Machine Learning for Predicting Inner Source Viability

BACHELOR THESIS

Juri Kembügler

Submitted on 16 March 2026

Friedrich-Alexander-Universität Erlangen-Nürnberg
Faculty of Engineering, Department Computer Science
Professorship for Open Source Software

Supervisor:
Julian Hirsch, M.Sc.
Prof. Dr. Dirk Riehle, M.B.A.

Declaration of Originality

I confirm that the submitted thesis is original work and was written by me without further assistance. Appropriate credit has been given where reference has been made to the work of others. The thesis was not examined before, nor has it been published. The submitted electronic version of the thesis matches the printed version.

Erlangen, 16 March 2026

License

Erlangen, 16 March 2026

Abstract

The adoption of inner source software development practices has grown substantially in recent years, yet organizations continue to lack data-driven instruments for assessing the viability of inner source initiatives prior to resource commitment. This thesis addresses that gap by designing, implementing, and evaluating a machine learning-based system for predicting inner source project viability. Following the Design Science Research Methodology, the artifact operationalizes the inner source viability framework of Hirsch and Riehle (2022) into a continuous scoring algorithm, combines repository-derived GitHub signals with user-provided organizational inputs, and trains a LightGBM regressor alongside a linear regression baseline on a purpose-built synthetic dataset of 150 simulated projects. Inner source viability is forecasted via recursive multistep forecasting on unfinished projects. Results demonstrate that the system is functional in principle, while identifying real-world data acquisition as the decisive prerequisite for any production deployment.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

AI Artificial Intelligence

API Application Programming Interface

BMD Baseline Maintenance Demand

DL Deep Learning

DSRM Design Science Research Methodology

EEMS Expected External Maintenance Share

EMC Expected Maintenance Contribution

FMD Future Maintenance Demand

ISOF Inner Source Overhead Factor

JSON JavaScript Object Notation

LightGBM Light Gradient-Boosting Machine

MAE Mean Absolute Error

ML Machine Learning

OSS Open Source Software

PERT Program Evaluation and Review Technique

RMSE Root Mean Squared Error

1 Introduction

The increasing adoption of collaborative software development practices has fundamentally transformed organizational structures in the technology sector. Inner source—the application of open source development practices within organizational boundaries—enables developers to collaborate on reusable software components across company-internal silos while maintaining proprietary control (Riehle et al., 2016). By adopting mechanisms such as transparent code repositories, peer review processes, and community-driven contribution models, organizations seek to capture the benefits of open source collaboration without exposing intellectual property to external parties (Capraro & Riehle, 2016).

1.1 Motivation and Problem Context

The recent *State of InnerSource 2025* report indicates that inner source has evolved from an emerging concept into an established organizational practice. Many organizations report several years of experience and describe their initiatives as scaling or integrated into existing development processes. While motivations often focus on technical objectives such as code reuse and development speed, reported benefits extend to cross-team collaboration, knowledge sharing, and improved communication skills (Dillon, 2025). These findings suggest that inner source contributes not only to software development efficiency, but also to the formation of organizational social capital.

Despite these benefits, the sustained success of inner source initiatives is not guaranteed. Prior research emphasizes the importance of aligned incentives, structured governance, and continuous contributor engagement (Riehle et al., 2016; Stol et al., 2014). At the same time, introducing inner source may increase organizational complexity if not carefully managed (Hirsch & Riehle, 2022).

However, measurement practices for inner source remain comparatively immature. Less than half of surveyed organizations report systematic measurement, with most relying on manual tracking or surveys rather than automated metrics (Dillon, 2025). This lack of structured, data-driven monitoring makes it difficult

to detect early signs of declining engagement or structural instability. Consequently, there is a need for predictive, data-driven approaches to assessing inner source project viability.

1.2 Research Questions and Thesis Structure

This thesis follows the Design Science Research Methodology (DSRM) as described by Peffers et al. (2007), which provides a structured framework for the development and evaluation of information systems artifacts. The methodology proceeds through six phases: problem identification, objective definition, design, implementation, demonstration, and evaluation. According to this structure, three research questions are defined to guide this thesis.

