gated recurrent sequence models remain widely used for predictive monitoring in recent journal work [52, 40], and recent studies have proposed time-aware variants for settings with explicit temporal structure [43, 44]. Related domains such as predictive maintenance have shown the value of combining sequence representations with explicit time information [45, 46].

Software task prediction differs from many of these settings because issue updates are sparse and irregular. A task might sit idle for days and then receive multiple comments in quick succession. Under these conditions, sequence index and elapsed time are not interchangeable. Two adjacent comments may be separated by a few minutes or by several weeks, but standard positional encoding processes them in a roughly similar way. This mismatch motivates the architecture to model elapsed time directly instead of relying solely on sequence order.

This distinction has practical implications. In issue-tracking settings, a series of closely spaced comments may indicate ongoing debugging or review. Long silence may mean something completely different: waiting, blockage, or work moving off-platform. Models that only see comment order cannot clearly distinguish these states. This is why the dissertation treats elapsed time as part of the representation problem rather than as metadata added after the fact.

2.6 Dynamic Forecasting and Project-Control Literature

Relevant project management and operations planning literature makes similar arguments at a larger unit of analysis. Vandevoorde and Vanhoucke compare project-duration forecasting methods based on earned value measurement and show that forecasting usefulness depends on how progress information is incorporated during execution, not only on the baseline plan [62]. In related work, de Mello and Sanchez use a Bayesian network to pre-

dict delay in software maintenance projects and frame schedule disruption as a probabilistic control problem under uncertainty [61].

Recent predictive research has also become more explicit about joint uncertainty across outcomes. Ünsal-Altuncan and Vanhoucke [76] proposed a hybrid Bayesian-network model for simultaneously predicting duration and cost performance. Although these studies operate above the level of individual issue-tracker tasks, their logic is closely consistent with this dissertation: prediction is most useful when it is updated as evidence arrives, uncertainty is represented explicitly, and the output supports intervention before failure becomes obvious. This broader perspective is also consistent with recent project-management syntheses on AI-enabled decision support and project control [17, 28].

This adjacent literature also clarifies a boundary. Project-control models typically look at coarser summaries such as work-package progress, budget, or milestone achievements. As such, they provide strong intuition about dynamic prediction but have limited coverage of textual and conversational signals present in software issue trackers. This dissertation extends dynamic prediction ideas to finer temporal and semantic granularity.

2.7 Project Memory, Cross-Project Learning, and Digital Accountability

Recent project management literature further clarifies why predictive support should be embedded in organizational routines rather than viewed as stand-alone algorithms [63]. Mariano and Avazu analyzed project memory in large-scale information systems implementations and argued that memory is enacted through ongoing practices such as situational focus, synchronized discourse, and convergent boundaries. Their argument is valuable here because it draws attention away from the idea that knowledge is passively stored in repositories. Digital records only become useful when teams revisit, interpret, and coordinate them. In software issue trackers, the same principle applies: the value of a task history depends not only on its existence, but also on whether the evolving record remains coherent enough to support later reuse, diagnosis, and prediction.

Zhang et al. make a complementary argument from the perspective of knowledge governance and inter-project learning [64]. Using survey and interview evidence, they show that knowledge-governance mechanisms can strengthen individual learning across projects and that relational capital helps translate stored knowledge into actual learning behavior. The implication for predictive analytics is straightforward. Model portability is not only a technical transfer-learning problem. It is also an organizational learning problem: teams benefit more from historical project data when they have norms and relationships that help them interpret and act on those traces. This provides another reason to evaluate predictive models under held-out project settings rather than assuming that archived data automatically transfer across contexts.

Digital tools also shape content that is visible enough to be learned [65]. In a qualitative study of project teams, Zhang shows that digital technologies influence accountability through visibility and associational affordances, but that both too little and too much transparency can be problematic. This finding is important for software issue trackers because logged signals are in part a product of accountability practices. Teams may document heavily when visibility is needed, under-document when coordination occurs elsewhere, or adopt linking and commenting conventions that make sense locally but are not easily comparable across repositories. This sociotechnical perspective reinforces the recurring theme in this dissertation: issue-tracker data are rich and operationally useful, but they are influenced by organizational incentives and therefore must be carefully explained in the transferability statement.

Olsson et al. [66] likewise show that benefits realization depends on organizational integration mechanisms linking projects, functions, and executives during execution, including feedback loops from ongoing work into later project decisions. Their unit of analysis is larger than an individual software task, but the logic transfers directly. Predictive models create value when they enter those feedback cycles early enough to support reprioritization, scope adjustment, or targeted intervention. In that sense, short-horizon task prediction can

be read as a fine-grained instance of the broader project-control problem explored in the project-management literature.

2.8 Transferability, Benchmarking, and Organizational Context

Another topic in recent literature is whether predictive models remain effective across project, organizational, and technical environments [7]. Yurum et al. argue that benchmarking remains a central bottleneck for machine-learning-based effort estimation because comparisons become unstable when models are evaluated on narrow or inconsistent corpora. This question is central to this dissertation. If short-term task prediction is intended to support real delivery settings, then model quality must be judged not only by how well it fits within the sample, but also by how well learned representations hold up as project vocabulary, workflow styles, and team practices shift.

Related evidence from adjacent information-systems literature points in the same direction. Weinzierl et al. [77] show in *Business & Information Systems Engineering* that transfer learning can support predictive process monitoring across intra- and inter-organizational IT service management processes, enabling useful predictions even when the target context lacks enough local training data to build a strong model from scratch. Their setting is business-process event logs rather than software issue trackers, but the logic is similar. Prediction is treated as operational support for a process already in motion, while transferred models, encodings, and preprocessing choices are treated as reusable organizational resources rather than one-off experiments.

The same operational problem also appears in studies of contemporary software delivery settings [15]. Ünlü et al. report that microservices-based agile projects change decomposition practices, ownership boundaries, and coordination patterns. In practice, this means that the semantics of "tasks" may shift before organizations can make meaningful comparisons across contexts. If a model trained in one environment depends too narrowly on local vocabulary or project-specific workflow conventions, it may not perform well in another

environment. This makes held-out project evaluation and secondary validation particularly important in this dissertation.

Management-oriented journal work further supports this operational framing. Müller et al. [17] characterize AI in project management as a decision-support capability rather than a replacement for managerial judgment, while Uysal and Akturk [28] emphasize lifecycle-wide system design for machine-learning-enabled project management. Koudriachov et al. [1] likewise argue that contemporary agile project-management research should be read in relation to planning practice and organizational learning rather than only algorithmic accuracy. Taken together, these studies show that the core problem is not only whether a model predicts correctly within one repository. The question is also whether the model can fit iterative planning and monitoring routines without imposing excessive data requirements, latency, or organizational cost.

This organizational concern is echoed in recent review literature on software analytics [19]. Cabral et al. and Giray et al. both observe that benchmark instability, inconsistent labeling, and uneven reporting practices make it hard to know whether reported performance gains reflect genuine modeling progress or merely dataset-specific advantages [21]. The critique matters even more when models are trained on issue-tracker data because workflow conventions, documentation density, and team vocabulary vary sharply across repositories. For that reason, this dissertation treats held-out project evaluation as more than a methodological preference. It is a direct response to the broader concern that predictive claims become overstated when portability across organizational settings is left untested.

The same idea also helps explain why transfer failures are often social as well as statistical. If one repository uses issue links to formalize dependencies, another uses them only rarely, and a third handles the same coordination work through meetings or chat, then model generalization will be limited by differences in project memory, accountability, and learning routines as much as by word choice alone [75, 63, 65]. A transferable predictor therefore requires an evaluation setting that exposes these organizational differences rather

than smoothing them away through random mixing.

2.9 Addressing Class Imbalance in SE Data

Class imbalance is a recurring challenge in empirical software engineering datasets. In many issue-tracking corpora, routine tasks are much more common than delayed or otherwise difficult tasks [59]. Under such conditions, accuracy alone can be misleading because a majority-class predictor may appear strong while remaining uninformative for risk-oriented decision making.

Earlier work often addressed this problem through synthetic resampling methods such as SMOTE [59]. These methods can be effective for tabular data, but they are less well matched to multimodal text sequences because interpolated text representations may distort the underlying semantics. More recent approaches therefore tend to emphasize cost-sensitive objectives, threshold selection, or evaluation metrics such as F1 and PR-AUC rather than synthetic generation alone [89].

2.10 Large Language Models in Software Engineering

Recent software engineering research has examined pretrained language models, code-oriented foundation models, and more recent large language models across a broader set of tasks than bug-time prediction alone. CodeBERT and related encoder-style models established early evidence that pretrained representations can transfer effectively to software and natural-language artifacts [47]. More recent journal work in *Information and Software Technology* and the *Journal of Systems and Software* extends this direction to test generation, test report prioritization, coding assistance, AI pair programming, bug-fixing-time prediction, and assurance-case instantiation [34, 36, 35, 24, 23, 48].

This broader literature shows that LLMs are useful in software engineering for at least three different reasons. First, they can generate or rank artifacts from natural-language prompts, as seen in LLM-based test generation and crowdsourced test report prioritization

[34, 36]. Second, they can participate interactively in development work, which has motivated recent studies of AI coding assistants and AI pair programming in practice [35, 24]. Third, they can support specialized reasoning tasks over engineering documentation, as shown in assurance-case instantiation and bug-related narrative analysis [23, 48]. These studies make it harder to treat LLMs as a narrow fringe topic in software engineering. They now cross multiple points in the development lifecycle.

At the same time, the recent literature also cautions against treating LLM performance as stable or costless. Kim and Ming [38] report that output reliability and similarity vary substantially across models and software-development tasks. Qu et al. [37] review backdoor attacks and defenses in code LLMs and argue that security and governance remain open concerns even when functional performance looks promising. These findings sharpen the boundary of the present dissertation. The relevant question is not whether LLMs are useful in software engineering broadly, but whether their additional latency, cost, reliability variation, and security exposure are justified for repeated short-horizon task-state updates under the reporting cadence typical of issue trackers.

It is also useful to distinguish between encoder-style pretrained models, decoder-style LLMs, and code-specific foundation models. All three can be strong. However, when deployment cost, controllability, and evaluation design come into focus, they are not interchangeable. Much of the recent literature still focuses on code generation, testing, vulnerability analysis, or one-off text classification rather than repeated multimodal task-state updates under tight latency budgets. This gap is one reason this dissertation compares against an LLM-style baseline without assuming that the largest available model is automatically the best operational choice.

2.11 Evaluation Concerns in Predictive Software Engineering

The literature also suggests that evaluation design is exceptionally important in this field. Many software engineering datasets are heterogeneous, imbalanced, and temporally struc-

tured, so nominally high performance can be difficult to interpret if the split strategy does not match the intended use case. A model that performs well under random or time-mixed resampling may still degrade sharply when evaluated on later time periods or on projects with different communication conventions.

This problem becomes more serious when the prediction unit is not a single item but a task state observed at multiple moments [88]. Repeated snapshots from the same task can preserve statistical dependencies, and time-mixed evaluation can expose the model to vocabulary or workflow patterns that would not yet exist in a forward-looking deployment. Recent methodological work in empirical software engineering therefore argues that the unit of analysis, split strategy, and statistical test should be matched carefully to the claim being advanced.

The same logic applies to metrics [59, 89]. Under class imbalance, accuracy can flatter a model that rarely identifies the outcome that matters most. F1-score and PR-AUC are usually more informative when the positive class is relatively rare or operationally salient. Once predictions are reported as risk scores, calibration matters too: ranking alone is not enough if the reported probabilities cannot be interpreted within the limits of the evaluation design.

These concerns shape the dissertation's experimental design rather than appearing later as after-the-fact caveats. The held-out project split, the chronological split, the project-level cross-validation procedures, and the calibration analysis were chosen to keep the empirical claims proportionate to the evidence actually produced.

2.12 Synthesized Taxonomic Literature Comparison

Table 2.1 summarizes representative studies on software-related task prediction and adjacent estimation settings. The comparison covers dataset scale, input modality, core architecture, and the main reported strengths and limitations.

Table 2.1: Taxonomic Comparison of Representative Studies Relevant to Short-Horizon Software Task Prediction

Study (Year)	Dataset Scale & Origin	Input Modality	Core Architecture	Contribution & Key Limitations
Jorgensen (2004) [60]	Industrial effort-estimation studies	Expert judgment	Review of judgment-based estimation	Establishes the persistence and variability of expert estimation. Not designed for continuous automated prediction.
de Melo and Sanchez (2008) [61]	Software maintenance projects	Project descriptors	Bayesian network	Treats delay as an uncertainty-management problem. Coarse project granularity and little text.
Porru et al. (2016) [84]	Agile issue trackers	Issue report text	Classical text-based estimation	Early evidence that issue text can support estimation. Limited contextual and temporal modeling.
Dragicevic et al. (2017) [68]	Agile development tasks	Structured task descriptors	Bayesian network	Interpretable probabilistic effort estimation. Limited free-text and ongoing-update support.
Choetkiertikul et al. (2019) [83]	Jira issue reports	Static text and metadata	Deep learning for story-point estimation	Stronger semantic modeling than classical baselines. Still based on largely static issue snapshots.
Conoscenti et al. (2019) [70]	Agile estimation records	Analytics plus developer feedback	Mixed-method estimation analysis	Identifies recurring reasons for inaccurate agile estimates. Diagnostic rather than sequential predictive model.
Abbas et al. (2023) [74]	Industrial requirements and linked source code	Requirements text plus trace-linked software	Similarity modeling with lexical and deep NLP baselines	Shows that requirement similarity can proxy software similarity. Still needs trace links and domain knowledge for reliable reuse judgments.
Lüders et al. (2023) [75]	15 public Jira repositories	Paired issue titles, descriptions, and link labels	BERT-based typed-link prediction	Reveals heterogeneity in issue-link semantics across repositories. Strong for structural links, weaker for ambiguous relations.
Du et al. (2022) [41]	Open-source bug repositories	Text plus sequential bug history	Sequence-fusion network	Demonstrates gains from modeling bug-history sequences. Specialized to bug-resolution settings.
Wang et al. (2023) [10]	Incident-management records	Text, metadata, mitigation history	Incident-time prediction pipeline	Extends prediction from issue creation to response duration. Focuses on incidents rather than general tasks.
Kula et al. (2023) [9]	Software-project delay records	Delay patterns over time	Bayesian dynamic prediction	Moves beyond one-shot duration estimation toward repeated delay prediction. Uses coarser delay signals than full multimodal task state.
Ardimento et al. (2025) [23]	Jira and Bugzilla archives	Narrative bug text	LLM-based classifier	Shows the promise of large language models for bug-fixing-time prediction. High latency and multimodality-integration challenges.
Proposed Approach	**3.7M+ raw mined Apache Jira events before task/snapshot filtering**	**Heterogeneous (Static, Dynamic, Meta)**	**Time-Aware Attn + Hierarchical GRU**	**F1: 0.769. Explicit irregular temporal fusion.**

2.13 Systematic Gap Analysis and Dissertation Positioning

Taken together, the studies in Table 2.1 point to six recurring gaps:

1. **Incomplete Multimodal Integration:** Many prior models emphasize either text or structured metadata, but not their interaction.

2. **Upstream Requirements Volatility:** Many studies treat the task definition as stable even though specification ambiguity and requirement change continue after planning.

3. **Irregular Temporal Gaps:** Many sequence models capture order more directly than elapsed time, which leaves sparse issue-tracker histories only partially represented.

4. **Transferability, Benchmark Fragility, and Organizational Mediation:** Recent work increasingly shows how hard it is to compare models across inconsistent corpora and how easily project-specific signals can fail to generalize when documentation, linking, and accountability practices differ.

5. **High-Cost Language Models:** Very large pretrained models can be effective, but repeated re-evaluation workflows may not justify their runtime and infrastructure cost.

6. **Evaluation Under Imbalance:** When outcomes are skewed, accuracy alone becomes a weak basis for comparison.

Against this background, this dissertation evaluates a heterogeneous, time-aware neural architecture that combines contextual text representations, structured metadata, and explicit modeling of irregular time gaps. The contribution claimed here is methodological. The empirical question is direct: does that combination improve short-horizon completion prediction relative to the selected baselines under the reported evaluation settings? The later chapters address that question while keeping the operational claims inside a research-prototype frame.

More specifically, this dissertation draws on five bodies of literature: software effort estimation, requirements and task volatility, predictive monitoring and transferability, multimodal issue-text modeling, and time-aware sequence learning. The discussion above indicates that each area contributes part of the necessary machinery, but no previous work has fully solved the problem of repeatedly predicting near-term task completion from evolving issue-tracker state. This combined framing motivates the architecture and evaluation strategy developed in the remaining chapters.

Chapter 3 - Theoretical Background

This chapter presents the technical background required for the proposed architecture. The scope is deliberately narrow: it retains the concepts used in later chapters and leaves aside broader tutorial material that is not needed for the dissertation's model design.

3.1 Sequence Models for Task Histories

Software tasks evolve through ordered updates. If descriptions, comments, and status changes are treated as independent observations, the temporal context of escalation, blockage, or recovery is lost. Recurrent models address this problem by maintaining a hidden state across the sequence:

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \tag{3.1}$$

Standard RNNs suffer from vanishing-gradient limitations when dependencies span multiple time steps [85, 92]. For this reason, practical sequence models often rely on gating variants such as LSTM and GRU-based encoders. The LSTM introduces separate memory and gating mechanisms to preserve distant information, while GRU-based models offer a simpler alternative with fewer parameters [52, 40]. When sequence order is important but computational cost must remain modest, recent journal models continue to use GRU-based designs. This dissertation uses a GRU because it provides an efficient balance between expressive power and computational cost for moderately long issue-tracker comment histories.

The GRU update can be summarized as:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \tag{3.2}$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t] + b) \tag{3.3}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{3.4}$$

where the update gate z_t controls how much new information replaces prior state. In the proposed system, this recurrent encoder provides the evolving representation of the comment sequence before later attention layers reweight that history.

This recurrent inductive bias is useful for issue-tracker data because the relevant histories are usually moderate in length rather than arbitrarily long. A task may accumulate a handful or a few dozen comments, and the main modeling question is often how those comments alter the current state estimate rather than how to preserve every token uniformly across a very long document. For that reason, a GRU-based backbone provides a reasonable starting point for the dynamic-text channel in this dissertation.

3.2 Models for Static and Sparse Inputs

Not all relevant software project signals are continuous [91]. Priority, components, story points, issue types, and similar fields are all structured metadata. For such inputs, tree-based ensemble methods remain a strong baseline in empirical software engineering. XGBoost is particularly useful because it combines additive trees with regularized optimization:

$$\mathscr{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) \tag{3.5}$$

XGBoost is therefore a reasonable baseline for structured Jira metadata. However, it is not a complete solution once prediction relies on long free-text descriptions, evolving comment histories, and interactions across modalities. These dependencies are difficult to express in a purely tabular pipeline.

For textual baselines, sparse lexical representations such as Bag-of-Words and TF-IDF remain useful reference points because they are simple, fast, and often competitive on limited data. Distributed word-vector methods such as GloVe provide another historical baseline for dense lexical representation, although they still lack the context sensitivity needed for evolving issue discussions [86]. Their main limitation is that they ignore the sequence order and capture very little contextual meaning beyond the surface term statistics.

These simpler baselines are still important. They anchor empirical comparisons, making it more difficult for people to mistake the novelty of the architecture for real performance improvement. They also correspond to the workflows that organizations can implement through moderate engineering work, making them useful low-complexity reference points.

3.3 Attention and Contextual Text Representations

Modern NLP systems often rely on attention to model interactions within a sequence [26, 51]. Attention allows the model to reweight elements according to their relationship with the current representation state. Scaled dot-product attention is defined as:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.6}$$

This setting is only partially applicable to issue-tracking data. In Jira-like systems, adjacent comments may be separated by minutes or by weeks, so index distance is not a reliable substitute for elapsed time. This is the motivation for the time-aware formulation introduced in Chapter 4, where temporal spacing is modeled directly rather than inferred from position alone.

Text representation matters as well. Technical issue text mixes ordinary language with identifiers, stack traces, abbreviations, and project-specific vocabulary [93]. Sparse lexical features capture part of this signal, but context encoders often match it better. In this dissertation, titles, descriptions, and comments are embedded with Sentence-BERT (SBERT),

which extends BERT-style contextual pretraining into fixed-size sentence representations while remaining tractable for large numbers of task updates [87, 49]. The contribution is therefore not a new encoder. It lies in how contextual text representations are combined with temporal sequence modeling and structured metadata.

Using a frozen sentence encoder comes with a trade-off rather than a free gain. In some settings, end-to-end fine-tuning may improve task-specific accuracy, but at the cost of being more computationally intensive and less stable for repeated evaluation across many task snapshots. In this dissertation, freezing SBERT keeps the text representation stage fixed, which makes ablation and comparison studies easier to interpret.

3.4 Probabilistic Prediction and Calibration

For decision-support purposes, probabilities often provide more information than hard class labels. A binary decision indicates whether the prediction falls within the target horizon; the probability indicates how strongly the observed evidence supports that prediction. This distinction matters when teams must allocate limited attention across tasks that carry different levels of risk.

In supervised classification, probabilistic predictions are commonly trained through logistic output layers and cross-entropy loss. Let $\hat{p}_i$ denote the predicted probability that task i completes within the horizon and let $y_i \in \{0,1\}$ denote the observed outcome. The standard binary cross-entropy objective is:

$$\mathscr{L}_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)] \tag{3.7}$$

This objective encourages correct classification, but it does not by itself guarantee that the resulting probabilities will be well calibrated.

Calibration refers to the consistency between the prediction probability and the observed frequency [90]. If tasks predicted at roughly 0.8 completion probability do in fact complete near 80% of the time under evaluation, the model is reasonably calibrated for that

setting. This matters here because the model is framed as decision support rather than as a fully automated classifier. Later chapters therefore evaluate not only classification metrics such as F1-score and PR-AUC, but also whether the probability outputs behave plausibly as short-horizon completion estimates.

Decision thresholds need not be fixed at 0.5, and in practice they should rarely be treated as universal. The cost of missing a likely delay may exceed the cost of reviewing a task that is later completed on time. Since these losses are asymmetric, ranking quality, calibration, and threshold selection are all important for practical use. Therefore, this dissertation retains continuous completion probabilities and, when useful for operations, interprets them supplementally as delay-risk scores.

3.5 Issue-Tracking Data as a Multimodal Process

Issue trackers such as Jira organize project work through records that evolve over time. Each task blends several distinct kinds of information:

- **Initial context:** title, description, priority, issue type, component, and estimated effort.
- **Dynamic discussion:** comments and status changes accumulated during implementation.
- **Outcome information:** timestamps and workflow transitions showing whether the task closed within the evaluation horizon.

Therefore, the prediction problem is multimodal, not just textual. Useful models must combine structured metadata, contextual language, and the temporal evolution of tasks. The architecture introduced in the next chapter is designed around this combination rather than around any single data type.

This framing also explains why prediction is difficult. The issue tracker records only part of the work: some coordination appears in comments, some appears in metadata

changes, and some does not enter the tracker at all. The goal is not to reconstruct the entire organizational process, but to learn from the observable multimodal traces in the dataset. This view motivates the architecture and explains why later chapters still need to discuss validity and deployment limits.

This partial visibility also helps to explain why early integration of modalities usually does not perform well. The statistical structure of priority labels is different from the statistical structure of technical discussions or a series of irregular comments. Treating all inputs as homogeneous features may obscure the role played by each pattern. Therefore, this architecture defers fusion until modality-specific processing produces a more suitable intermediate representation.

Chapter 4 - Proposed Methodology

The central hypothesis of this dissertation is simple: architectures designed for heterogeneous software task data and irregular temporal updates should outperform simpler baselines in short-time completion predictions. This chapter translates this statement into specific model specifications. The focus is on feature-specific time modeling, controlled multimodal fusion, and the assumptions required to convert issue tracker history into predictive representation.

4.1 Formal Problem Definition

Within agile project management, the immediate objective is to evaluate software task i at its current state and estimate the probability that it will be completed within a short horizon H. In this study, $H = 7$ days. That window was chosen because it approximates the short planning interval that often matters in sprint-level coordination.

Let X_i denote the complete feature space of task i. Because software tasks mix several kinds of evidence, X_i is not treated as a single homogeneous vector. It is decomposed into three feature modalities with different statistical roles:

1. **Dynamic Textual Features (X_i^{DT}):** These encode the evolving dialogue, debug logs, and status-related discussion generated during work. Formally, this modality is represented as a sequence of comments $C_i = \{c_{i,1}, c_{i,2}, \ldots, c_{i,T_c}\}$ with corresponding timestamps $T_i^c = \{t_{i,1}^c, t_{i,2}^c, \ldots, t_{i,T_c}^c\}$.

2. **Static Textual Features (X_i^{ST}):** These provide the baseline semantic context of the task. They include the title ($Text_{title}$) and detailed description ($Text_{desc}$), both of which are usually established at task creation and then change little.

3. **Structured (Numeric & Categorical) Features (X_i^S):** These encode the formal metadata of the task. The set includes priority ($Cat_{priority}$), issue type (Cat_{type}), es-

timating variables such as story points (Num_{sp}), and the task creation timestamp ($Time_{create}$).

4.1.1 Data Dictionary and Semantic Taxonomy

Table 4.1 defines the extracted features and their structural origins within the Jira data.

Table 4.1: Data Dictionary of heterogeneous feature space

Feature Group	Jira Field Name	Type	Semantic Description & Distribution Notes
Static Text	`summary`	String	The concise, top-level title of the task. Dense with jargon, but static after creation.
	`description`	String	The verbose original bug report or feature specification. Often contains code snippets, stack traces, and acceptance criteria.
Dynamic Text	`comment.body`	String	Actual conversational text. Follows a heavy-tailed length distribution. Essential for tracking emerging complexity.
	`comment.author`	Auxiliary	Used only during preprocessing to preserve within-task speaker turns; not passed to the model as a learnable identity feature.
	`comment.created`	Timestamp	The precise ISO-8601 UTC timestamp of the comment, used to calculate the Δt decay intervals.
Metadata	`priority`	Categorical	Ordered nominal scale (e.g., Blocker, Critical, Major, Minor, Trivial). Often subject to inflation ("everything is a blocker").
	`issuetype`	Categorical	Differentiates between 'Bug', 'New Feature', 'Improvement', and 'Sub-task'. Dictates expected lifecycle flow.
	`customfield_10004`	Numeric	'Story Points.' A quasi-Fibonacci scale (1, 2, 3, 5, 8...) estimating cognitive effort. Highly subjective and team-specific.
	`created`	Timestamp	The anchor point denoting the inception of the task ($T = 0$).

Methodological Note on Developer Identity and Leakage: Raw identity fields such as `assignee.id`, explicit developer-name fields, and persistent author identifiers were excluded from the learnable feature set. The raw `comment.author` field was retained only during preprocessing to preserve speaker boundaries within a task's comment history; it was not embedded or otherwise supplied as a developer-identity variable to the model. That distinction matters. The model may use comment order and timing within a task, but it does not learn per-developer productivity profiles. Therefore, excluding persistent identifying characteristics can reduce the risk of leakage and ethical issues related to workplace analysis.

4.1.2 Prediction Semantics and Operational Interpretation

The ultimate learning goal is to optimize a highly nonlinear mapping function f_θ such that $\hat{y}_i = f_\theta(X_i)$, where $\hat{y}_i$ represents the predicted probability that task i will be completed within the horizon $H = 7$ days. Define the ground truth result accordingly: $y_i = 1$ if the task is completed within H days of the prediction point, and $y_i = 0$ otherwise. For managerial interpretation, the complementary delay-risk score can be defined as $r_i = 1 - \hat{y}_i$.

The distinction between statistical objectives and management interpretation needs to remain clear. The model is trained to estimate completion within a range; it is not trained on the independently annotated concept of "risk". Completion status within 7 days can be reconstructed directly from the issue tracker records. Labels such as potential task difficulty or upgrade requirements are more incidental and rely more on local process conventions.

This option has two benefits. Replicability comes first: another researcher can derive the target from the same historical record. With that comes flexibility. A lower probability of completion may be operationally interpreted as a higher risk of delay, but the supervisory goal itself remains a stable binary event rather than an ever-changing organizational structure.

The same distinction also illustrates thresholds. In the reported experiments, traditional classification metrics were calculated with fixed decision thresholds when the metric definition required a fixed decision threshold. The operating purposes are different. Organizations may prefer lower intervention thresholds when false assurances are costly, and higher intervention thresholds when review capabilities are scarce. This approach therefore treats predicted probabilities as the primary statistical object and binary cutoffs as downstream policy choices.

This representation also helps prevent label drift between chapters. The same output can be discussed in two complementary ways: the modeled short-term completion probability and the derived delay-risk indicator used for managerial interpretation. The fundamental definition of the target variable must remain stable. For that reason, the methods chapter

defines the target once at the data level and treats risk-oriented language as a transformation of that probability rather than as a separate prediction task.

4.2 Feature Extraction and Embedding Pipeline

Before fusion can occur, the heterogeneous inputs must be mapped into continuous representations.

4.2.1 Dense Semantic Embedding (SBERT)

Natural-language features (titles, descriptions, comments) are processed with a pretrained Sentence-BERT model, specifically `all-mpnet-base-v2` [49]. Its tokenizer maps raw text into contextualized subword units before embedding.

Token limits are used to control computational cost without discarding the basic narrative structure of the text. Titles and descriptions are truncated at 256 tokens, while individual comments are limited to 128 tokens. For the temporal sequence, only the $T_c = 20$ most recent comments are processed; older comments are omitted under the assumption that short-horizon prediction depends most strongly on recent context. The resulting embeddings lie in a $D = 768$ dimensional space and feed the downstream attention blocks. Tasks with no comments are padded with a zero vector of matching dimension so that batch structure remains uniform.

4.2.2 Sequence Windowing and Information Budget

The token and sequence limits above are not incidental implementation details; they are part of the modeling assumption. Short-horizon task prediction is most plausibly influenced by the current technical state of the task, recent developer discussion, and the immediately relevant historical context. It is less plausible that every early comment in a long ticket lifecycle should retain equal influence, especially when many of those comments concern setup or scoping issues that were resolved earlier.

Therefore, the comment channel is treated as a restricted recent-history channel rather

than an unlimited document-ingestion stream. The model is limited to the last 20 comments, and each comment is truncated to 128 tokens to create a clear information budget. Within this budget, the architecture can still distribute attention unevenly: recent routine updates may receive little emphasis, while older but semantically salient blocker comments can remain influential through time-aware mechanisms. The window therefore does not impose a purely recency-driven heuristic, but it does impose practical constraints on the historical context of each prediction instance.

This design also improves comparability between tasks. Issue-tracker histories vary widely in length. Some tasks contain no comments, many contain only a few comments, and smaller subsets contain longer debugging threads or pasted stack traces. Without a limited window, the computational burden of training would be disproportionately driven by a few unusually long tasks. The selected window scheme concentrates the dynamic channel on the most likely relevant dialogue fragments within the seven-day horizon, while still allowing tractable batch construction.

The consequences of this choice cannot be ignored. A bounded recent-history model may miss early but still relevant information that is never repeated later in the thread. That limitation is one reason the preprocessing and failure-analysis chapters pay close attention to the interaction among truncation, normalization, and rare technical cues. Put differently, the information budget is a deliberate tradeoff between representational completeness and experimental feasibility, not a claim that earlier task history is universally irrelevant.

4.2.3 Categorical and Numeric Normalization (Mathematical Feature Engineering)

Modern neural architectures require that all input modalities be mapped into a continuous vector space. While text is embedded using SBERT, structured metadata (e.g. story points, priorities) requires separate transformation before fusion.

Numeric features such as customfield_10004 (story point) vary greatly from project to project. The raw value of 5 does not necessarily represent a comparable effort from two

different teams. Injecting these values directly into the network will also produce poorly scaled activations during optimization.

Z-Score Normalization of Continuous Variables

For that reason, **Z-Score Normalization** is applied to continuous variables. Let $x_{i,j}$ be the raw numeric value of feature j for task i. The normalized scalar $z_{i,j}$ is calculated as:

$$z_{i,j} = \frac{x_{i,j} - \mu_j}{\sigma_j} \tag{4.1}$$

The population mean (μ_j) and standard deviation (σ_j) are computed from the training partition only:

$$\mu_j = \frac{1}{N_{train}} \sum_{k=1}^{N_{train}} x_{k,j} \tag{4.2}$$

$$\sigma_j = \sqrt{\frac{1}{N_{train} - 1} \sum_{k=1}^{N_{train}} (x_{k,j} - \mu_j)^2} \tag{4.3}$$

Entity Embeddings for Categorical Metadata

Categorical variables (e.g., `Priority`, `IssueType`) are often processed using one-hot encoding, which treats all categories as equally distant from one another.

To allow the model to learn graded relationships among categories, the architecture uses **trainable categorical embeddings**. This choice is consistent with recent software-prediction research that emphasizes learned feature representation and relational encoding over purely fixed manual encodings [42, 53]. Let the categorical variable $Cat_{priority}$ possess K unique classes. Each class $k \in \{1 \dots K\}$ is mapped to a trainable dense vector $e_k \in \mathbb{R}^D$ within an embedding matrix $E \in \mathbb{R}^{K \times D}$.

$$x_i^{priority} = E \times \text{OneHot}(Cat_{priority}) \tag{4.4}$$

Because the matrix E is updated during training, the model can place categories such as *Blocker* and *Critical* closer together than categories with less similar meanings.

Business-Hour Temporal Decay Parsing

Timestamps are converted into consecutive relative intervals. For dynamic features, elapsed time is measured between comment generation and the prediction point. In this study, elapsed time is operationalized in active working time rather than raw wall-clock time. This choice was made after an exploratory examination of the ASF data, in which many tickets showed clustering of communication during weekdays, separated by regular gaps on weekends; equating weekend gaps with midweek inactivity may exaggerate evidence of delays for some tasks. Therefore, Saturdays and Sundays were excluded during construction. This should be understood as a workflow-specific modeling decision rather than a general time standard. A wall-clock ablation remains a useful extension.

4.2.4 Representation of Silent and Sparse Tasks

The issue tracker dataset contains many sparse cases. Some tasks have no annotations before the prediction point; others have only brief updates or long periods of inactivity. These cases cannot simply be dismissed. Doing so biases the dataset toward widely discussed work items and systematically removes tasks for which silent or delayed responses may themselves be informative.

The architecture therefore represents empty or very short histories explicitly rather than discarding them. When comments are absent, the dynamic-text channel is padded with a zero vector so the task can still be processed jointly with static text and metadata. In this way, the lack of dialogue evidence is still part of the observation, not the reason to exclude the task from training. Then, the model combines the missing dynamic channels with the remaining signals, including priority, problem type, story points, title and description.

This approach is conservative in two senses. First, it avoids synthesizing dialogue through interpolation. Second, it preserves the distinction between “no observed evidence

of dynamics" and "observed evidence of dynamics with low semantic urgency". At the same time, it has limitations. A zero-padded comment history cannot fully convey the difference between a truly quiet but healthy task and a neglected task whose lack of updates is alarming in itself. This limitation will be discussed later with a failure case where silence appears to be informative but is only indirectly represented by the current model.

Sparse histories also interact with the temporal formulation. When only a few comments exist, the model's decisions are driven more heavily by static text and structured metadata, and the time-aware mechanism has fewer opportunities to differentiate among historical states. The architecture is still defined in those cases, but the effective source of predictive evidence changes. This is one reason the dissertation evaluates multimodal ablations and failure cases rather than presenting only aggregate performance summaries.

4.3 The Context-Aware Multi-Channel Architecture

The architecture aims to avoid integrating all patterns at the earliest stage. In software-task data, static text, dynamic comment history, and structured metadata play different statistical roles. Flattening them immediately into a single input tensor risks obscuring that modality-specific structure.

The model therefore adopts a **multi-channel topology** (Figure 4.1). Static text, structured metadata, and dynamic text are processed separately before cross-feature interaction is introduced.

4.3.1 Modeling Sequence Dynamics via GRU

As outlined in Chapter 3, dynamic textual sequences, that is, the comment threads, are fed sequentially into a Gated Recurrent Unit (GRU). The GRU encodes conversational flow and dependency across successive developer responses.

$$h^c_{i,1}, \ldots, h^c_{i,T_c} = \mathrm{GRU}_c(\mathrm{emb}(c_{i,1}), \ldots, \mathrm{emb}(c_{i,T_c})) \tag{4.5}$$

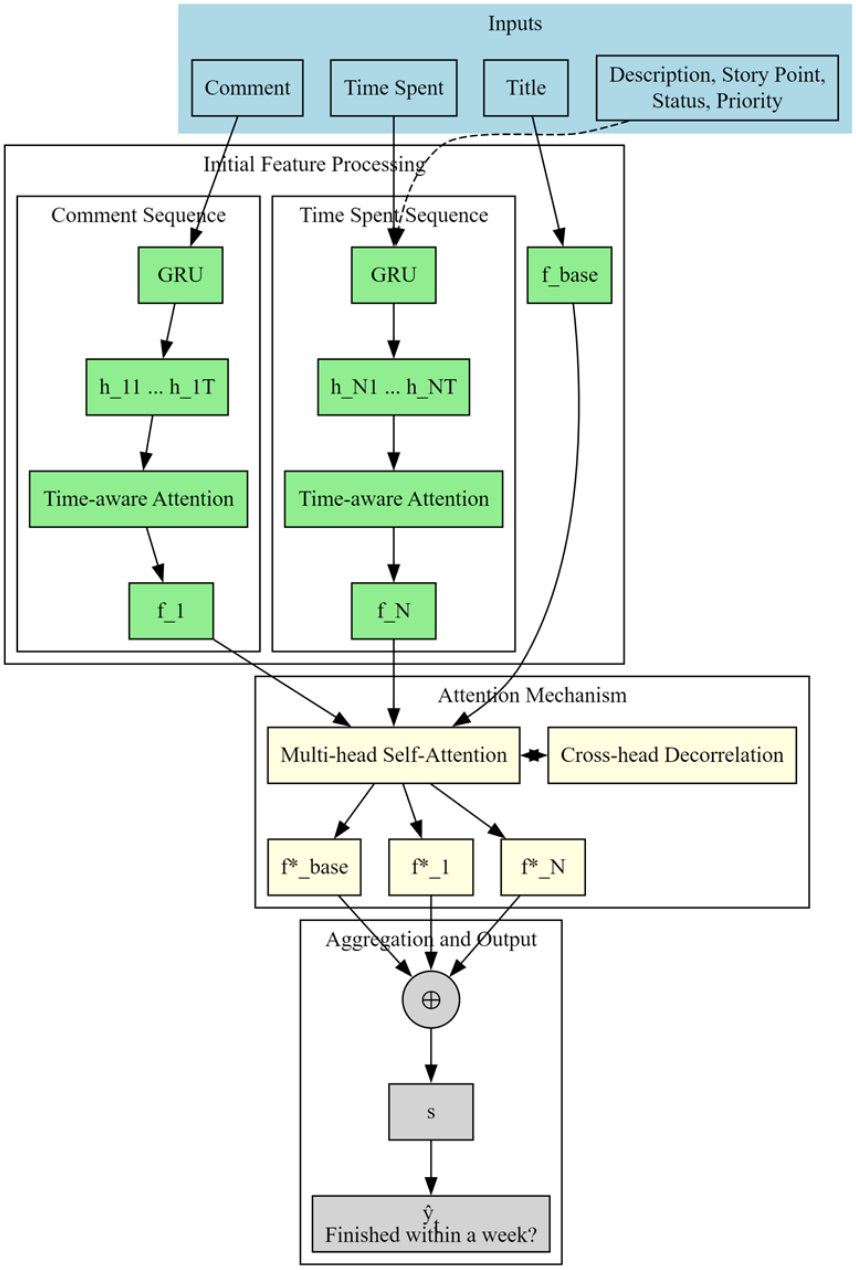

Figure 4.1: Illustration of the proposed context-aware architecture for predicting short-term task completion. The model uses different inputs in different channels, applies GRU + time-aware attention to dynamic sequences, and then uses a hierarchical cross-feature self-attention module designed with decorrelation regularization.