The first phase of the DSRM concerns the identification of the problem and the definition of objectives. Grounded in the management accounting models for inner source proposed by Hirsch and Riehle (2022), this thesis investigates whether the viability of an inner source project can be predicted by a Machine Learning (ML) model. This motivates the first research question:

> **RQ1: What are the objectives for an inner source viability prediction system?**

This question is addressed in chapters 2 and 3, where related work is reviewed and the concrete objectives of the system are derived.

Building on these objectives, the DSRM requires the design and implementation of an artifact that addresses the identified problem. In this work, the artifact consists of three linked components: a viability scoring algorithm, a synthetic dataset, and a ML model. This leads to the second research question:

> **RQ2: How can an inner source viability prediction system be conceptually designed and implemented?**

The conceptual design of the system is presented in chapter 4, while chapter 5 documents its concrete implementation.

Finally, the DSRM requires that the developed artifact be evaluated against the previously defined objectives. This motivates the third research question:

> **RQ3: To what extent does the developed system fulfill its objectives, and what are its limitations?**

This question is answered in chapter 7 through a systematic evaluation of the implemented system. The thesis concludes in chapter 8 with a summary of contributions and directions for future work.

2 Problem Identification and Related Work

As discussed in the previous chapter, assessing the viability of inner source initiatives is an important challenge for organizations adopting collaborative development practices. This chapter reviews the relevant foundations and related work on inner source, viability assessment, and machine learning for software project prediction.

2.1 Inner Source Software Engineering

The widespread success of Open Source Software (OSS) development demonstrates that collaborative, transparent, and community-based development practices can produce high-quality software (Crowston et al., 2008), while enabling faster development cycles and more efficient reuse of software components (Ajila & Wu, 2007; Capraro & Riehle, 2016). These advantages have prompted organizations to explore how open source principles might be internalized within proprietary development environments—a phenomenon referred to in the literature as *inner source* (Capraro & Riehle, 2016; Stol et al., 2014). Rather than releasing software under a permissive license, inner source applies open source development practices exclusively within the organizational boundary (Edison et al., 2020; Stol et al., 2014). These practices include universal access to development artifacts, peer review of contributions, informal communication channels, and self-selection of tasks. In doing so, it enables cross-team collaboration on shared components without the intellectual property risks associated with public release (Hirsch & Riehle, 2022).

Crucially, however, inner source is not simply open source constrained to a corporate firewall. The organizational context introduces structural differences that alter both the dynamics of collaboration and the requirements for governance. Wan et al. (2022) empirically show that the motivational profile of inner source contributors differs significantly from that of open source contributors. While participation in open source projects is often driven by intrinsic motivations such

as fun or ideological alignment, inner source contributions are more strongly influenced by job responsibilities and organizational culture. In particular, contribution behavior depends on the practitioner's belief in the effectiveness of inner source as a development approach. This difference suggests that inner source cannot be treated as a simple transplant of open source methodology. Instead, it requires dedicated management structures, measurement instruments, and decision support tools tailored to the organizational environment.

The benefits of successful inner source adoption are well documented. Organizations that implement inner source practices can reduce duplicated development effort across organizational silos, improve software quality through community-wide peer review, and achieve better adherence to development schedules. In addition, inner source can lower long-term maintenance costs (Edison et al., 2020; Hirsch & Riehle, 2022). However, these benefits depend on a set of structural and cultural preconditions. Edison et al. (2020) identify several major challenges in their systematic literature review of 37 primary studies. These challenges include integration and architectural issues, deficits in knowledge management, cultural resistance to open collaboration, and a lack of metrics for evaluating inner source initiatives. From a management perspective, the absence of suitable metrics is particularly critical. Without such metrics, organizations often lack the means to assess whether a component or project is a suitable candidate for inner source development before committing the necessary resources.

2.2 Inner Source Viability Assessment

While the benefits of inner source adoption are well-established in the literature, the decision of whether to inner source a particular software component is non-trivial and carries both organizational and financial consequences. Hirsch and Riehle (2022) frame this decision as a *make-or-inner-source* assessment, drawing an analogy to the classical make-or-buy decision in management accounting. In this framing, independent closed development represents the baseline alternative. Inner source collaboration, by contrast, constitutes a form of internal sourcing. Its cost includes not only the contribution effort but also the overhead associated with adoption, integration, and ongoing coordination across organizational boundaries. Unlike a conventional make-or-buy decision, however, the make-or-inner-source assessment has no clear buyer-seller distinction, because all participating units contribute and benefit simultaneously. This characteristic makes the evaluation considerably more complex.

This complexity is compounded by the multidimensional nature of the factors that determine inner source viability. As Hirsch and Riehle (2022) demonstrate, a meaningful viability assessment must account for:

- **Strategic value factors:** Alignment with organizational priorities, including urgency of delivery, potential for sales growth, technical differentiation, and overall profitability.
- **Human factors:** Availability of organizational prerequisites for successful collaboration, such as sufficient resources, relevant skills, know-how, and prior inner source experience.
- **Performance factors:** Expected impact on project outcomes, including software quality, development flexibility, innovation capacity, and delivery time.
- **Financial factors:** Full cost profile of the inner source decision, e.g., contribution costs, conversion and integration costs, expected maintenance contributions, and the cost of alternative independent development.

Critically, these dimensions interact: A project may be financially viable but organizationally infeasible due to insufficient human resource availability, or strategically desirable but financially prohibitive given the expected maintenance contribution. No single factor is therefore sufficient as a basis for a viability determination.

The practitioner-level evidence from Wan et al. (2022) reinforces this multidimensionality from a different angle. Their finding indicates that the strongest predictor of sustained inner source participation is a practitioner's belief in the effectiveness of inner source, rather than financial incentives or formal job obligations. This suggests that organizational culture and managerial support provide viability-relevant signals that are not captured by purely economic models. A robust viability assessment framework must therefore integrate both quantifiable project-level features and more qualitative organizational indicators.

Despite this recognized complexity, the literature has produced few systematic tools for viability assessment prior to the initiation of an inner source project. Buchner and Riehle (2023) identify a structural divide between the business domain, which specifies what needs to be assessed, and the algorithmic domain, which determines what can be measured, and observe that these two domains have largely developed in isolation. They further find that of the algorithmic approaches surveyed, only those operating on cross-boundary code flow data, primarily derived from commit histories, are directly applicable to the inner source context without requiring significant adaptation. This constrains the practically feasible feature space for automated viability assessment and motivates a repository-centric approach to feature engineering. Edison et al. (2020) identify the lack of quantitative measurement instruments for evaluating inner source initiatives as one of the primary open challenges in the field. They therefore call for future research to define and validate such metrics in real-world settings.

Taken together, these findings motivate the central contribution of this thesis: the design and evaluation of a ML-based artifact for predicting inner source viability, drawing on repository-derived features to produce decision support at the project planning stage.

2.3 Machine Learning for Software Project Prediction

In recent years, Artificial Intelligence (AI) and ML have emerged as one of the fastest-growing fields in computer science (Maslej et al., 2025). At its core, ML focuses on algorithms that learn patterns from data to improve their performance on a given task, rather than relying on explicitly programmed rules (Jordan & Mitchell, 2015). A fundamental distinction exists between *supervised* and *unsupervised* learning: in the supervised setting, a model is trained on labeled input-output pairs to learn a mapping from features to a target variable, whereas unsupervised methods seek to uncover hidden structure in unlabeled data (Jordan & Mitchell, 2015). One important application domain within supervised learning is *time series prediction*, where the objective is to forecast future values of a variable based on its historical observations (Hall & Rasheed, 2025).