The result is a time series of hidden states. Each vector $h_{i,t}^{c}$ represents the contextual meaning of the t-th comment together with a compressed summary of the preceding thread.

4.3.2 Feature-Specific Time-Aware Attention

The architecture also needs to account for irregular gaps within the sequence. Standard attention mechanisms rely mainly on index order and do not encode elapsed time between comments directly.

To address that limitation, a feature-specific time-aware attention mechanism is introduced. Given the GRU sequence outputs $h_{i,t}^{c}$, the attention weights $\alpha_{i,t}^{c}$ decay as a function of the continuous time gap Δt from the most recent comment, while semantically salient comments are still allowed to retain influence.

The attention weights are therefore calculated through an information-driven scaling mechanism:

$$q_{i,T_c}^{c} = W_c^{q} \cdot h_{i,T_c}^{c} \quad \text{(Query: contextual representation of the current state)} \tag{4.6}$$

$$k_{i,t}^{c} = W_c^{k} \cdot h_{i,t}^{c} \quad \text{(Key: representation of the historical state)} \tag{4.7}$$

The crude energy score (ζ) combines the continuous delta time ($\Delta t_{i,t}^{c}$) controlled by a semantic gate:

$$\zeta_{i,t}^{c} = \tanh\left(\frac{q_{i,T_c}^{c} \cdot k_{i,t}^{c}}{\beta_c \cdot \log(e + (1 - \sigma(q_{i,T_c}^{c} \cdot k_{i,t}^{c})) \cdot \Delta t_{i,t}^{c})}\right) \tag{4.8}$$

This equation has two properties that matter for the present task:

- The dot product $q \cdot k$ represents the raw semantic resonance between the current state (T_c) and the past comment (t).
- The denominator introduces a **logarithmic temporal decay penalty**. The logarithmic form is used instead of a linear penalty because software communication is heavy-tailed; the contextual difference between a 1-hour and 24-hour gap is usually much larger than the difference between a 300-hour and 324-hour gap.

- This term acts as a content-aware gate. When semantic resonance is high, for example, because both the query and the key emphasize blocking-related language, the sigmoid output approaches 1. The time penalty then weakens and older but still relevant comments may continue to play a role.

These raw energy values are then passed through a softmax to produce the final attention distribution, and the weighted sequence summary becomes the dynamic vector f_i^c:

$$\alpha_{i,1}^c, \ldots, \alpha_{i,T_c}^c = \text{Softmax}(\zeta_{i,1}^c, \ldots, \zeta_{i,T_c}^c) \tag{4.9}$$

$$f_i^c = \sum_{t=1}^{T_c} \alpha_{i,t}^c \cdot h_{i,t}^c \tag{4.10}$$

4.3.3 Interpretive Role of the Time-Aware Summary

Summary vectors should be viewed as task state representations rather than textual compression of transcripts. Its job is to merge the comment history into a vector that emphasizes comments that are consistent with the current state and still temporally relevant at the prediction point. The architecture is not an attempt to recreate a conversation in miniature. It's estimating which parts of the conversation are still predictive of short-term outcomes.

This distinction is important because attention weights can easily be overinterpreted. A high weight does not mean that the comment uniquely led to the prediction, and a low weight does not mean that the comment lacks project-management importance. Claims are narrower. In the learned representation, some comments contribute more than others to the summary vector used by downstream classifiers. Time-aware summaries are therefore treated here as engineering statistical representations with partial qualitative interpretability rather than as transparent causal decompositions of team behavior.

To formalize this mathematical sequence into an engineered software module, Algorithm 1 delineates the explicit forward propagation layer constructed within the deep learning framework.

Algorithm 1 Forward Pass: Feature-Specific Time-Aware Attention

Require: sequence of GRU hidden states $H^c \in \mathbb{R}^{T_c \times d}$, time intervals $\Delta T \in \mathbb{R}^{T_c}$, trainable matrices W^q, W^k, scaling scalar β_c
Ensure: Aggregated dynamic representation vector $f^c \in \mathbb{R}^d$

$q_{T_c} \leftarrow H^c_{T_c} \cdot W^q$ ▷ Derive current state context query
for $t = 1$ **to** T_c **do**
  $k_t \leftarrow H^c_t \cdot W^k$ ▷ Map historical state to Key space
  $score_{sem} \leftarrow q_{T_c} \cdot k_t$ ▷ Compute dot-product semantic resonance
  $gate \leftarrow \text{Sigmoid}(score_{sem})$ ▷ Activate the content-aware gate $\in (0,1)$
  $penalty \leftarrow \log(e + (1 - gate) \cdot \Delta T_t)$ ▷ Apply gated logarithmic temporal decay
  $\zeta_t \leftarrow \tanh\left(\frac{score_{sem}}{\beta_c \cdot penalty}\right)$ ▷ Compute bounded raw energy
end for
$\alpha \leftarrow \text{Softmax}(\zeta_1, \ldots, \zeta_{T_c})$ ▷ Normalize to an attention distribution
$f^c \leftarrow \sum_{t=1}^{T_c} \alpha_t \cdot H^c_t$ ▷ Compute weighted sequence summary
return f^c

4.3.4 Theoretical Bounds of the Logarithmic Penalty

The mathematical formulation of the time-decay penalty, $\text{penalty}(\Delta t) = \log(e + \beta \Delta t)$, was selected after considering alternative decay functions with different growth rates.

Consider the sequence of past states $H_T = [h_1, h_2, \ldots, h_T]$ and the temporal gap vector $\Delta T = [\Delta t_1, \Delta t_2, \ldots, \Delta t_T]$. The objective of the attention mechanism is to compute a weighting score α_t that degrades as $\Delta t \to \infty$ without collapsing older but still relevant signals too quickly. Three decay distributions were considered:

1. Linear Decay: $\text{penalty}_{lin}(\Delta t) = 1 + \lambda \Delta t$

Linear decay assumes a constant loss of influence per hour. In this setting that is often too rigid, because the difference between very recent comments can matter more than equal-sized differences much later in the task history.

2. Exponential Decay: $\text{penalty}_{exp}(\Delta t) = \exp(\lambda \Delta t)$

Exponential decay grows too quickly for this application. Because $\exp(x)$ increases rapidly, the denominator in Equation 4.8 can become extremely large within a short horizon. For example, if $\lambda = 0.1$ and $\Delta t = 168$ hours (1 week), the penalty is $e^{16.8} \approx 19.7 \times 10^6$. In practice, such growth would suppress the influence of older comments too aggressively.

3. Logarithmic Decay (Proposed): $\text{penalty}_{log}(\Delta t) = \log(e + \lambda \Delta t)$

For the present task-tracking setting, the logarithmic formulation provides a useful compromise:

1. **Identity Bound at $t = 0$:** When $\Delta t = 0$, corresponding to an immediate reply, $\log(e + 0) = \log(e) = 1$. The penalty is neutral and the semantic dot product $q \cdot k$ governs the attention weight directly.

2. **Large Starting Penalty:** For small values (e.g., hours), the logarithmic curve still drops off quickly enough to separate recent comments from older ones.

3. **Asymptotic Saturation:** As $\Delta t \to \infty$, for example over several months, the curve flattens. The penalty difference between 3,000 and 3,100 hours is therefore small (Δpenalty ≈ 0.03).

This flat tail also leaves room for the content-aware gate to retain older comments that remain semantically relevant. Elapsed time remains a factor, but the logarithmic penalty prevents older, semantically relevant comments from being prematurely and heavily discounted.

4.3.5 Hierarchical Cross-Feature Attention for Context Fusion

After multi-channel processing, the isolated vectors must be synthesized: the dynamic summary f_i^c, along with the static embeddings for title, priority, and story points. The correlation structure in Figure 4.2 suggests that these modalities are not independent.

To model those dependencies while preserving the distinction between feature groups, the architecture uses a two-stage *Hierarchical Cross-Feature Attention* block.

The representation vectors are partitioned into G logical modality groups: (1) Dynamic Text, (2) Static Text, and (3) Non-Textual Numeric/Categorical.

Stage 1: Intra-Group Attention. Within each isolated group g, localized multi-head self-attention models internal dependencies (e.g., how the title is essentially consistent with

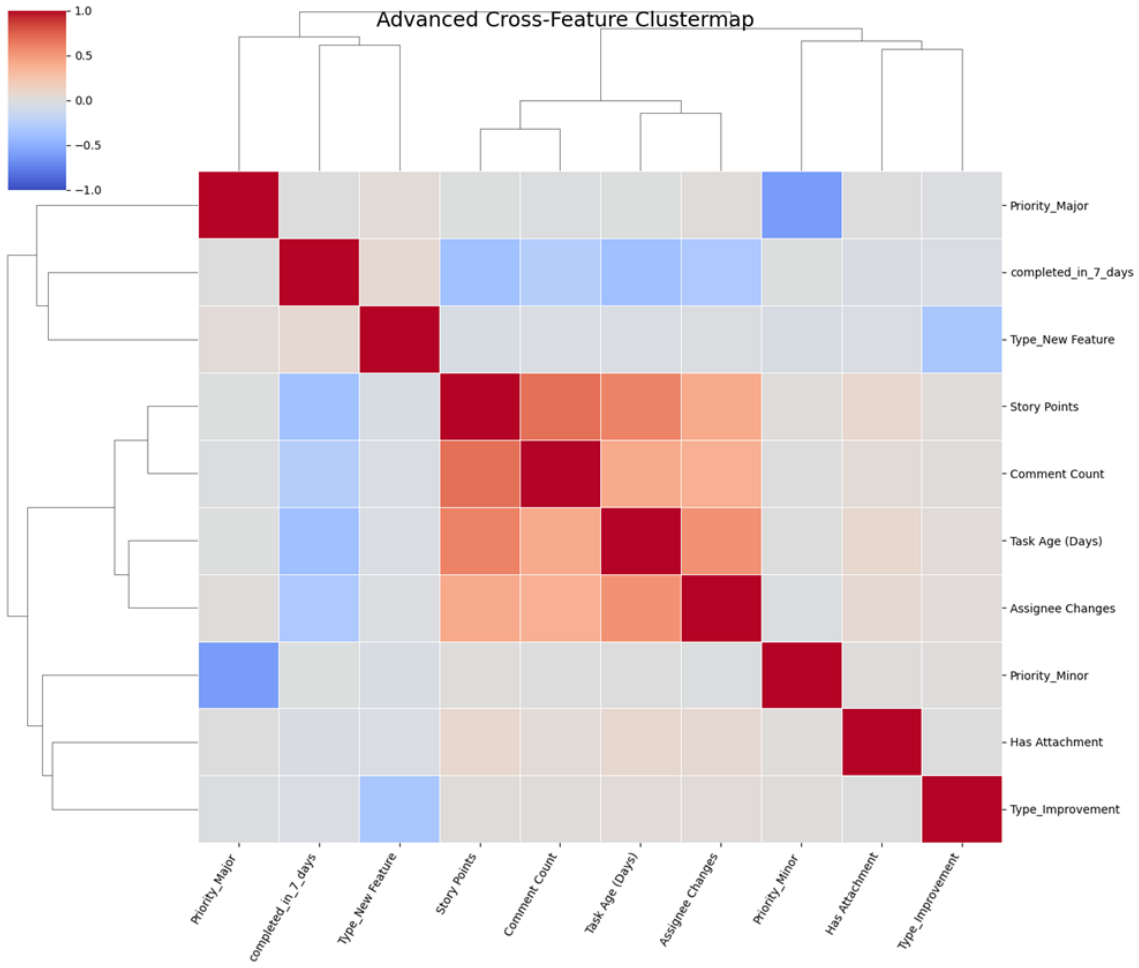

Figure 4.2: Cluster plots of correlation matrices across heterogeneous task features emphasize the need for structured rather than flat modal fusion.

the description). Let $F_i^{(g)}$ be the matrix of features for group g.

$$U_g = \text{LayerNorm}\left(F_i^{(g)} + \text{MultiHeadAttention}(F_i^{(g)})\right) \tag{4.11}$$

Standard residual connections and layer normalization help stabilize optimization during this fusion stage.

Stage 2: Inter-Group Attention. The refined, locally-aware group matrices (U_g) are then passed into a global pooling layer generating high-level summary vectors $\bar{u}_g$. These vectors are concatenated iteratively into a master fusion matrix $\tilde{F}_i = [\bar{u}_1; \bar{u}_2; \ldots; \bar{u}_G]$. A secondary multi-head self-attention layer then models cross-modal relationships (e.g., how urgency expressed in the dynamic text may interact with the priority category):

$$\tilde{U}_i = \text{LayerNorm}\left(\tilde{F}_i + \text{MultiHeadAttention}(\tilde{F}_i)\right) \tag{4.12}$$

4.3.6 Architecture Regularization: Cross-Head Decorrelation

Multi-head attention can exhibit head redundancy, where several heads learn highly similar mappings. To encourage more diverse interaction patterns across heads, a decorrelation constraint is added to the loss.

Let A represent the matrix of mapping outputs from the M attention heads, and C represent their correlation coefficient matrix. Redundancy is penalized by minimizing the off-diagonal elements of C relative to the identity matrix I:

$$L_{\text{decorr}}(A) = \|C - I\|_F^2 = \sum_{i \neq j} (C_{ij})^2 \tag{4.13}$$

The total hierarchical decorrelation loss applies this penalty simultaneously to both stages of the hierarchy:

$$L_{\text{decorrelation}}^{\text{hier}} = \sum_{g=1}^{G} L_{\text{decorr}}(A_g) + L_{\text{decorr}}(\tilde{A}) \tag{4.14}$$

The purpose of this penalty is modest but important. It does not guarantee that each

head learns a human-interpretable role, and it does not transform attention heads into independent explanatory factors. It acts instead as a structural regularizer against a failure mode in which multiple heads collapse toward nearly identical interaction patterns. In heterogeneous software-task data, that collapse would weaken the rationale for using multiple heads at all.

Therefore, the expected benefit of the decorrelation penalty is representational diversity rather than strict interpretability. If one head emphasizes interactions involving dynamic technical comments while another places more weight on structured urgency or effort signals, the fusion block may capture a wider range of cross-modal relations than would be available from a redundant head set. Comparable uses of structured multi-head attention over heterogeneous feature sources have also appeared in non-software prediction settings [50]. The subsequent ablation test evaluates whether this regularization produces measurable improvements, but the method does not claim that the effect has a directly causal interpretation.

4.3.7 Aggregation and Final Classification Output

In the final stage, the fused representation is mapped to a scalar prediction. The refined vectors from both local and global stages are flattened and aggregated via a Feed-Forward Network:

$$Z_i = \text{Concat}(\bar{u}_1, \ldots, \bar{u}_G, \text{Flatten}(\tilde{U}_i)) \tag{4.15}$$

$$s_i = \text{DenseNetwork}(Z_i) \tag{4.16}$$

The final predictive probability $\hat{y}_i$ is then bounded in $[0,1]$ by the logistic sigmoid activation:

$$\hat{y}_i = \sigma(W^{\text{fin}} \cdot s_i + b^{\text{fin}}) \tag{4.17}$$

4.4 Formal Computational Complexity and Gradient Analysis

This section characterizes the computational complexity of the architecture and examines whether the time-aware term introduces undesirable gradient behavior.

4.4.1 Computational Complexity (Big-$\mathcal{O}$) Bounds

The main computational cost of the proposed architecture lies within the dynamic text channel, specifically the interaction between the recurrent (GRU) base and the time-aware attention head. Let T_c be the maximum sequence length (number of comments), d be the embedding dimension (768 for SBERT), and d_k be the Key/Query projection space.

1. **Standard GRU Processing:** A single step of the GRU block entails calculating update and reset gates, requiring matrix multiplications of dimension $d \times d$. Across the sequence T_c, the time complexity is strictly $\mathcal{O}(T_c \cdot d^2)$.

2. **Time-Aware Attention Forward Pass:** Unlike standard Transformer self-attention, which computes a full $T_c \times T_c$ similarity matrix with complexity $\mathcal{O}(T_c^2 \cdot d)$, the proposed attention compares the final contextual state with each prior state. As shown in Algorithm 1, the loop iterates T_c times and performs d_k-dimensional dot products, giving complexity $\mathcal{O}(T_c \cdot d_k)$.

3. **Hierarchical Fusion:** The fusion stages operate exclusively on the highly compressed, fixed-dimension representation vectors from each of the G modalities (where $G = 3$). Because $G \ll T_c$, the $G \times G$ self-attention operation is computationally trivial: $\mathcal{O}(G^2 \cdot d) \approx \mathcal{O}(d)$.

The asymptotic upper bound of the forward pass per task is therefore defined by the GRU: $\mathcal{O}(T_c \cdot d^2)$. This is lighter than full-sequence processing with very large language models, although those systems are not otherwise directly comparable because their input assumptions and task framing differ.

4.4.2 Gradient Flow and Decay Stability Analysis

A common concern in deep sequence models is the vanishing-gradient problem. The analysis below checks whether the temporal penalty in Equation 4.8 introduces an additional source of gradient collapse. Let $P = \log(e + (1 - \text{gate}) \cdot \Delta t)$ represent the decay penalty constraint, where gate $\in (0, 1)$. The partial derivative of the raw energy ζ with respect to the pre-decay semantic score $S = q \cdot k$ dictates the gradient flow through the attention head:

$$\frac{\partial \zeta}{\partial S} = \frac{\partial}{\partial S}\left[\tanh\left(\frac{S}{\beta \cdot P}\right)\right] \tag{4.18}$$

Using the chain rule, recognizing that P is also a function of S (since the gate relies on S):

$$\frac{\partial \zeta}{\partial S} = \left(1 - \tanh^2\left(\frac{S}{\beta P}\right)\right) \cdot \left[\frac{1 \cdot (\beta P) - S \cdot \left(\beta \frac{\partial P}{\partial S}\right)}{(\beta P)^2}\right] \tag{4.19}$$

Because Δt refers to elapsed time, it is nonnegative ($\Delta t \geq 0$). Therefore, the minimum possible value of the argument inside the logarithm is e.

$$\min(P) = \log(e + 0) = \log(e) = 1 \tag{4.20}$$

Because the penalty modifier P is bounded below by 1, the denominator $(\beta P)^2$ remains well behaved. Because P increases only logarithmically as Δt grows, the temporal term scales sublinearly rather than explosively. Under this formulation, the time-aware component does not appear to drive the gradient toward numerical instability on its own.

4.5 Negative Results and Architectural Evolution

After several simpler alternatives did not perform well, the final architecture emerged. Reporting these negative results helps to explain why the final design adopts the current form.

4.5.1 Failure 1: Naive Early Fusion

The initial prototype adopted an early-fusion design. The original SBERT text vector, one-hot priority matrix, and normalized story points are passed to the standard multilayer perceptron (MLP).

The Result: The model overfit substantially, reaching a training F1-score of 0.94 but only a validation F1-score of 0.58. That pattern suggests that the flat MLP learned project-specific lexical regularities more readily than stable cross-modal structure, which in turn motivated the later late-fusion design.

4.5.2 Failure 2: Time-Aware Multi-Head Attention without Decorrelation

In designing the Hierarchical Cross-Feature Attention block, the model was first trained with vanilla Multi-Head self-attention without the $L^{\text{hier}}_{\text{decorrelation}}$ penalty term (Equation 4.14).

The Result: Attention-head redundancy was observed. In the 4-head configuration, 3 heads converged toward very similar weight distributions and concentrated mainly on the interaction between story points and the GRU output. Adding the decorrelation term produced more differentiated head behavior across textual, categorical, and quantitative signals.

4.6 Optimization Objective and Training Loop

The network is optimized end-to-end with a combined loss function.

$$L_{total} = L_{\text{cross-entropy}}(y_i, \hat{y}_i) + \lambda \cdot L^{\text{hier}}_{\text{decorrelation}} \tag{4.21}$$

The first term is the standard binary cross-entropy prediction loss. The second term is the decorrelation penalty, weighted by the tunable hyperparameter λ, which encourages diversity across attention heads.

Algorithm 2 summarizes the corresponding training procedure used to fit the PyTorch model on the imbalanced dataset.

Algorithm 2 End-to-End Task Prediction Training Loop

Require: Training dataset $\mathscr{D}_{train}$, batch size B, max epochs E, initial learning rate $\eta = 1e-3$

Ensure: Tuned model weights Θ

Initialize model weights Θ orthogonally
Initialize Adam Optimizer with η, weight_decay $= 0.01$
for epoch $\leftarrow 1$ **to** E **do**
    Shuffle dataset $\mathscr{D}_{train}$ randomly
    for each batch $b = \{X_1, \ldots, X_B\}$ **in** $\mathscr{D}_{train}$ **do**
        $f_{text_dyn}, f_{text_stat}, f_{meta} \leftarrow \text{ChannelForward}(X_b)$ ▷ Parallel processing
        $\tilde{U}_b \leftarrow \text{HierarchicalFusion}(f_{text_dyn}, f_{text_stat}, f_{meta})$ ▷ Synthesis
        $\hat{Y} \leftarrow \text{Sigmoid}(\text{DenseNetwork}(\tilde{U}_b))$ ▷ Generate predictions
        $L_{BCE} \leftarrow -\frac{1}{B}\sum[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$ ▷ Compute batch loss
        $L_{decorr} \leftarrow \text{ComputeDecorrelationPenalty}(\Theta_{heads})$
        $L_{total} \leftarrow L_{BCE} + (\lambda \cdot L_{decorr})$ ▷ Equations 4.13 and 4.21
        Compute Gradients $\nabla_{\Theta} L_{total}$ via backpropagation
        Optimizer.step()
        Optimizer.zero_grad()
    end for
    if epoch $\pmod 5 == 0$ **then**
        $\eta \leftarrow \eta \times 0.1$ ▷ Step learning rate decay schedule
    end if
end for
return Θ

Chapter 5 - Experimental Setup

This chapter describes the empirical framework used to evaluate the proposed model. It defines the dataset, clarifies how prediction instances were constructed, describes the baseline models, and specifies the evaluation procedures used for held-out, chronological, robustness, and statistical-comparison analyses.

5.1 Dataset Acquisition and Curation

The empirical dataset was extracted from the **Apache Software Foundation's (ASF) public Jira issue tracking system**. ASF projects were selected because they provide long-lived, publicly inspectable issue histories across multiple software domains [58]. To align the dataset with the short-horizon prediction task studied here, the following inclusion criteria were applied:

1. **Agile Footprint:** Projects had to show explicit agile workflow evidence in Jira metadata, such as sprint fields, iteration labels, board-linked status categories, or recurring planning/review states rather than only a generic issue list.

2. **Temporal Depth and Volume:** Projects had to span at least 24 months of active development and include at least 500 completed tasks.

3. **Usable-Data Filter:** Core fields listed in the data dictionary had to be available at usable rates. Two projects were excluded at this stage because, after field-availability screening, they retained only 2 and 14 usable tasks respectively.

The final dataset contains 37 projects drawn from several architectural contexts:

- **Large-Scale Distributed Systems:** e.g., Hadoop, Cassandra.
- **Web Frameworks and Enterprise Middleware:** e.g. Struts, Camel.
- **Build Automation and Utility Libraries:** e.g., Maven, Commons.

This diversity is very useful because it exposes the model to different development speeds, comment density and issue tracker usage patterns.

The selection procedure was designed to be operational rather than impressionistic. Projects first had to satisfy the explicit metadata, duration, and volume rules listed above. They were then retained only if those rules yielded a nontrivial number of usable task records after field-availability filtering. This approach does not remove all judgment from corpus construction, but it does narrow the role of judgment to edge cases rather than allowing ad hoc project inclusion.

5.2 Prediction Instances and Snapshot Construction

The model is trained on *task snapshots* rather than on one record per task. A prediction instance corresponds to the state of a task at a historical *prediction point*, using only the information available up to that point. Prediction points were created from recorded task states after task creation and subsequent logged updates in the Jira history. For each prediction point, the target label is defined relative to the next seven days: $y_i = 1$ if the task closes within $H = 7$ days after that point, and $y_i = 0$ otherwise.

Operationally, each snapshot contains the task's then-current static fields, the comment history available up to the prediction point, and the derived temporal intervals between those comments. No future comments, future status transitions, future field edits, or post hoc labels are included in the feature representation at that point in time. In particular, the close date is used only to determine whether the task completed within the subsequent seven-day horizon; it is not exposed to the model as an input feature. If a task remains open, later states of the same task can appear as additional snapshots, each with more accumulated evidence than the earlier one.

Since a single task may be observed at multiple points in its lifecycle, the dataset expands from 33,624 unique tasks to 131,069 predicted snapshots. Therefore, multiple snapshots originate from the same underlying task. This is important for leakage control: during

all project retention, validation, and project-level cross-validation processes described below, all snapshots for a given task are retained within the same project partition. Therefore, the task does not provide one snapshot for training and another snapshot for testing in these evaluations. The grouping principle is strict: the unit of the partition is actually the task follow-up embedded in the project split, not a single snapshot line.

Project-level separation does not, by itself, eliminate forward-looking temporal leakage. Models can still benefit from vocabulary and workflow patterns learned from the same ecosystem in the later stage. For that reason, a separate chronological evaluation is included as a primary deployment-oriented check rather than being treated only as a supplementary analysis.

This distinction between *task-level leakage control* and *deployment-level temporal validity* is important for interpreting the experiments. The project-held-out and project-level cross-validation procedures test generalization to unseen projects within the same corpus under controlled partitioning. The chronological split addresses a different question: whether a model trained on earlier periods remains effective when evaluated on materially later periods, which is closer to the intended use case of forward-looking decision support. Where partition-specific fitting was required, such as model selection and normalization statistics for structured features, those quantities were estimated on the training side of each split only.

5.3 Feature Engineering and Distribution Analysis

The joint corpus contains 33,624 distinct tasks and 131,069 prediction snapshots from all 37 projects. Table 5.1 summarizes the final corpus.

5.3.1 Exploratory Data Analysis (EDA)

Exploratory analysis was conducted to characterize task duration, comment activity, and project heterogeneity before model training.

This preliminary characterization served two purposes. First, it checked whether the

Table 5.1: Descriptive Statistics of the Combined Dataset (37 Apache Projects)

Category	Metric	Value
Overall	Total Projects	37
	Total Unique Tasks	33,624
	Total Prediction Snapshots	131,069
Target Variable	Completed in 7 Days (1)	75,989 (58.0%)
	Not Completed in 7 Days (0)	55,080 (42.0%)
Task Type	Bug	51,541 (39.3%)
	New Feature	33,892 (25.9%)
	Improvement	26,913 (20.5%)
	Other	18,723 (14.3%)

prediction target was empirically meaningful at a seven-day horizon rather than being dominated by either trivially easy or extremely long-lived tasks. Second, it informed architectural choices such as the use of explicit temporal intervals, contextual text embeddings, and project-level evaluation procedures.

1. Task Lifecycle and Completion Times: Task completion times were highly right skewed with a median task completion time of 4.2 days across the 37 projects, while about 12% of the monitored tasks remained open beyond 60 days and 3% remained open for over one year before closure. This distribution motivates casting the task as binary short-horizon classification rather than long-range duration regression.

2. Comment Volume and Temporal Sparsity: The average task contains 4.1 comments, with substantial dispersion ($\sigma = 8.3$). About 18% of tasks contain no comments, while more complex tasks can generate comment threads exceeding 50 entries. Inter-comment gaps are also irregular: short bursts of discussion are often separated by longer periods of implementation silence. This observation motivated the explicit temporal component in the proposed time-aware attention module. As discussed in Chapter 7, the use of business-time intervals rather than raw wall-clock time should be interpreted as a modeling choice for this ASF setting rather than as a universal assumption.

5.3.2 Project Heterogeneity within the Apache Ecosystem

Projects such as `HADOOP` and `CASSANDRA` exhibit longer and denser issue discussions than projects such as `COMMONS` or `LOG4J`. In the present dataset, Hadoop tasks average 9.4 comments per ticket with a median comment length of 312 words, whereas Commons tasks average 2.1 comments per ticket with a median length of 45 words. Median time-to-completion also varies substantially across projects.

Text volume alone is therefore not a reliable proxy for short-term completion. The comparison also explains why evaluation across diverse projects is preferable to evaluation within a single repository style.

5.3.3 Handling Class Imbalance

The target variable is moderately imbalanced: 58.0% of snapshots are task completions within the 7-day horizon, while 42.0% are tasks that were not completed within this window.

Since this ratio is moderate rather than extreme, no synthetic oversampling is used. In multimodal sequence settings, oversampling methods like SMOTE may distort the underlying text representation. Therefore, the evaluation emphasizes F1-score, PR-AUC, and supplementary calibration analysis rather than just accuracy.

5.3.4 Multi-Collinearity Analysis via Variance Inflation Factor (VIF)

To assess that risk, a **Variance Inflation Factor (VIF)** analysis was performed on the static numeric/categorical matrix (X_i^S). VIF quantifies how much the variance of an estimated regression coefficient increases because of collinearity. Values above 5.0 are commonly treated as concerning, while values above 10.0 typically motivate variable removal.

The VIF test result was 1.82 for Priority and 2.05 for Story Points. Those values indicate that urgency and effort are not strongly collinear in this dataset, so both features were retained in the metadata branch.

5.3.5 The Natural Language Processing Pipeline

Issue-tracker text contains substantial noise, including stack traces, code snippets, commit hashes, links, and project-specific abbreviations. The preprocessing pipeline is designed to retain signals useful for task state while normalizing artifacts that would otherwise overwhelm the token budget of the sentence encoder.

Stage 1: Structural Stripping and Regex Cleansing

The first pipeline step makes use of aggressive Regular Expression (Regex) and structural parsing to remove non-narrative noise:

1. **Stack Trace Normalization:** Multi-line blocks matching standard Java/Python trace signatures were replaced with the sentinel token ‘[STACKTRACE]‘.

2. **Code Snippet Normalization:** Text enclosed within Jira ‘{code}‘ tags was replaced with ‘[CODE_SNIPPET]‘.

3. **Hyperlink and Hash Normalization:** HTTP/HTTPS URLs were replaced with ‘[URL]‘, and 40-character hexadecimal Git commit hashes were replaced with ‘[COMMIT_HASH]‘.

Stage 2: Domain-Specific Tokenization and OOV Handling

The residual text following the structural normalization step was then tokenized with SBERT’s native WordPiece tokenizer. Software engineering text nevertheless contains many out-of-vocabulary abbreviations and code-like identifiers.

1. **Acronym Expansion:** A domain-specific dictionary was used to expand common acronyms such as "NPE" and "OOM" before tokenization.

2. **Identifier Splitting:** `camelCase` and `snake_case` strings were segmented into component words before tokenization.

3. **Sequence Truncation Strategy:** For long comments, tail truncation was used so that later portions of the discussion remained available to the encoder.

The resulting normalized text is then passed to the frozen Sentence-BERT model to produce the 768-dimensional semantic vector E_i.

5.3.6 Characteristics of the Dynamic Text (Comments)

For example, a comment such as "I received ConcurrentModificationException in line 42 of TaskRunner. Blocked until we resolve the lock." illustrates the kind of task-state evidence that the dynamic-text channel is intended to preserve.

Two modeling choices in this pipeline should be read as design decisions for the evaluated workflow rather than as universal optima. The first is a bounded dynamic text channel: only the last 20 comments are kept, and each comment is truncated to 128 tokens. This setting reflects information-budget assumptions for the 7-day horizon rather than a claim that early history is always irrelevant. The second is the use of business-time intervals rather than raw wall-clock time, motivated by the observed workday communication rhythm in the ASF corpus. The reported results should therefore be interpreted with these choices in mind. Without additional ablations over wider cutoff windows or raw wall-clock timing, this dissertation does not claim that the same settings will perform best in different ecosystems or longer horizons.

5.4 Baseline Configurations

The proposed architecture is compared with eight baseline models covering traditional machine learning, standard neural sequence models, and pretrained Transformer encoders.

5.4.1 Traditional Machine Learning Baselines

These models provide reference points based primarily on structured relationships or sparse lexical features. Text inputs were vectorized with TF-IDF.

- **Logistic Regression:** A linear baseline with L2 regularization.
- **Random Forest:** A tree-ensemble baseline for structured and sparse features.
- **Extreme Gradient Boosting (XGBoost):** A strong tabular baseline included because of its widespread use in software-engineering prediction tasks.

5.4.2 Standard Deep Learning Baselines

These models use the same SBERT embeddings as the proposed model but simpler fusion structures.

- **Artificial Neural Network (ANN):** A flat-fusion multilayer perceptron over the concatenated feature vector.
- **GRU and LSTM:** Standard recurrent architectures that summarize the comment sequence and concatenate it with the static features.
- **Vanilla Transformer Encoder:** A standard attention baseline with fixed positional encodings [26].

5.4.3 Pretrained Transformer Encoder Benchmark

- **Fine-Tuned RoBERTa:** A pretrained RoBERTa-base contextual text classifier is included as a software-text prediction baseline, motivated by recent journal evidence that pretrained language representations and transformer-based encoders can be effective on bug- and task-related prediction problems [33, 23]. The model was fine-tuned for sequence classification using the title, description, and concatenation of the 5 most recent comments, truncated to 512 tokens. The ‘[CLS]‘ representation was concatenated with the metadata for final classification. This baseline tests whether a generic pretrained text encoder outperforms the proposed domain-specific multimodal architecture.

5.5 Evaluation Strategy

To assess generalization while keeping the leakage assumptions explicit, four complementary evaluation procedures were used.

1. **Primary Evaluation I (Held-Out Project Split):** The 37 projects were partitioned at the *project level*. Thirty projects formed the training pool and the remaining 7 projects formed the held-out test set. Because the split is project-level, no task from a test project, and no snapshot derived from that task, appeared during training.

2. **Primary Evaluation II (Strict Chronological Split):** Because deployment is forward-looking, a separate chronological evaluation was treated as a primary generalization check. Within the 30-project training pool, prediction points dated January 1, 2010 through December 31, 2015 were used for training, while prediction points dated January 1, 2018 through December 31, 2022 were reserved for evaluation. The intermediate period was left unused to create a temporal gap between train and test periods.

3. **Robustness Check (5-Fold Project-Level Cross-Validation):** To assess sensitivity to the specific fixed split, 5-fold project-level cross-validation was conducted across the full set of 37 projects.

4. **Statistical Comparison Procedure (10-Fold Project-Level CV on the Training Partition):** To support the Wilcoxon tests reported in Chapter 6, a separate 10-fold project-level cross-validation procedure was conducted on the 30-project training partition. This procedure generated the paired F1-score samples used for non-parametric pairwise comparison against each baseline. Because this resampling remains temporally mixed within the historical training corpus, it is interpreted as a statistical-comparison device rather than as forward-looking deployment evidence.

Hyperparameter tuning and early stopping were conducted on a project-level validation

Table 5.2: Evaluation Protocol Summary and Interpretation

Protocol	Data Partition	Question Answered	Interpretation Boundary
Held-out project split	30 ASF projects for training; 7 ASF projects held out for testing	Does the model generalize to unseen projects within the same ASF Jira ecosystem?	Primary evidence for project-level generalization inside the ASF dataset family.
Strict chronological split	Earlier prediction points from the 30-project pool for training; 2018–2022 prediction points for evaluation after a temporal gap	Does performance persist when deployment is forward-looking rather than time-mixed?	Primary temporal-validity check, but still within ASF Jira.
5-fold project-level cross-validation	All 37 projects rotated through project-level folds	Is the fixed 30/7 split unusually favorable or broadly stable across project partitions?	Robustness evidence, not a stronger deployment simulation than the chronological split.
10-fold training-partition procedure	30-project training partition only	Are paired fold-level F1 differences large enough to support Wilcoxon comparison with baselines?	Statistical-comparison device; not a hypothesis test on the 7-project held-out set.

split carved from the 30-project training pool. This ensured that no task, and no snapshot derived from that task, crossed the train-validation boundary during model selection.

These procedures answer related, but not identical, questions. The held-out project split tests generalization to unseen projects within the ASF ecosystem. The chronological split raises a stricter question: whether a model trained on earlier periods remains effective under later vocabulary and workflow conditions. The 5-fold project-level cross-validation procedure measures robustness to the fixed project split, while the separate 10-fold training-partition procedure exists mainly to support paired statistical comparison under controlled resampling. In other words, project splitting handles snapshot dependencies across tasks, whereas temporal splitting handles the separate problem that time-mixed training can leak future organizational patterns into evaluation.