2.3.1 Relevant Machine Learning Methods

Several supervised learning methods can be applied to time series prediction. *Linear Regression* is a fundamental technique that models the target variable as a weighted linear combination of input features. Due to its interpretability and efficiency, it is commonly used as a baseline for evaluating more complex models. (Toner & Darlow, 2024)

Gradient boosting methods, such as *LightGBM*, represent a more expressive class of tree-based ensemble models. These algorithms construct an additive sequence of weak learners, typically shallow decision trees, where each successive tree corrects the residual errors of its predecessor. This sequential correction mechanism allows gradient boosting models to capture complex non-linear relationships in the data without requiring explicit feature engineering for non-linearity. (Hall & Rasheed, 2025)

A key methodological consideration in applying supervised learning to time series data is the construction of appropriate input features. Rather than relying on architectures designed specifically for sequential data, classical and tree-based methods can also be applied to time series forecasting. When combined with *lag features*—past observations of the target variable used as input features at each time step—these models often achieve competitive forecasting accuracy. In many

cases, they match or even outperform deep learning approaches while remaining more interpretable and requiring less training data. (Makridakis et al., 2018)

2.3.2 Machine Learning Applications in Software Project Assessment

The application of ML to software project data has a well-established tradition. Static code attributes mined from version control repositories constitute effective predictors in classification models, establishing the foundational principle that repository-derived metrics carry meaningful predictive signal (Menzies et al., 2007). This perspective has been extended to effort estimation, where ML methods have been shown to consistently outperform traditional algorithmic models when structured project-level features are available (Nassif et al., 2016). These two bodies of work jointly justify treating repository-derived metrics—such as lines of code, bug ratios, and pull request activity—as legitimate model inputs, as is the case in the artifact developed in this thesis.

The most structurally analogous prior work is found in OSS health prediction. Predictive models constructed from observable GitHub activity metrics have been shown to identify projects at risk of abandonment—a task closely related to the inner source viability prediction addressed in this thesis (Coelho & Valente, 2017). However, that work targets binary abandonment classification for public open source projects, whereas the present artifact predicts a continuous, inner source viability score based on the work of Hirsch and Riehle (2022). As Kalliamvakou et al. (2014) conclude, however, "The story told by mined data is not always the whole story". Therefore, the use of GitHub data for such purposes has itself been critically examined, with several limitations (Kalliamvakou et al., 2014). This motivates the inclusion of user-reported input features alongside automated repository signals in the present model.

In summary, while ML has been successfully applied to defect prediction, effort estimation, and OSS health assessment, no prior work has addressed inner source viability prediction in a time-series forecasting context. This gap motivates the design artifact presented in this thesis.

2.4 Research Gap

The preceding review reveals three converging open problems that motivate the present work:

1. **Absence of quantitative viability assessment tools.** Despite the recognized importance of pre-project viability evaluation, the inner source literature lacks data-driven instruments for assessing project candidates prior to resource commitment (Buchner & Riehle, 2023; Edison et al., 2020).
2. **Manual dependency of existing frameworks.** The management accounting model of Hirsch and Riehle (2022) provides a structured, multi-dimensional viability framework, but relies on expert judgment and manual input rather than automated predictive modeling.
3. **Absence of ML-based inner source viability prediction.** While ML has been successfully applied to related software project assessment tasks, no prior work addresses inner source viability prediction in a time-series forecasting context.

3 Objective definition

Building upon the identified problems in chapter 2, the final objectives for our predictive system wil be introduced in this chapter:

O1 – Operationalize the viability framework into a computable scoring algorithm. The lack of quantitative tools for viability assessment identified in section 2.2 is directly attributable to the absence of a deterministic evaluation function to translate the qualitative evaluation structure of the make-or-inner-source framework into a calculable target variable. Without such a function, supervised learning is not applicable. O1 therefore represents the fundamental prerequisite from which all subsequent objectives are derived.

O2 – Generate a labeled synthetic dataset reflecting inner source project timelines. As stated in section 2.2, no labeled real-world dataset of inner source viability scores exists. The scoring function produced by O1 makes labeling possible in principle, but a dataset must still be constructed. O2 addresses this by requiring the synthesis of project timelines across multiple archetypes and development phases, producing the training data without which the ML pipeline of O4 cannot be realized.