This order has been carefully considered. When interpreting the results, priority is given to held-out project segmentation and chronological segmentation, because they are closer to the substantive generalization proposition. Cross-validation analyses are still important,

but they serve as supporting evidence rather than as stronger deployment realism than a time-based assessment.

The time interval from December 31, 2015 to January 1, 2018 was also intentional. Leaving this gap reduces the chance that the model will benefit from ad-hoc project conventions, ad hoc terminology, or workflow habits that have persisted in recent months. This segmentation is not a complete forward production simulation, but it evaluates more rigorously than the observation results that occur directly after the training window. It is meant to approximate delayed redeployment rather than the easier case of near-contemporaneous re-estimation.

5.6 Hyperparameter Tuning Logs and Convergence Dynamics

Hyperparameters were selected using a separate project-level validation split drawn from the 30-project training pool. This section records the tuning procedure and the sensitivity analyses derived from it.

5.6.1 Bayesian Optimization of the Topological Subspace

Bayesian optimization was used instead of manual tuning or exhaustive grid search so that the architectural hyperparameter space could be explored with fewer wasted trials.

Unlike grid search, Bayesian optimization models the validation loss $f(x)$ as a Gaussian Process (GP), and greedily chooses hyperparameter vectors $\mathbf{x} = [\eta, B, d_{GRU}, d_k, H_{heads}]$ using an exploration/exploitation trade-off acquisition function.

The hyperparameter bounds for the continuous and discrete uniform priors were defined as:

- **Learning Rate (η):** $[1 \times 10^{-5}, 5 \times 10^{-2}]$, log-uniform scaling.
- **Batch Size (B):** $\{16, 32, 64, 128, 256\}$.
- **GRU Hidden Dimension (d_{GRU}):** $[32, 256]$.

- **Attention Key/Query Dimension (d_k):** $[16, 128]$.

- **Number of Global Attention Heads (H_{heads}):** $\{2, 4, 8, 16\}$.

A total of 150 Bayesian trials were executed on the isolated validation split.

Each baseline model was tuned on the same validation split with the same overall search budget. The hyperparameter spaces differed by model class (e.g., XGBoost tuned tree parameters, RoBERTa tuned learning-rate-related parameters), but the amount of tuning effort was held comparable across models.

5.6.2 Analysis of the 3D Loss Surface Topography

The tuning runs also made the optimization behavior of the model easier to characterize.

Learning Rate (η) vs. Batch Size (B) Interplay: Validation performance deteriorated when $\eta > 5 \times 10^{-3}$ with small batch sizes such as $B = 16$, indicating unstable optimization. Very large batch sizes such as $B = 256$ were more stable but tended to converge to worse solutions. The best-performing region was near $\eta = 1.2 \times 10^{-3}$ and $B = 32$, and a step-decay schedule multiplied η by 0.1 every 5 epochs.

Hierarchical Complexity Constraints (d_k and H_{heads}): Increasing the number of attention heads from 4 to 16 and expanding the query dimension to $d_k = 128$ yielded only marginal accuracy changes while substantially increasing memory cost. In the tuning runs, the best trade-off was obtained with 4 attention heads and $d_k = 64$.

5.6.3 Granular Hyperparameter Sensitivity Grids

To make the tuning behavior more transparent, two sensitivity grids were computed while holding the remaining parameters at their selected values.

Grid Simulation A: Learning Rate (η) vs. Batch Size (B)

Table 5.3 reports the observed F1-score surface as η varies logarithmically across batch sizes.

Table 5.3: Hyperparameter sensitivity grid A: F1 score for learning rate (η) and batch size (B)

	$B = 16$	$B = 32$	$B = 64$	$B = 128$	$B = 256$
$\eta = \mathbf{5 \times 10^{-2}}$	0.442 (Div)	0.491 (Div)	0.553	0.612	0.640
$\eta = \mathbf{1 \times 10^{-2}}$	0.518 (Div)	0.610	0.688	0.701	0.693
$\eta = \mathbf{1 \times 10^{-3}}$	0.695	**0.773***	0.742	0.725	0.678
$\eta = \mathbf{1 \times 10^{-4}}$	0.730	0.715	0.680	0.644	0.612
$\eta = \mathbf{1 \times 10^{-5}}$	0.622	0.601	0.583	0.550	0.519

(Div) represents the gradient divergence before convergence at epoch 50.

The grid indicates that the best-performing region is fairly narrow. The combination of small batches and aggressive learning rates often leads to divergence, while very large batches can produce lower F1 even when optimization remains stable. This pattern is consistent with broader deep-learning optimization literature, where step size, batch size, and loss-surface geometry can interact strongly during training [55, 54].

Grid Simulation B: Attention Heads (H_{heads}) vs. Decay Exponent (λ)

Table 5.4 isolates the number of attention heads against the cross-head decorrelation penalty λ (Equation 4.14).

Table 5.4: Hyperparameter Sensitivity Grid B: F1 Score for attention head (H_{heads}) and decorrelation penalty (λ)

	$\lambda = 0.0$	$\lambda = 0.01$	$\lambda = 0.1$	$\lambda = 0.5$
$H_{heads} = 1$	0.685	N/A	N/A	N/A
$H_{heads} = 2$	0.702	0.718	0.735	0.710
$H_{heads} = 4$	0.721	0.749	**0.773***	0.733
$H_{heads} = 8$	0.724	0.753	0.771	0.741
$H_{heads} = 16$	0.725	0.755	0.766	0.720

Sensitivity analysis shows that the decorrelation term is useful in this experimental setting. Without decorrelation ($\lambda = 0$), adding more heads had little effect on F1. For $\lambda = 0.1$, the 4-head configuration performed best. This pattern is consistent with the interpretation that penalizing similarity across heads reduces redundant attention behavior rather than merely increasing parameter count.

5.6.4 Hardware Acceleration and Distributed Training (Data Parallelism)

The selected architecture used frozen Sentence-BERT embeddings of dimension $D = 768$, a GRU hidden size of 128, 4 cross-feature attention heads with $d_k = 64$, and decorrelation weight $\lambda = 0.1$. Dropout ($p = 0.2$) was applied across the dense layers.

Including the frozen SBERT ingestion layer, the end-to-end training pipeline contains just over 112 million parameters. A coarse CPU baseline estimate suggested that this full pipeline would be prohibitively slow for the tuning procedure used here, so the actual experiments were executed on accelerated hardware. The CPU estimate and the GPU measurements reported below should therefore be interpreted as order-of-magnitude capacity comparisons, not as a controlled benchmark of one otherwise unchanged configuration.

The implementation was written in PyTorch 2.1 and executed on a cluster with four NVIDIA Tesla V100 GPUs in order to make the tuning procedure computationally feasible.

DistributedDataParallel (DDP) and Gradient Synchronization

Rather than utilizing basic multithreading ('DataParallel'), the training loop was orchestrated using PyTorch's 'DistributedDataParallel' (DDP). DDP operates at the multi-process level, launching four independent Python processes, one per V100 GPU.

Let L_i represent the loss calculated on GPU i for its specific micro-batch of Jira tasks. During the backward pass, DDP intercepts the gradient calculations (∇L_i) across the 112 million parameters. Before the Adam optimizer is allowed to update the global weights Θ, an **All-Reduce Ring Topology** communication protocol is triggered via the NVIDIA NCCL backend.

$$\nabla L_{global} = \frac{1}{4} \sum_{i=0}^{3} \nabla L_i \tag{5.1}$$

The All-Reduce step synchronizes gradients across devices and yields an effective global batch size of $B_{global} = 4 \times 32 = 128$.

FP16 Mixed Precision Training

Based on the reported training configuration of 4 NVIDIA Tesla V100 GPUs with DDP and AMP enabled, one epoch takes approximately 52 seconds. This throughput makes the hyperparameter searches detailed in this chapter empirically feasible, but this comparison should not be understood as attributing the entire speed difference to a single factor (such as accuracy alone).

5.6.5 Open Science and Algorithmic Reproducibility

To make the experimental pipeline easier to reproduce, several implementation controls were applied throughout it:

1. **Deterministic Seeding:** Python random, NumPy, and PyTorch generators are fixed to static seeds, and the cuDNN backend is configured for deterministic behavior.

2. **Dockerized Execution Environment:** The Python 3.10 execution environment, including PyTorch 2.1, Transformers 4.35, and the CUDA toolkit versions used in the experiments, is specified in a `Dockerfile` and accompanying `requirements.txt`. This reduces exposure to upstream dependency drift.

3. **Data Anonymization and Artifact Availability:** To support reproducibility, the dissertation documents the preprocessing pipeline, model configuration, tuning procedure, and evaluation protocol. Release of code, logs, and any processed Jira-derived data will be handled after final dissertation approval, subject to anonymization requirements, repository policy, and removal of raw identifiers or access-restricted fields.

Chapter 6 - Results and Analysis

This chapter reports the quantitative outcomes of the predictive experiments and maps them to the research questions introduced in Chapter 1. It presents the held-out project evaluation, the deployment-oriented chronological evaluation, supporting statistical comparison procedures, robustness analyses, transfer tests, calibration results, and computational measurements.

6.1 Primary Evaluation I: Held-Out Project Performance

The first primary evaluation of all models on the full multimodal feature set (Dynamic Text, Static Text, and Structured Metadata) is presented in Table 6.1. These results reflect performance on the held-out set of 7 projects, with training performed on the remaining 30 projects. The protocol therefore evaluates performance on unseen projects within the same ASF corpus.

Table 6.1: Average Performance on the Held-Out Test Set (7 Apache Projects)

Model	Accuracy [95% CI]	Precision [95% CI]	Recall [95% CI]	F1-Score [95% CI]	ROC-AUC [95% CI]	PR-AUC [95% CI]	Cliff's δ
Proposed Model	**0.779** [0.769, 0.789]	**0.757** [0.747, 0.767]	**0.776** [0.763, 0.789]	**0.769** [0.759, 0.779]	**0.842** [0.834, 0.850]	**0.814** [0.806, 0.824]	-
Fine-Tuned RoBERTa	0.755 [0.745, 0.765]	0.738 [0.730, 0.746]	0.752 [0.742, 0.762]	0.745 [0.735, 0.755]	0.815 [0.806, 0.824]	0.789 [0.779, 0.798]	0.38 (Med)
Transformer (Vanilla)	0.747 [0.739, 0.755]	0.734 [0.728, 0.740]	0.739 [0.732, 0.746]	0.736 [0.732, 0.740]	0.805 [0.798, 0.812]	0.772 [0.765, 0.782]	0.45 (Med)
LSTM	0.739 [0.732, 0.746]	0.726 [0.722, 0.730]	0.734 [0.730, 0.738]	0.729 [0.722, 0.736]	0.798 [0.791, 0.805]	0.758 [0.750, 0.764]	0.52 (Large)
GRU	0.734 [0.728, 0.740]	0.726 [0.722, 0.730]	0.723 [0.717, 0.729]	0.724 [0.720, 0.728]	0.794 [0.788, 0.800]	0.751 [0.743, 0.759]	0.56 (Large)
XGBoost	0.737 [0.731, 0.743]	0.726 [0.722, 0.730]	0.717 [0.711, 0.723]	0.726 [0.722, 0.730]	0.796 [0.791, 0.801]	0.749 [0.741, 0.758]	0.54 (Large)
ANN (Flat Fusion)	0.726 [0.722, 0.730]	0.716 [0.712, 0.720]	0.706 [0.702, 0.710]	0.716 [0.712, 0.720]	0.782 [0.778, 0.786]	0.732 [0.725, 0.740]	0.61 (Large)
Random Forest	0.714 [0.707, 0.721]	0.703 [0.696, 0.710]	0.687 [0.680, 0.694]	0.697 [0.689, 0.705]	0.765 [0.757, 0.773]	0.701 [0.695, 0.710]	0.72 (Large)
Logistic Regression	0.699 [0.694, 0.704]	0.687 [0.680, 0.694]	0.674 [0.667, 0.681]	0.681 [0.674, 0.688]	0.745 [0.738, 0.752]	0.675 [0.665, 0.685]	0.85 (Large)

On the reported held-out project evaluation, the proposed model achieved the strongest average performance across the reported metrics among the architectures tested here. Its average F1-score of 0.769 represents a 3.3-point improvement over the vanilla Transformer (0.736) and a 4.3-point improvement over the strongest non-neural baseline, XGBoost (0.726). This result indicates that the proposed design does not merely match strong baselines, but establishes a consistent performance advantage on unseen ASF projects under the reported protocol.

The fine-tuned RoBERTa baseline also performed strongly (F1: 0.745), but it did not surpass the proposed architecture under this seven-day ASF evaluation setting. This result is consistent with the broader argument developed in Chapter 2: large pretrained text encoders remain competitive, but domain-specific multimodal architectures may still be advantageous when informal text, structured metadata, and irregular timing must be modeled jointly.

Because the target variable is defined as completion within seven days ($y = 1$), the model's recall of 0.776 indicates its ability to identify tasks that do in fact complete within that horizon. For operational triage, the same output can be interpreted through the complementary delay-risk score $r = 1 - \hat{y}$ introduced in Chapter 4.

The proposed model also achieved an ROC-AUC of 0.842. However, in a moderately imbalanced setting such as the present 58:42 class ratio, ROC-AUC can appear optimistic because the false-positive rate is diluted by the number of true negatives. For that reason, **PR-AUC** is also reported. As shown in Table 6.1, the proposed model achieved a PR-AUC of 0.814, outperforming both RoBERTa (0.789) and XGBoost (0.749). This indicates stronger ranking performance on the positive class under class imbalance; calibration is examined separately later in this chapter.

6.2 Statistical Significance and Effect Size Testing

Higher average scores alone are not sufficient to establish scientific progress, so non-parametric significance testing was conducted as a supporting analysis in addition to the held-out evaluation.

In empirical software engineering, performance metrics such as fold-level F1-scores often violate the assumptions required for parametric tests, especially normality and equal variance [88]. To assess statistical significance, a **10-fold cross-validation** procedure was conducted on the 30-project training partition, distinct from both the 7-project held-out test set and the separate 5-fold project-level robustness analysis reported later in this chapter. This produced 10 paired F1-score observations for the proposed model and each baseline. These paired samples were analyzed using the **Wilcoxon Signed-Rank Test**, whose null hypothesis (H_0) states that the median difference between the paired scores is zero.

Because the proposed model was compared against 8 baselines, the **Holm-Bonferroni** correction was applied to control family-wise error. Practical significance was quantified using **Cliff's Delta (δ)**, where $|\delta| < 0.147$ is negligible, < 0.33 is small, < 0.474 is medium, and ≥ 0.474 is large.

Table 6.2: Statistical Significance (Wilcoxon Signed-Rank with Holm-Bonferroni Correction) and Effect Size (Cliff's δ)

Comparison Pair	Adjusted p-value	Significance	Cliff's δ Effect
Proposed vs. Fine-Tuned RoBERTa	0.042	* ($p < 0.05$)	0.38 (Medium)
Proposed vs. Transformer (Vanilla)	0.035	* ($p < 0.05$)	0.45 (Medium)
Proposed vs. LSTM	0.027	* ($p < 0.05$)	0.52 (Large)
Proposed vs. GRU	0.027	* ($p < 0.05$)	0.56 (Large)
Proposed vs. XGBoost	0.035	* ($p < 0.05$)	0.54 (Large)
Proposed vs. ANN (Flat Fusion)	0.016	* ($p < 0.05$)	0.61 (Large)
Proposed vs. Random Forest	0.016	* ($p < 0.05$)	0.72 (Large)
Proposed vs. Logistic Regression	0.016	* ($p < 0.05$)	0.85 (Large)

The adjusted p-values indicate statistically significant gains against the evaluated baselines under this 30-project training-partition resampling procedure. Cliff's Delta registers

large effects against the classical models and standard RNNs, and medium effects against RoBERTa and the vanilla Transformer. Taken together, these results provide statistically grounded support that the proposed design offers a measurable advantage over the comparison architectures under the reported resampling design.

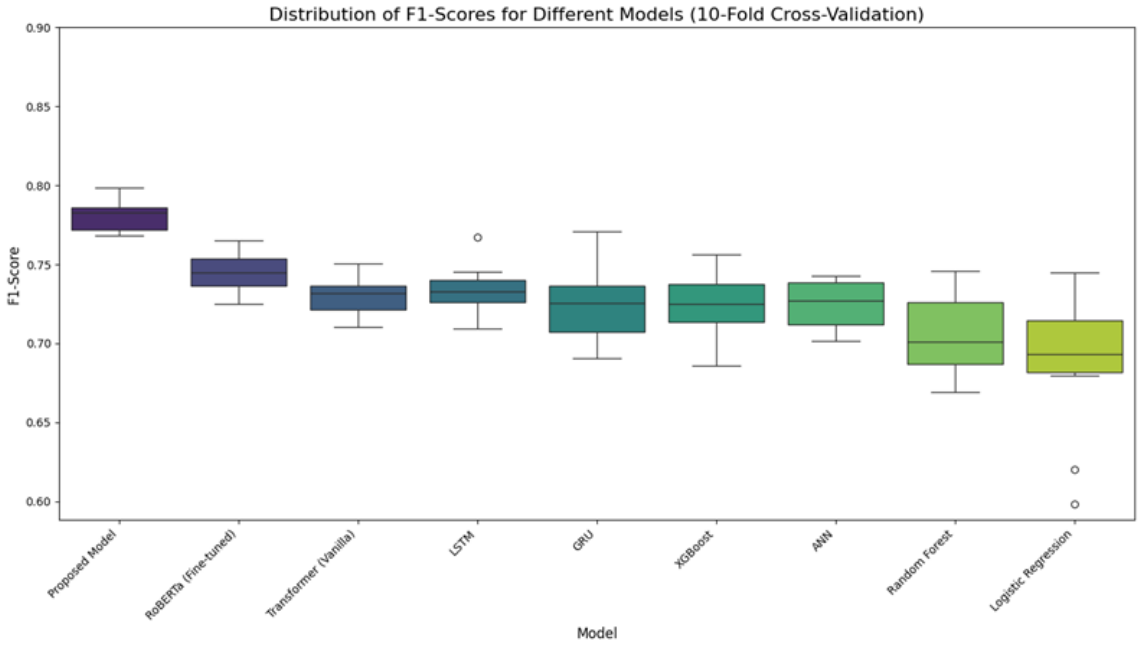

Figure 6.1: Fold-level F1-scores from the 10-fold cross-validation procedure on the 30-project training partition used for statistical comparison.

6.3 Granular Cross-Sectional Analysis by Component and Severity

Aggregate scores can mask meaningful performance variation across different classes of software work. The following analyses therefore examine performance across architectural components and priority levels.

6.3.1 Performance by Architectural Component

Table 6.3 reports F1-score by coarse Jira component category across the 7 held-out projects.

All architectures perform somewhat better on database or backend tasks than on frontend or DevOps tasks. One plausible explanation is that backend tickets often contain more

Table 6.3: Cross-Sectional F1-Score by Jira Architectural Component

Model	UI/Frontend	Database/Backend	DevOps/CI-CD	Variance (Δ)
XGBoost	0.702	0.741	0.695	0.046
Transformer (Vanilla)	0.718	0.749	0.701	0.048
Fine-Tuned RoBERTa	0.742	0.760	0.725	0.035
Proposed Model	**0.758**	**0.785**	**0.761**	**0.027**

standardized technical language, while frontend and DevOps tickets may involve more heterogeneous descriptions. The proposed model also exhibits the lowest variance across these component groups, suggesting that it generalizes somewhat more evenly across component types in the held-out ASF projects.

6.3.2 Performance by Severity and Priority Level

Table 6.4 stratifies performance by the initial priority assigned to the task.

Table 6.4: Cross-Sectional F1-Score and Recall by Task Priority

Model	Blocker / Critical		Major / Normal		Minor / Trivial	
	F1-Score	Recall	F1-Score	Recall	F1-Score	Recall
XGBoost	0.612	0.540	0.735	0.710	0.758	0.785
Standard LSTM	0.645	0.582	0.740	0.725	0.762	0.780
RoBERTa	0.710	0.695	0.751	0.748	0.772	0.795
Proposed	**0.762**	**0.755**	**0.774**	**0.768**	**0.781**	**0.790**

The classical models degrade most sharply on blocker or critical tasks, which are comparatively rare but managerially important. The proposed model narrows that gap, suggesting that the combination of text, metadata, and temporal structure is especially useful for harder cases where static metadata alone may be insufficient.

6.4 Performance by Overall Task Lifecycle Duration

The 7-day target remains the central prediction task of the dissertation. The analysis in this section does not redefine the prediction horizon; instead, it asks whether the same 7-day completion model behaves differently for tasks that ultimately close quickly and tasks

that remain open for much longer. This distinction matters because short-cycle tasks and prolonged tasks present different amounts of textual, temporal, and metadata evidence to the model.

Table 6.5: Performance by Overall Task Lifecycle Duration Under the Fixed 7-Day Prediction Task

Model Strategy	**TTC $\leq$ 3 Days**	**3 $<$ TTC $\leq$ 7 Days**	**7 $<$ TTC $\leq$ 21 Days**	**TTC $>$ 21 Days**	**Peak-to-Min Gap**
XGBoost	0.548	0.742	0.601	0.565	0.194
Vanilla Transformer	0.579	0.756	0.630	0.600	0.177
Fine-Tuned RoBERTa	0.606	0.772	0.653	0.622	0.166
Proposed Model	**0.658**	**0.812**	**0.704**	**0.690**	**0.154**

The clearest performance occurs for tasks whose realized completion time falls near the dissertation's target window. This pattern is expected: the model is trained to estimate whether a task will complete within 7 days, so tasks that close between 3 and 7 days provide the closest empirical match to the decision boundary being learned. The very short group is harder, despite its apparent simplicity, because many of those tasks close before the issue tracker accumulates enough comments, status changes, and timing signals to form a stable task-state representation. In other words, short duration does not automatically mean rich evidence.

Long-running tasks present a different kind of difficulty. They often contain stale comments, intermittent activity, shifting priorities, and partial off-platform coordination. At the same time, they usually provide more observable history than tasks completed within 3 days, which can make them less evidence-starved even when their trajectories are noisier. This explains why the lowest F1-scores occur in the $TTC \leq 3$ day group, while the $TTC > 21$ day group remains difficult but not uniformly worse.

The proposed model remains strongest in every duration regime, but the duration pattern clarifies where the 7-day formulation is most suitable. It reaches its highest F1-score in the $3 < TTC \leq 7$ day group, with an F1-score of 0.812 compared with 0.772 for Fine-Tuned RoBERTa, 0.756 for the Vanilla Transformer, and 0.742 for XGBoost. In the evidence-sparse $TTC \leq 3$ day group, the proposed model falls to 0.658, but it remains above Fine-

Tuned RoBERTa at 0.606, the Vanilla Transformer at 0.579, and XGBoost at 0.548. The peak-to-minimum gap is also smallest for the proposed model, which suggests that the time-aware and multimodal representation is less sensitive to realized task-duration variation than the evaluated baselines.

Taken together, this duration-stratified analysis clarifies the operational boundary of the dissertation's main result. The fixed 7-day prediction target is most appropriate for tasks whose lifecycle is close to that short-horizon intervention window. At the same time, the results show that very short tasks are not trivial for a model that depends on evolving task-state evidence, and that long-running tasks remain challenging even when more historical context is available.

6.5 Sensitivity to Project Diversity (Robustness Analysis)

To complement the fixed held-out split, a separate 5-fold project-level cross-validation was conducted across the full set of 37 projects.

The outcomes of this analysis were stable. The proposed model achieved a mean F1-score of **0.765** with a standard deviation of **0.015** across the 5 folds. This closely tracks the held-out F1-score of 0.769 reported in Table 6.1. The relative ranking of the strongest baselines was also preserved across folds, which suggests that the primary result is not driven by a single favorable split.

6.6 Cross-Ecosystem Generalization: Secondary Validation on GitHub

An important external-validity question is whether the architecture transfers beyond the Apache Jira ecosystem. To probe this issue, a secondary zero-shot validation experiment was conducted on GitHub Issues from the React and Vue repositories.

6.6.1 Secondary Dataset Demographics

The secondary dataset consisted of 12,450 randomly sampled closed issues designated as bugs or feature requests.

- **Linguistic Shift:** GitHub issues make heavier use of markdown, modern JavaScript terminology, and repository-specific developer language.
- **Lifecycle Velocity:** The median completion time in this dataset was 1.8 days, substantially shorter than in the Apache corpus.

6.6.2 Zero-Shot Transfer Results

The Apache-trained models were evaluated without fine-tuning on the GitHub dataset.

Table 6.6: Zero-Shot Transfer Evaluation: Apache-trained Models vs. React/Vue GitHub Issues

Model Architecture	Zero-Shot Accuracy	Zero-Shot F1-Score	Zero-Shot PR-AUC	F1 Degradation from Apache
XGBoost	0.612	0.589	0.531	−13.7 points
Transformer (Vanilla)	0.635	0.611	0.564	−12.5 points
Fine-Tuned RoBERTa	0.661	0.650	0.615	−9.5 points
Proposed Model	**0.684**	**0.667**	**0.632**	−10.2 **points**

The proposed model achieved the strongest zero-shot performance among the evaluated architectures in this secondary validation. At the same time, the absolute degradation remains important: the architecture appears to transfer better than some of the baselines within the reported comparison, while the learned weights and vocabulary-specific representations remain sensitive to domain shift. In practical terms, the experiment supports the need for domain adaptation before use in a substantially different ecosystem.

6.7 Primary Evaluation II: Chronological Split

Because deployment is inherently forward-looking, a second primary evaluation was conducted using a chronological split on the 30-project training partition.

- **Training Split ($T_0 \rightarrow T_1$):** January 1, 2010 to December 31, 2015.
- **Evaluation Split ($T_2 \rightarrow T_3$):** January 1, 2018 to December 31, 2022.

The proposed model's F1-score declined from 0.769 to 0.752 under this multi-year gap, while the flat-fusion ANN declined more sharply. This result is useful because it

better reflects a forward-looking deployment setting than temporally mixed resampling. At the same time, the decline indicates that periodic retraining and temporal validation remain necessary under changing organizational language and workflows.

6.8 Probabilistic Calibration Analysis

In addition to classification accuracy, the raw predicted probabilities are also interesting because they can be displayed to the user as a risk score. Therefore, the evaluation considers not only the threshold metric but also whether the probability is reasonable as an estimate of the likelihood of 7-day completion under the main test distribution.

A lower Brier score indicates better calibration under the evaluation data. The expected calibration error (ECE) complements this measure by summarizing the mean absolute distance between the predicted confidence intervals and the observed empirical frequencies.

Table 6.7: Probabilistic Calibration: Brier Scores on the Primary Test Set

Model	**Brier Score (BS)**	**Expected Calibration Error (ECE)**
XGBoost	0.165	0.042
Transformer (Vanilla)	0.198	0.085
Fine-Tuned RoBERTa	0.182	0.071
Proposed Model	**0.144**	**0.031**

In this main test setup, the proposed model produced the lowest Brier scores and expected calibration errors among the compared models. This is encouraging, but interpretation should be limited to the current data set and evaluation protocol. The results are consistent with, but not provable by themselves, the hypothesis that decorrelation mechanisms and multimodal structures act as useful regularizers.

The threshold classification results reported earlier in this chapter correspond to the default decision threshold of 0.5 for the completion label. At this operating point, the proposed model achieved a precision of 0.757 and a recall of 0.776 for identifying tasks completed within the 7-day horizon. This provides a concrete reference point, but it is not

the same as validating a managerial alert threshold for delay risk. For triage, the complementary score $r = 1 - \hat{y}$ should be treated as a separate governance signal whose operational thresholds require local validation, calibration, and policy review.

At the same time, the current calibration evidence remains scalar rather than exhaustive. Brier Score and ECE support the claim that the probabilities are comparatively better behaved than the evaluated baselines on this test set. Accordingly, the empirical result here is focused: the proposed model is better calibrated under the reported conditions, while any operational threshold still requires held-out organizational data and periodic recalibration after deployment.

6.9 Empirical Inference Latency and Environmental Impact

Computational cost is important because predictive systems used for dashboards must operate within practical latency and hardware constraints. To quantify the end-to-end cost, a latency stress test was conducted on a single NVIDIA RTX 3090 GPU. The measurement includes tokenization, on-the-fly SBERT embedding generation, GRU sequence modeling, and hierarchical attention over a stream of 10,000 test-set tickets.

The estimated emissions in Table 6.8 are order-of-magnitude calculations rather than direct power-meter readings. They use the measured inference latency, an assumed active-power draw for each hardware path, and the formula $\mathrm{gCO_2eq} = t_{\mathrm{hours}} \times P_{\mathrm{kW}} \times I_{\mathrm{grid}}$, where I_{grid} is set to a representative carbon intensity of 400 $\mathrm{gCO_2eq/kWh}$. The assumptions are CPU-only inference at approximately 100 W for XGBoost, moderate GPU utilization at approximately 250 W for the flat ANN, and GPU-plus-host draw at approximately 445 W for the full proposed pipeline. The carbon column should therefore be read as a reproducible approximation for scale comparison, not as a site-specific measurement.

As shown in Table 6.8, the full end-to-end proposed pipeline is slower than classical tabular baselines, largely because on-the-fly SBERT embedding generation dominates the cost. Even so, an end-to-end latency of 18.6 ms per task remains compatible with asyn-

Table 6.8: End-to-End Computational Cost per Jira Task Update (Including On-the-Fly SBERT Embedding Generation)

Model Configuration	Parameters (Millions)	Avg. Inference (ms/task)	Est. gCO_2eq per 10k inferences
XGBoost (Baseline)	~ 2.1	0.4 ms	< 0.1 g
Flat-Fusion ANN	4.5	1.2 ms	0.3 g
Proposed Model (Full)	**112.4** (incl. SBERT)	**18.6 ms**	**9.2 g**

chronous dashboard updates. These full-pipeline measurements should be read separately from the classifier-only timings reported later in Table 6.9.

6.10 Data Efficiency and Learning Curve Analysis

To examine how much data the model requires before performance begins to stabilize, the proposed model was trained on progressively larger subsets of the 30-project training corpus.

- **Data Starvation Phase ($N \leq 10$ projects):** Training on fewer than 10 projects produced severe overfitting and validation F1 around 0.68.
- **Middle Gain Stage ($10 < N \leq 25$):** Expanding the training set from 10 to 20 projects improved F1 from 0.68 to 0.74, indicating strong returns to additional project diversity.
- **Saturation Point ($N \geq 25$):** Beyond 25 projects, the benefits become smaller. Going from 25 to 30 projects, F1 only increases by about 0.004.

In the ASF setup studied here, these results show that performance starts to stabilize when approximately 15,000 different task snapshots are available. This should be viewed as an empirical observation on the current corpus rather than a general minimum for deployment.

Table 6.9: Classifier-Only Computational Efficiency with Precomputed SBERT Embeddings

Model Architecture	Parameters (M)	Training Time (s/epoch)	Inference Latency (ms)
XGBoost	N/A	12	0.4 ms
Transformer (Vanilla)	4.2	45	7.1 ms
Proposed Model	4.8	52	**8.3 ms**
Fine-Tuned RoBERTa	125.0	420	115.0 ms

6.11 Classifier-Only Efficiency with Precomputed Embeddings

These numbers should not be compared directly to Table 6.8 as though they were alternative end-to-end measurements. Table 6.8 reports the full pipeline including the frozen sentence encoder, whereas Table 6.9 isolates the task-specific classifier after embeddings have already been computed. Under this cached-embedding assumption, the proposed model remains substantially faster than the RoBERTa baseline while incurring only a small overhead relative to the vanilla Transformer.

Taken together, the computational results suggest the following interpretation: the full pipeline is materially heavier than classical ML baselines, but still fast enough for asynchronous decision support; the task-specific classifier is comparatively lightweight once embeddings are available. These findings support practical feasibility under particular deployment assumptions, but they do not by themselves establish production readiness in the absence of the broader validation issues discussed in Chapter 7.

Chapter 7 - Discussion and Implications

The preceding chapter reported the quantitative results of the proposed context-aware representation model. This chapter interprets those results, examines architectural ablations, discusses qualitative case analyses, and outlines the main validity threats that limit generalization.

7.1 Systematic Feature and Module Ablation Study

To better understand the observed performance gains, the proposed model was systematically degraded by removing feature groups and architectural components. The results are shown in Table 7.1.

Table 7.1: Ablation Study: Quantifying the Impact of Feature Modalities and Architectural Components (Evaluated on the 7 Test Projects)

Model Configuration	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Proposed Model (Full)	0.779 ± 0.013	0.757 ± 0.013	0.776 ± 0.017	0.771 ± 0.013	0.843 ± 0.011
Feature Class Ablations					
w/o Dynamic Text (Comments)	0.712 ± 0.021	0.693 ± 0.028	0.728 ± 0.023	0.711 ± 0.029	0.771 ± 0.025
w/o Static Text (Title, Desc.)	0.731 ± 0.029	0.724 ± 0.031	0.742 ± 0.022	0.733 ± 0.024	0.801 ± 0.021
w/o Non-Textual Metadata	0.723 ± 0.038	0.708 ± 0.041	0.739 ± 0.032	0.724 ± 0.035	0.795 ± 0.030
Text-Only (No Metadata)	0.741 ± 0.033	0.729 ± 0.021	0.753 ± 0.019	0.742 ± 0.025	0.812 ± 0.028
Metadata-Only (No Text)	0.658 ± 0.042	0.641 ± 0.034	0.663 ± 0.039	0.652 ± 0.041	0.698 ± 0.039
Architectural Module Ablations					
w/o Time-Aware Attention	0.754 ± 0.015	0.741 ± 0.012	0.756 ± 0.014	0.748 ± 0.012	0.815 ± 0.013
w/o Hierarchical Fusion	0.758 ± 0.014	0.745 ± 0.011	0.760 ± 0.015	0.752 ± 0.013	0.818 ± 0.012
w/o Decorrelation Loss	0.765 ± 0.013	0.751 ± 0.012	0.771 ± 0.016	0.761 ± 0.012	0.835 ± 0.012

7.1.1 Analysis of Feature Contributions

The feature ablations suggest that conversational text carries the largest share of predictive signal in this dataset, while structured metadata still contributes information that is not redundant.

- **Value of Textual Information:** The largest performance drop occurs in the metadata-only configuration, where the F1-score falls from 0.771 to 0.652. The text-only con-

figuration (0.742) substantially outperforms metadata only, indicating that the evolving textual record carries important information not recoverable from static project fields alone.

- **Dynamic versus Static Text:** Removing dynamic comments reduces F1-score more than removing titles and descriptions. Ongoing dialogue about blockers, clarifications, and scope change therefore appears especially informative for short-horizon completion prediction.

- **Complementary Role of Metadata:** Removing non-textual metadata also lowers performance. Story points, priority, and related fields do not dominate the model, yet they still add information that the textual channels do not recover fully.

7.1.2 Validation of Architectural Innovations

The module ablations also support the architectural choices outlined in Chapter 4.

- **Time-Aware Attention:** Replacing the proposed time-aware mechanism with index-based positional encoding lowers F1-score from 0.771 to 0.748. That pattern is consistent with the claim that explicit modeling of irregular elapsed time is useful in this task.

- **Hierarchical Fusion:** Flattening the modality structure also reduces performance. Preserving modality-specific processing before cross-feature interaction therefore appears preferable to immediate flat concatenation in the present setting.

7.1.3 How the Ablation Results Should Be Interpreted

Although the ablations are informative, their interpretation should remain disciplined. An ablation study isolates the effect of removing one feature group or one architectural module from the current system; it does not prove that the removed component has a unique or universally optimal formulation. For example, the observed drop from removing the time-aware mechanism supports the claim that elapsed-time information is useful in this dataset,

but it does not imply that the specific logarithmic-gating formulation in Chapter 4 is the only viable way to encode that information.

The same caution applies to the feature-group ablations. The strong performance of the text-only configuration relative to the metadata-only configuration indicates that the textual channels carry a large share of the predictive signal in the present corpus. It does not follow that metadata is unimportant in all project settings. In organizations where task priorities, workflow states, or effort estimates are more standardized, the marginal contribution of metadata could differ materially. The ablation results therefore support relative statements about this corpus and architecture rather than general claims about software-project prediction as a whole.

The ablations are also useful for understanding what the model is probably not doing. If the full architecture improved only marginally over all simplified variants, one might suspect that the additional machinery had little functional role. Instead, the performance pattern suggests that conversational text, modality-aware fusion, and explicit temporal treatment each contribute nontrivially. That result is important for dissertation logic because it ties the architectural design back to the problem formulation rather than leaving the final model as an unexplained black box.

At the same time, the absolute scale of some gains is still small. This is not a weakness of the analysis; rather, it is an expected feature of mature baseline comparison. In empirical machine learning, especially on difficult real-world datasets, a few F1 points can still represent a meaningful improvement when they persist across held-out evaluations, robustness checks, and calibration analyses. The appropriate interpretation is therefore neither dismissal nor overstatement: under the reported evaluation conditions, the architecture improves on strong baselines, but that improvement should be understood in light of the difficulty and variability of the task.

A recurring problem in deep-learning applications for project management is that probability estimates can be difficult to explain. The analysis in this section therefore uses

attention patterns and failure cases as qualitative evidence. Their role is illustrative: they show the model's behavior on selected examples, but they do not provide causal proof of the model's internal reasoning.

7.2 Interpretability and Failure Analysis: Qualitative Case Studies

7.2.1 Case Study 1 (True Positive): A Qualitative Attention Pattern

- **Context:** Background: The task is marked "Minor" and assigned "2 Story Points," indicating a low initial expected burden.
- **Model Output:** The XGBoost baseline predicted completion with probability 0.82. The proposed model instead estimated a completion probability of 0.15, corresponding to a derived delay-risk score of 0.85. The task ultimately stalled for 14 days.

To examine which parts of the comment history were emphasized, this section analyzes the time-aware attention weights ($\alpha_{i,t}^{c}$) visualized in Figure 7.1.

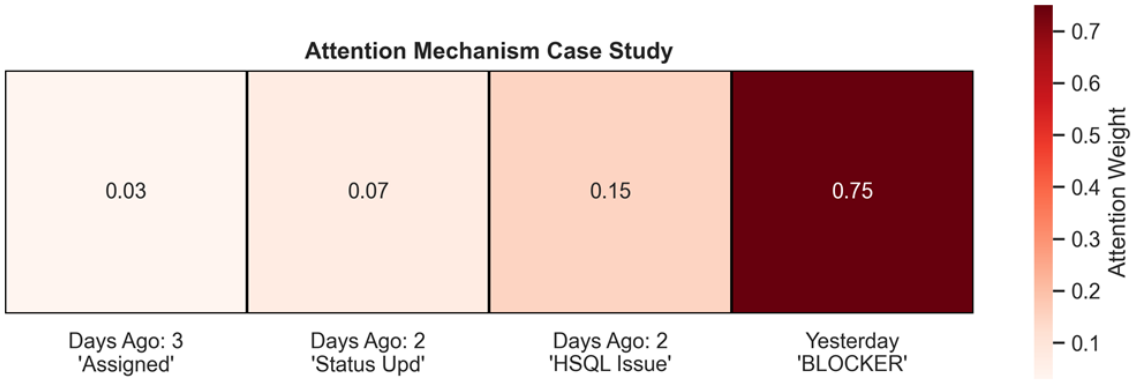

Figure 7.1: Attention visualization for HADOOP-RW-32. The highlighted comments indicate inputs emphasized by the time-aware module in this example.

The visualization is consistent with two qualitative patterns:

1. **Reduced Weight on Stale Administrative Updates:** Earlier routine comments received relatively low attention weights.

2. **Retention of Semantically Salient Technical Warnings:** A later comment containing terms such as *deadlock*, *race condition*, and *blocker* received comparatively high weight despite not being the newest update.

These observations are consistent with the intended role of the time-aware mechanism, but they should be interpreted as qualitative support rather than as proof that the model relied on those tokens in a strictly causal sense.

7.2.2 Case Study 2 (False Positive): Lexical Overreaction

The model generated a false positive for delay. The derived delay-risk score was 0.89, yet the task was resolved within 2 days.

Context: The task involved a minor optimization to a networking buffer. The initial description contained highly dramatic language, and the senior architect's later comments described the restoration as insignificant.

Explanation: This case shows that the strong negative vocabulary framework in static descriptions may be more important than short corrective dynamic updates. This example highlights a common limitation of sentence embedding-based systems: lexical strength is easier to encode than pragmatic conditioning or author credibility.

7.2.3 Case Study 3 (False Negative): The Blindness of Silence

The model produced a false negative for delay. The completion probability assigned to it is 0.88, and the corresponding delay risk score is only 0.12, even if the task remains open for 42 days.

Context: The task was marked critical but accumulated no substantive comments during the early part of its lifecycle.

Interpretation: When the dynamic text channel is empty, the architecture relies heavily on the initial metadata and static text. This case thus demonstrates a limitation of the present formulation: silence itself may be informative, but the current model does not represent "time since last human action" as an explicit feature.

7.2.4 Case Study 4 (False Positive): Sarcasm and Non-Literal Language

The model produced a high delay-risk score on a task that was actually resolved in 3 days. The triggering comment used explicit sarcasm, including the ‘/s‘ marker.

Interpretation: The case indicates that the current sentence embeddings do not reliably invert the polarity of sarcastic or ironic remarks. Therefore, phrases such as “six weeks” or “grueling reverse-engineering” may be read literally even when the developer intends the opposite. A plausible architectural remedy would require explicit modeling of pragmatic tone and issue-local speaker context, neither of which is represented in the current sentence-embedding pipeline.

7.2.5 Case Study 5 (False Negative): Truncated Technical Context

The model predicted rapid completion, but the task ultimately stalled for 28 days and required significant rollback work.

Context: In that case, there is a very long stack trace in the original report, and this part had been cut off by the preprocess pipeline due to the token limit and noise removal.

Interpretation: This failure illustrates a tension in the text pipeline: aggressive normalization may protect the encoder from irrelevant code noise, but it can also remove rare domain-specific cues that are genuinely diagnostic. So the problem here is not only the failure of the model, but also the interaction between preprocessing and representation.

7.3 Illustrative Deployment Scenario

The following illustrative deployment scenario demonstrates how model output and attention visualization could support project-management decision making.

7.3.1 Persona Definition and Workflow Scenario

Persona: “Alex” is a highly skilled Scrum Master who oversees a backend engineering team of 8 people.

Scenario: It is day 12 of a 14-day sprint. The burndown chart looks healthy, but Alex wants to identify tasks whose text and time patterns indicate hidden carry-over risk.

7.3.2 Phase 1: Proactive Triaging

Alex opens the forecast dashboard. Tasks that appear routine in the traditional tracking view are assigned a higher derived delay-risk score. The function of this interface is not only to render score values. It also highlights substantive technical comments from several days earlier that mentioned cache-layer failure modes.

7.3.3 Phase 2: Evidence-Guided Follow-Up

This scenario illustrates one possible intervention path:

1. Alex points out the technical problem at stand-up.
2. The team re-evaluates whether the current assignee has the necessary subsystem expertise.
3. Work is re-scoped or paired earlier than it otherwise would have been.

The value of the model in this scenario is not that it proves causation, but that it helps surface evidence worth discussing before the sprint boundary is reached.

7.3.4 Phase 3: Backlog Review

The same interface can be imagined during backlog grooming. If a task is labeled "Minor" yet its description and comment history resemble previously delayed tasks, the dashboard may prompt additional review of story points, hidden dependencies, or compliance requirements before the item is committed to a sprint.

7.3.5 Illustrative Stakeholder Perspectives

From a Scrum Master perspective, a system of this kind could support earlier challenge to optimistic task framing by surfacing technically salient comments that might otherwise remain buried in long threads.

From a junior developer perspective, the highlighted past comment could provide missing context for a given task if some crucial constraints were discussed previously in the ticket's life cycle.

From an engineering-management perspective, the main potential value is earlier identification of structurally risky tasks. Even in that setting, however, the model should be treated as a decision-support tool rather than as an automatic staffing or escalation authority.

7.4 Organizational Implications and Use Constraints

The experimental results suggest that short-horizon task prediction may have organizational value, but that value depends heavily on how the output is incorporated into decision processes. From an organizational perspective, the core issue is not only whether the model can generate numbers. The key is whether that output can be used responsibly without distorting job allocation, accountability, or explanations of developer behavior.

7.4.1 Use as Triage Support Rather Than Decision Automation

The most reliable use case is classification support. In this role, the model presents the tasks that require additional human attention in the current state. The output may prompt questions such as "Are there any unresolved technical obstacles here?" or "Does this task still look safe relative to the sprint scope?" Without background checks, it is much less defensible as an automatic trigger for personnel evaluation, reassignment, or management escalation.

This distinction is important because the model is based on some organizational evidence. Issue tracker captures part of the workflow, not all of it. It may retain the technical clues that the team ignores in long threads, but will not observe offline design discussions, informal assistance, corridor coordination or hidden implementation progress. For that reason, an assistive deployment philosophy is easier to defend than an authoritative one.

7.4.2 Threshold Selection as a Governance Choice

If an organization is to implement this model, intervention thresholds should be considered a governance choice rather than a purely technical choice. A lower threshold exposes more potentially risky tasks, increasing recall at the expense of more false positives. A higher threshold will reduce the review burden, but may also miss some emerging delays. Since teams have different tolerances for audit overhead, threshold selection should be handled as a policy parameter to adjust the local balance between risk of missed and unnecessary audits.

That policy choice should also be revisited over time. A threshold calibrated on one project or one phase of development may become inappropriate if the team changes workflow, tooling, sprint cadence, or documentation norms. For that reason, the dissertation's calibration analysis is relevant not only as an academic exercise but also as a reminder that probability interpretation is contingent on the evaluation distribution.

7.4.3 Human Oversight and Interpretive Limits

Human oversight remains necessary even when the probability output is accurate on average. A model may highlight a task because of historically meaningful lexical patterns while still misunderstanding sarcasm, incomplete context, or process-specific administrative conventions. The qualitative case studies in this chapter illustrate exactly those failure modes. In practice, a useful interface would need to present both the risk estimate and the relevant textual evidence so that a human reviewer can decide whether the flag is substantively meaningful [56, 57].

This point is especially important for managerial fairness. Because developer identity features were deliberately excluded, the model is not designed to infer per-developer productivity. Even so, a risk score attached to a task can easily be overinterpreted as a judgment about the assignee if governance is weak. Predictions should therefore attach to task state and coordination needs, not to personal evaluation.

7.4.4 Adoption Costs and Data Preconditions

Even high-performance research prototypes can incur adoption costs. This model requires sufficient historical task data for calibration, sufficient issue-tracker discipline for text and metadata to be informative, and some operational workflow for interpreting risk outputs. Teams with sparse task history, inconsistent priority use, or heavily dependent on off-platform communication may not be able to achieve the same advantages observed in ASF data.

These prerequisites are part of this dissertation's broader caution about deployment. The reported results support the potential value of decision support under the evaluated conditions, but they do not indicate that any organization can install the model and receive reliable advice immediately. Effective use may require local verification, threshold setting, protocol review, and regular retraining or adjustment as project language and workflow practices evolve.

7.5 Threats to Validity

Despite the encouraging results, some threats to effectiveness limit the power of any broader generalization.

7.5.1 Construct Validity

1. **Label Ambiguity in Agile Tracking:** The goal variable defines task success as reaching any final "end" or "done" status within a 7-day window. But in reality, the word "done" is a management state whose meaning may vary from team to team, and may not exactly align with persistent functionality.

2. **Story-Point Interpretation:** Story points are used as an input feature, but they are not an absolute measure of time and may change with the team or over time.

3. **Developer Identity Exclusion:** This model excludes assignee identification and does not embed persistent comment author identifiers. This option reduces ethical

concerns and leakage risks, but it also places socially mediated productivity differences outside the scope of current models.

4. **Task-Duration Coverage:** The dissertation centers its main claims on completion within 7 days. That prediction target is operationally meaningful for sprint-level coordination in the ASF Jira setting, but model effectiveness still varies across tasks with different realized lifecycles. The duration-stratified analysis shows the strongest performance for tasks completed between 3 and 7 days, which is the task-duration regime most closely aligned with the target window. Tasks completed within 3 days are harder because they often provide only sparse issue-tracker evidence before closure, while long-running tasks remain difficult because their histories are noisier and less stable. This means the framework should be treated as duration-sensitive and still requires local validation when an organization has a substantially different mix of task durations.

7.5.2 Internal Validity

1. **Correlation vs. Causation in Communication:** Comment frequency may rise toward the end of a task, creating a risk that the model learns a lifecycle proxy rather than a signal of substantive difficulty.

2. **Temporal Leakage Risk:** A separate 10-fold training-partition cross-validation procedure was used for statistical testing, and a 5-fold project-level cross-validation procedure was used for robustness. Even with project-level controls, temporally mixed resampling can still expose the model to vocabulary and workflow patterns that would not have been available in a strictly forward-looking deployment.

7.5.3 Statistical Conclusion Validity

1. **Small Effective Sample Sizes in Some Comparisons:** Although the snapshot-level dataset is large, some inferential procedures operate on a much smaller number of

project- or fold-level observations. This means that uncertainty in effect-size and significance estimates should be interpreted at the level of the evaluation protocol actually used, not at the raw snapshot count.

2. **Dependence on the Chosen Resampling Design:** The Wilcoxon tests are informative only for the paired scores produced by the 10-fold procedure on the 30-project training partition described in Chapter 5; they are not significance tests on the 7-project held-out test set. Different project partitions or different time windows may produce different statistical conclusions, especially when comparing stronger baselines.

3. **Metric-Specific Sensitivity:** The F1 score fits the class balance and operational framing studied here, but it emphasizes the trade-off between precision and recall. Therefore, the conclusion about model superiority is strongest for the reported metric suite and should not be automatically extended to every possible decision-making criterion.

7.5.4 External Validity

1. **Open Source vs. Proprietary Communication:** The dataset comes from the Apache Software Foundation, whose communication norms may differ significantly from those of proprietary teams.

2. **Communication Outside Jira:** Many organizations rely on Slack, Teams, code-review systems, or meetings for critical coordination. These signals are invisible to current models.

3. **Cold-Start and Adaptation Limits:** The model relies on project-specific distributions and language usage. In a brand-new project with little history, these reference distributions may be unstable, which limits immediate transferability.

For these reasons, the current work supports the model as a reliable research prototype for decision support within the reported evaluation setting. Organizations considering practical use would still require domain adaptation, temporal validation, local threshold calibration, and policy constraints around how predictions may inform managerial action.

Chapter 8 - Conclusion and Future Work

This dissertation addressed short-horizon software-task completion prediction in agile settings with heterogeneous data, dynamic textual context, and uneven comment frequency. Its main contribution is a context-aware representation model that combines dynamic comments, static task text, and structured metadata in one predictive model.

8.1 Summary of Research Outcomes

The empirical results address the RQs posed at the start of the dissertation:

- **Addressing Temporal Sparsity (RQ1):** The proposed time-aware attention mechanism improved performance over index-based positional baselines, suggesting that explicit modeling of irregular update timing is useful for this prediction task.
- **Architectural Fusion Strategy (RQ2):** The hierarchical multichannel design performed better than flatter fusion strategies under the reported evaluation settings, indicating that preserving modality-specific processing before cross-feature interaction is beneficial in this setting.
- **Language-Model-Based Baselines (RQ3):** Under the reported experimental settings, within the ASF dataset family, and for the seven-day prediction horizon studied here, the proposed architecture outperformed the fine-tuned RoBERTa baseline across F1-score, ROC-AUC, and PR-AUC while remaining substantially smaller at the task-specific classifier level. When measured end-to-end, including on-the-fly SBERT embedding generation, the full pipeline required 112.4 million parameters and 18.6 ms per task. When SBERT embeddings were precomputed, the task-specific classifier contained 4.8 million parameters and required 8.3 ms per task. These results suggest that the architecture is compatible with asynchronous decision-support workflows under the reported setup.

- **Qualitative Interpretability (RQ4):** Attention visualizations and case analyses provided qualitative evidence that the model often emphasized semantically important comments and temporal patterns. These analyses are useful for interpretation, but they should be treated as explanatory aids rather than as proof of the model's internal causal reasoning.

In general, this dissertation provides evidence that software-task prediction benefits from modeling textual, structured, and temporal information jointly. Within the reported baselines, dataset construction protocol, ASF dataset family, and seven-day horizon, the proposed framework delivered the strongest average performance among the evaluated architectures and did so with a favorable efficiency profile that strengthens its case as a practical asynchronous decision-support model. Statistical testing on the 30-project training partition further supports the observed gains under the reported resampling design, including a medium-effect comparison against RoBERTa after Holm-Bonferroni correction. Duration-stratified analysis further shows that the architecture reaches its strongest duration-specific performance on tasks completed between 3 and 7 days, while remaining strongest among the evaluated models when tasks are evidence-sparse or long-running. Those Wilcoxon significance results should still be interpreted in their proper frame because they are based on the 10-fold procedure on the 30-project training partition, not on the 7-project held-out test set. At the same time, the discussion chapter and the secondary GitHub validation make clear that domain adaptation, temporal validation, and local calibration remain necessary before use in a new production environment.

8.2 Avenues for Future Work

Several directions naturally emerged from the current research.

8.2.1 Ingestion of Direct Code Signals

The current architecture relies on metadata and human-generated textual narrative surrounding a task. A natural extension is to incorporate code-level telemetry, such as code churn, complexity deltas, or commit volatility, so that the prediction model can draw on both conversational evidence and direct implementation signals.

8.2.2 Modeling Developer Social Topologies via Graph Neural Networks

Another natural extension is to model developer collaboration structure directly, allowing the predictor to incorporate interaction topology alongside textual and task-state evidence [53].

8.2.3 Short- and Long-Duration Task Applicability

The present dissertation centered its main task on a fixed horizon of $H = 7$ days because that window aligns naturally with sprint-level coordination and short-horizon intervention. The duration-stratified results indicate that this fixed-horizon model is most closely aligned with tasks that complete near the 7-day window, while very short tasks and tasks that remain open for more than 21 days present different sources of difficulty. Future work should therefore study duration-aware evaluation and adaptation strategies that preserve strong performance near the intervention window while improving reliability on quickly resolved tasks with sparse evidence and long-running work items with stale comments, interrupted activity, and shifting priorities.

8.2.4 Pragmatic Tone and Speaker-Aware Modeling

The current sentence-embedding pipeline also remains weak in the presence of sarcasm, irony, and other forms of non-literal language. A useful architectural extension would be to augment the dynamic comment channel with an auxiliary pragmatic-tone classifier trained to detect sarcasm, hedging, and polarity reversal, while pairing it with an issue-local speaker-context encoder that represents author role, participation depth, and consistency of

prior comments within the same thread. Those pragmatic and speaker-aware signals could then gate the time-aware attention mechanism, allowing the model to discount lexically dramatic but pragmatically weak comments without turning the system into a persistent developer-profiling model.

In summary, the dissertation establishes context-aware multimodal modeling as a reliable research prototype for short-term software task prediction within the reported ASF-centered evaluation setting. The final performance claim should be read across F1-score, ROC-AUC, and PR-AUC, with PR-AUC especially relevant because the completion label is moderately imbalanced. The added duration-stratified results also clarify the operational boundary of the method: the architecture is strongest when task duration is close to the 7-day target window, while very short tasks are limited by sparse evidence and long-running tasks are limited by noisy histories. The main contribution is therefore a stronger methodological and empirical basis for agile decision support, with the practical boundary that deployment in new organizational contexts still requires local validation, threshold calibration, and domain adaptation.

Appendix A - Standard GRU Formulation

As mentioned in Chapter 4, a standard Gated Recurrent Unit (GRU) is employed for processing comment sequences. For completeness, the standard transition equations omitted from the main text for conciseness are provided here. Given an input x_t at time step t and the previous hidden state h_{t-1}, the GRU computes the new hidden state h_t using the following update gate z_t, reset gate r_t, and candidate activation $\tilde{h}_t$:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{A.1}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{A.2}$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t]) \tag{A.3}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{A.4}$$

where σ denotes the sigmoid activation function, $\cdot$ denotes matrix multiplication, and $*$ denotes element-wise multiplication.

Bibliography

[1] Koudriachov C, Tam C, Aparicio M (2025) Success with Agile Project Management: Looking back and into the future. *Journal of Systems and Software* 226:112428. https://doi.org/10.1016/j.jss.2025.112428

[2] Rosa W, Jardine S (2025) A novel method to early agile effort estimation through functional initiatives. *Journal of Systems and Software* 223:112302. https://doi.org/10.1016/j.jss.2024.112302

[3] Teslyuk V, Batyuk A, Voityshyn V (2022) Method of Software Development Project Duration Estimation for Scrum Teams with Differentiated Specializations. *Systems* 10:123. https://doi.org/10.3390/systems10040123

[4] Schwaber K, Sutherland J (2020) The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. Scrum Guides. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

[5] Project Management Institute (2024) Pulse of the Profession 2024: The Future of Project Work: Moving Past Office-Centric Models. Project Management Institute. https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pmi-pulse-of-the-profession-2024-report.pdf

[6] Project Management Institute (2018) Pulse of the Profession 2018: Success in Disruptive Times. Project Management Institute. https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2018.pdf/

[7] Yürüm OR, Ünlü H, Demirörs O (2026) An alternative software benchmarking dataset: effort estimation with machine learning. *Journal of Systems and Software* 231:112591. https://doi.org/10.1016/j.jss.2025.112591

[8] Stradowski S, Madeyski L (2023) Machine learning in software defect prediction: A business-driven systematic mapping study. *Information and Software Technology* 155:107128. https://doi.org/10.1016/j.infsof.2022.107128

[9] Kula E, Greuter E, van Deursen A, et al. (2023) Dynamic Prediction of Delays in Software Projects Using Delay Patterns and Bayesian Modeling. In: *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23)*. Association for Computing Machinery, New York, NY, USA, pp 1012-1023. https://doi.org/10.1145/3611643.3616328

[10] Wang W, Chen J, Yang L, et al. (2023) Understanding and predicting incident mitigation time. *Information and Software Technology* 155:107119. https://doi.org/10.1016/j.infsof.2022.107119

[11] Bomström H, Kelanti M, Annanperä E, et al. (2023) Information needs and presentation in agile software development. *Information and Software Technology* 162:107265. https://doi.org/10.1016/j.infsof.2023.107265

[12] Abdellatif TM, Capretz LF, Ho D (2019) Automatic recall of software lessons learned for software project managers. *Information and Software Technology* 115:44-57. https://doi.org/10.1016/j.infsof.2019.07.006

[13] Alsaadi B, Saeedi K (2024) Ensemble effort estimation for novice agile teams. *Information and Software Technology* 170:107447. https://doi.org/10.1016/j.infsof.2024.107447

[14] Alpes da Silva KO, Ferreira Lima RM, Botelho da Silva V (2025) Process mining for agile software process assessment and improvement. *Information and Software Technology* 181:107680. https://doi.org/10.1016/j.infsof.2025.107680

[15] Ünlü H, Kennouche DE, Soylu GK, et al. (2024) Microservice-based projects in agile world: A structured interview. *Information and Software Technology* 165:107334. https://doi.org/10.1016/j.infsof.2023.107334

[16] Carbonera CE, Farias K, Bischoff V (2020) Software development effort estimation: A systematic mapping study. *IET Software* 14:328-344. https://doi.org/10.1049/iet-sen.2018.5334

[17] Müller R, Locatelli G, Holzmann V, et al. (2024) Artificial Intelligence and Project Management: Empirical Overview, State of the Art, and Guidelines for Future Research. *Project Management Journal* 55:9-15. https://doi.org/10.1177/87569728231225198

[18] Rahman M, Roy PP, Ali M, et al. (2023) Software Effort Estimation using Machine Learning Technique. *International Journal of Advanced Computer Science and Applications* 14:822-827. https://doi.org/10.14569/IJACSA.2023.0140491

[19] Cabral JTHA, Oliveira ALI, da Silva FQB (2023) Ensemble Effort Estimation: An updated and extended systematic literature review. *Journal of Systems and Software* 195:111542. https://doi.org/10.1016/j.jss.2022.111542

[20] Rosa W, Jardine S (2023) Data-driven agile software cost estimation models for DHS and DoD. *Journal of Systems and Software* 203:111739. https://doi.org/10.1016/j.jss.2023.111739

[21] Giray G, Bennin KE, Köksal Ö, et al. (2023) On the use of deep learning in software defect prediction. *Journal of Systems and Software* 195:111537. https://doi.org/10.1016/j.jss.2022.111537

[22] Rossi BB, Fontoura LM (2025) AI-Based Approaches for Software Tasks Effort Estimation: A Systematic Review of Methods and Trends. In: *Proceedings of the 27th*

International Conference on Enterprise Information Systems (ICEIS). vol 2, pp 144-151. https://doi.org/10.5220/0013218200003929

[23] Ardimento P, Capuzzimati M, Casalino G, et al. (2025) A novel LLM-based classifier for predicting bug-fixing time in Bug Tracking Systems. *Journal of Systems and Software* 230:112569. https://doi.org/10.1016/j.jss.2025.112569

[24] Zhou X, Liang P, Zhang B, et al. (2025) Exploring the problems, their causes and solutions of AI pair programming: A study on GitHub and Stack Overflow. *Journal of Systems and Software* 219:112204. https://doi.org/10.1016/j.jss.2024.112204

[25] Younas W, Chen R, Zhao J, et al. (2025) SPERT: Reinforcement Learning-Enhanced Transformer Model for Agile Story Point Estimation. *International Journal of Software Engineering and Knowledge Engineering* 35:293-325. https://doi.org/10.1142/S0218194025500044

[26] Vaswani A, Shazeer N, Parmar N, et al. (2017) Attention Is All You Need. In: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, pp 5998-6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[27] Sousa AO, Veloso DT, Gonçalves HM, et al. (2023) Applying Machine Learning to Estimate the Effort and Duration of Individual Tasks in Software Projects. *IEEE Access* 11:89933-89946. https://doi.org/10.1109/ACCESS.2023.3307310

[28] Uysal MP, Akturk E (2025) A Systemic Approach to Machine Learning Project Management. *IEEE Engineering Management Review* 53:87-108. https://doi.org/10.1109/EMR.2024.3503677

[29] De Bortoli Fávero EM, Casanova D, Pimentel AR (2022) SE3M: A model for software effort estimation using pre-trained embedding models. *Information and Software Technology* 147:106886. https://doi.org/10.1016/j.infsof.2022.106886

[30] Lishner I, Shtub A (2022) Using an Artificial Neural Network for Improving the Prediction of Project Duration. *Mathematics* 10:4189. https://doi.org/10.3390/math10224189

[31] Noyori Y, Washizaki H, Fukazawa Y, et al. (2023) Deep learning and gradient-based extraction of bug report features related to bug fixing time. *Frontiers in Computer Science* 5:1032440. https://doi.org/10.3389/fcomp.2023.1032440

[32] Aldndni W, Meng N, Servant F (2023) Automatic prediction of developers' resolutions for software merge conflicts. *Journal of Systems and Software* 206:111836. https://doi.org/10.1016/j.jss.2023.111836

[33] Gomes L, da Silva Torres R, Côrtes ML (2023) BERT- and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS: A comparative study. *Information and Software Technology* 160:107217. https://doi.org/10.1016/j.infsof.2023.107217

[34] Moradi Dakhel A, Nikanjam A, Majdinasab V, et al. (2024) Effective test generation using pre-trained Large Language Models and mutation testing. *Information and Software Technology* 171:107468. https://doi.org/10.1016/j.infsof.2024.107468

[35] Sergeyuk A, Golubev Y, Bryksin T, et al. (2025) Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology* 178:107610. https://doi.org/10.1016/j.infsof.2024.107610

[36] Ling Y, Yu S, Fang C, et al. (2025) Redefining crowdsourced test report prioritization: An innovative approach with large language model. *Information and Software Technology* 179:107629. https://doi.org/10.1016/j.infsof.2024.107629

[37] Qu Y, Huang S, Nie P (2025) A review of backdoor attacks and defenses in code large language models: Implications for security measures. *Information and Software Technology* 182:107707. https://doi.org/10.1016/j.infsof.2025.107707

[38] Kim DK, Ming H (2025) Assessing output reliability and similarity of large language models in software development: A comparative case study approach. *Information and Software Technology* 185:107787. https://doi.org/10.1016/j.infsof.2025.107787

[39] Tang M, Li T, Gao W, et al. (2023) AttenSy-SNER: software knowledge entity extraction with syntactic features and semantic augmentation information. *Complex & Intelligent Systems* 9:25-39. https://doi.org/10.1007/s40747-022-00742-5

[40] Harish Kumar K, Srinivas K (2024) An improved analogy-rule based software effort estimation using HTRR-RNN in software project management. *Expert Systems with Applications* 251:124107. https://doi.org/10.1016/j.eswa.2024.124107

[41] Du J, Ren X, Li H, et al. (2022) Prediction of bug-fixing time based on distinguishable sequences fusion in open source software. *Journal of Software: Evolution and Process* 35:e2443. https://doi.org/10.1002/smr.2443

[42] Zou Y, Wang H (2025) A three-stage cross-project defect prediction framework based on feature representation and knowledge transfer. *Complex & Intelligent Systems* 11:459. https://doi.org/10.1007/s40747-025-02098-y

[43] Deng L, Wang Y, Wang H, et al. (2024) Time-Aware Attention-Based Transformer (TAAT) for Cloud Computing System Failure Prediction. In: *Proceedings of the 30th*

ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24). Association for Computing Machinery, New York, NY, USA, pp 4906-4917. https://doi.org/10.1145/3637528.3671547

[44] Dai R, Wang Z, Jie J, et al. (2025) VTformer: a novel multiscale linear transformer forecaster with variate-temporal dependency for multivariate time series. *Complex & Intelligent Systems* 11:6335-6353. https://doi.org/10.1007/s40747-025-01866-0

[45] Zhang J, Li X, Tian J (2023) An integrated multi-head dual sparse self-attention network for remaining useful life prediction. *Reliability Engineering & System Safety* 233:109096. https://doi.org/10.1016/j.ress.2023.109096

[46] Zhao K, Jia Z, Jia F, et al. (2023) Multi-scale integrated deep self-attention network for predicting remaining useful life of aero-engine. *Engineering Applications of Artificial Intelligence* 120:105860. https://doi.org/10.1016/j.engappai.2023.105860

[47] Feng Z, Guo D, Tang D, et al. (2020) CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp 1536-1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[48] Odu O, Belle AB, Wang S, et al. (2025) Automatic instantiation of assurance cases from patterns using large language models. *Journal of Systems and Software* 222:112353. https://doi.org/10.1016/j.jss.2025.112353

[49] 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 and the 9th International Joint Conference on Natural*

Language Processing (EMNLP-IJCNLP). Hong Kong, China, pp 3980-3990. https://doi.org/10.18653/v1/D19-1410

[50] Zhao C, Li F, Peng Z, et al. (2023) A structured multi-head attention prediction method based on heterogeneous financial data. *PeerJ Computer Science* 9:e1653. https://doi.org/10.7717/peerj-cs.1653

[51] Pham NQ, Nguyen TS, Niehues J, et al. (2019) Very Deep Self-Attention Networks for End-to-End Speech Recognition. In: *Proceedings of Interspeech 2019*. Graz, Austria, pp 61-65. https://doi.org/10.21437/Interspeech.2019-2702

[52] Hu Z, Liu G, Li Y, et al. (2024) SAGB: self-attention with gate and BiGRU network for intrusion detection. *Complex & Intelligent Systems* 10:8467-8479. https://doi.org/10.1007/s40747-024-01577-y

[53] Wang Z, Tong W, Li P, et al. (2023) BugPre: an intelligent software version-to-version bug prediction system using graph convolutional neural networks. *Complex & Intelligent Systems* 9:3835-3855. https://doi.org/10.1007/s40747-022-00848-w

[54] Liu X, Qi H, Jia S, et al. (2025) Recent Advances in Optimization Methods for Machine Learning: A Systematic Review. *Mathematics* 13:2210. https://doi.org/10.3390/math13132210

[55] Sun RY (2020) Optimization for deep learning: An overview. *Journal of the Operations Research Society of China* 8:249-294. https://doi.org/10.1007/s40305-020-00309-6

[56] Rahimi M, Vierhauser M (2022) Visualization of aggregated information to support class-level software evolution. *Journal of Systems and Software* 192:111421. https://doi.org/10.1016/j.jss.2022.111421

[57] Santos JI, Pereda M, Ahedo V, et al. (2023) Explainable machine learning for project management control. *Computers & Industrial Engineering* 180:109261. https://doi.org/10.1016/j.cie.2023.109261

[58] Ortu M, Destefanis G, Adams B, et al. (2015) The JIRA Repository Dataset. In: *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE '15)*. Association for Computing Machinery, New York, NY, USA, pp 1-4. https://doi.org/10.1145/2810146.2810147

[59] Chawla NV, Bowyer KW, Hall LO, et al. (2002) SMOTE: Synthetic Minority Oversampling Technique. *Journal of Artificial Intelligence Research* 16:321-357. https://doi.org/10.1613/jair.953

[60] Jørgensen M (2004) A review of studies on expert estimation of software development effort. *Journal of Systems and Software* 70:37-60. https://doi.org/10.1016/S0164-1212(02)00156-5

[61] de Melo A, Sanchez A (2008) Software maintenance project delays prediction using Bayesian Networks. *Expert Systems with Applications* 34:908-919. https://doi.org/10.1016/j.eswa.2006.10.040

[62] Vandevoorde S, Vanhoucke M (2006) A comparison of different project duration forecasting methods using earned value metrics. *International Journal of Project Management* 24:289-302. https://doi.org/10.1016/j.ijproman.2005.10.004

[63] Mariano S, Awazu Y (2024) Managing large-scale projects: Unpacking the role of project memory. *International Journal of Project Management* 42:102573. https://doi.org/10.1016/j.ijproman.2024.102573

[64] Zhang X, Shen W, Tang W, et al. (2024) How to improve the effects of knowledge governance on individual learning across projects: From the perspective of re-

lational capital. *International Journal of Project Management* 42:102562. https://doi.org/10.1016/j.ijproman.2024.102562

[65] Zhang YF (2024) Too little or too much transparency? Influencing accountability in project management through the practices of using digital technologies. *International Journal of Project Management* 42:102636. https://doi.org/10.1016/j.ijproman.2024.102636

[66] Olsson T, Artto K, Hauhia A, et al. (2024) When project outcomes matter: Organizational integration in managing long-term target benefits. *International Journal of Project Management* 42:102648. https://doi.org/10.1016/j.ijproman.2024.102648

[67] McGee S, Greer D (2012) Towards an understanding of the causes and effects of software requirements change: two case studies. *Requirements Engineering* 17:133-155. https://doi.org/10.1007/s00766-012-0149-0

[68] Dragicevic S, Celar S, Turic M (2017) Bayesian network model for task effort estimation in agile software development. *Journal of Systems and Software* 127:109-119. https://doi.org/10.1016/j.jss.2017.01.027

[69] Usman M, Petersen K, Börstler J, et al. (2018) Developing and using checklists to improve software effort estimation: A multi-case study. *Journal of Systems and Software* 146:286-309. https://doi.org/10.1016/j.jss.2018.09.054

[70] Conoscenti M, Besner V, Vetrò A, et al. (2019) Combining data analytics and developers feedback for identifying reasons of inaccurate estimations in agile software development. *Journal of Systems and Software* 156:126-135. https://doi.org/10.1016/j.jss.2019.06.075

[71] Jørgensen M, Halkjelsvik T (2020) Sequence effects in the estimation of software development effort. *Journal of Systems and Software* 159:110448. https://doi.org/10.1016/j.jss.2019.110448

[72] Franch X, Palomares C, Quer C, et al. (2023) The state-of-practice in requirements specification: an extended interview study at 12 companies. *Requirements Engineering* 28:377-409. https://doi.org/10.1007/s00766-023-00399-7

[73] Frattini J, Fischbach J, Mendez D, et al. (2023) Causality in requirements artifacts: prevalence, detection, and impact. *Requirements Engineering* 28:49-74. https://doi.org/10.1007/s00766-022-00371-x

[74] Abbas M, Ferrari A, Shatnawi A, et al. (2023) On the relationship between similar requirements and similar software. *Requirements Engineering* 28:23-47. https://doi.org/10.1007/s00766-021-00370-4

[75] Lüders CM, Pietz T, Maalej W (2023) On understanding and predicting issue links. *Requirements Engineering* 28:541-565. https://doi.org/10.1007/s00766-023-00406-x

[76] Ünsal-Altuncan I, Vanhoucke M (2024) A hybrid forecasting model to predict the duration and cost performance of projects with Bayesian Networks. *European Journal of Operational Research* 315:511-527. https://doi.org/10.1016/j.ejor.2023.12.029

[77] Weinzierl S, Zilker S, Liessmann A, et al. (2025) From Source to Target: Leveraging Transfer Learning for Predictive Process Monitoring in Organizations. *Business & Information Systems Engineering*. Advance online publication. https://doi.org/10.1007/s12599-025-00969-9

[78] Stray V, Moe NB (2020) Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and Slack. *Journal of Systems and Software* 170:110717. https://doi.org/10.1016/j.jss.2020.110717

[79] Dingsøyr T, Bjørnson FO, Schrof J, et al. (2023) A longitudinal explanatory case study of coordination in a very large development programme: the impact of transitioning from a first- to a second-generation large-scale agile development method. *Empirical Software Engineering* 28:1. https://doi.org/10.1007/s10664-022-10230-6

[80] Menzies T, Chen Z, Hihn J, et al. (2006) Selecting Best Practices for Effort Estimation. *IEEE Transactions on Software Engineering* 32:883-895. https://doi.org/10.1109/TSE.2006.114

[81] Pospieszny P, Czarnacka-Chrobot B, Kobylinski A (2018) An effective approach for software project effort and duration estimation with machine learning algorithms. *Journal of Systems and Software* 137:184-196. https://doi.org/10.1016/j.jss.2017.11.066

[82] Kocaguneli E, Menzies T, Keung JW (2012) On the Value of Ensemble Effort Estimation. *IEEE Transactions on Software Engineering* 38:1403-1416. https://doi.org/10.1109/TSE.2011.111

[83] Choetkiertikul M, Dam HK, Tran T, et al. (2019) A Deep Learning Model for Estimating Story Points. *IEEE Transactions on Software Engineering* 45:637-656. https://doi.org/10.1109/TSE.2018.2792473

[84] Porru S, Murgia A, Demeyer S, et al. (2016) Estimating Story Points from Issue Reports. In: *Proceedings of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE '16)*. Association for Computing

Machinery, New York, NY, USA, pp 1-10. https://doi.org/10.1145/2972958.2972959

[85] Pascanu R, Mikolov T, Bengio Y (2013) On the Difficulty of Training Recurrent Neural Networks. In: *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, PMLR 28:1310-1318. https://proceedings.mlr.press/v28/pascanu13.html

[86] Pennington J, Socher R, Manning CD (2014) GloVe: Global Vectors for Word Representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp 1532-1543. https://doi.org/10.3115/v1/D14-1162

[87] Devlin J, Chang MW, Lee K, et al. (2019) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp 4171-4186. https://aclanthology.org/N19-1423/

[88] Arcuri A, Briand L (2011) A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In: *Proceedings of the 33rd International Conference on Software Engineering*, pp 1-10. https://doi.org/10.1145/1985793.1985795

[89] Lin TY, Goyal P, Girshick R, et al. (2017) Focal Loss for Dense Object Detection. In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp 2999-3007. https://doi.org/10.1109/ICCV.2017.324

[90] Guo C, Pleiss G, Sun Y, et al. (2017) On Calibration of Modern Neural Networks. In: *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, PMLR 70:1321-1330. https://proceedings.mlr.press/v70/guo17a.html

[91] Chen T, Guestrin C (2016) XGBoost: A Scalable Tree Boosting System. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp 785-794. https://doi.org/10.1145/2939672.2939785

[92] Sherstinsky A (2020) Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena* 404:132306. https://doi.org/10.1016/j.physd.2019.132306

[93] Karampatsis RM, Babii H, Robbes R, et al. (2020) Big code != big vocabulary. In: *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pp 1073-1085. https://doi.org/10.1145/3377811.3380342

[94] Zheng W, Zhang M, Tang H, et al. (2021) Automatically Identifying Bug Reports with Tactical Vulnerabilities by Deep Feature Learning. In: *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pp 333-344. https://doi.org/10.1109/ISSRE52982.2021.00043