O3 – Combine GitHub-derived repository signals with user-provided organizational inputs as model features. Section 2.3 establishes that repository-derived metrics carry meaningful predictive signal, but also that mined data does not always capture the full picture of a project's health. The viability framework reviewed in section 2.2 further shows that organizational factors, such as inner source experience, resource availability, and strategic alignment, are essential to any viable assessment. O3 responds to both findings by mandating a hybrid feature space that integrates automated repository signals with manually supplied organizational parameters.

O4 – Design a ML pipeline that predicts the next-x-month viability score. The research gap identified in section 2.4 includes the complete absence of ML-based inner source viability prediction in a time-series forecasting context. O4 addresses this directly by framing the prediction task as a supervised forecasting problem in which all features are derived exclusively from observations prior to the target month. This ensures that the model's performance reflects real predictive capability rather than access to contemporaneous information.

O5 – Evaluate generalization on unseen projects. The same gap noted in section 2.4 implies that a methodologically grounded evaluation protocol is as important as the model itself. Evaluating on held-out time steps of known projects would measure extrapolation within a project, not generalization to new ones. O5 therefore requires that the evaluation be conducted exclusively on projects entirely absent from training, stratified by project archetype to ensure that every type is proportionally represented in both splits.

4 Solution Design

Having identified the research gaps (chapter 2) and derived the system objectives (chapter 3), this chapter presents the conceptual design of the artifact. The chapter is structured as follows. Section 4.1 introduces the overall pipeline and the relationships between its components. Section 4.2 presents the viability scoring framework that operationalizes the inner source viability model into a computable scoring function. Section 4.3 describes the design of the synthetic dataset required for model training, and section 4.4 presents the ML model architecture and its training procedure.

4.1 Design Overview and Artifact Pipeline

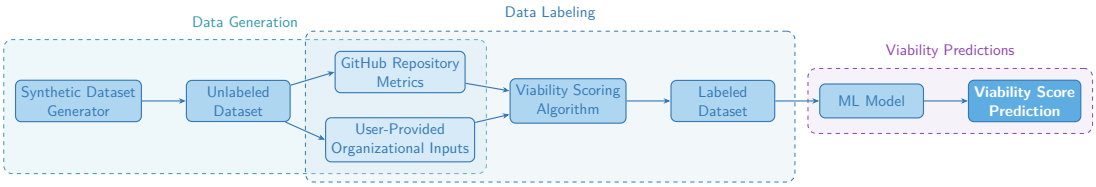

Figure 4.1: Artifact pipeline

The artifact consists of three linked components whose relationships are governed by the objectives defined in chapter 3. As required by O1, a viability scoring algorithm is designed to transform the inner source viability model of Hirsch and Riehle (2022) into a continuous, deterministic score. This algorithm accepts as input a combination of GitHub-derived repository metrics and user-provided organizational parameters, as specified in O3, and produces a `ViabilityScore`, which is later used to label the synthetic dataset.

As established by O2, no real-world labeled dataset of inner source viability scores exists. A synthetic dataset generator is therefore designed to produce project timelines of sufficient size and archetype diversity. Each generated record is subsequently labeled by the scoring algorithm, yielding the supervised training data required by O4.

Finally, the labeled dataset serves as the basis for training the ML model. The model is designed as a time-series forecasting pipeline in accordance with O4, predicting the next-x-month viability score from lag-transformed features derived exclusively from prior observations. The three components and their dependencies are illustrated in fig. 4.1.

4.2 Viability Scoring Framework

The viability scoring framework constitutes the central design artifact of this thesis. Its purpose is to produce a quantitative, multi-dimensional score that estimates the degree to which an inner source initiative is likely to succeed, given observable repository metrics and practitioner-supplied context. As already mentioned The framework is grounded in the make-or-inner-source decision model introduced by Hirsch and Riehle (2022), which organizes the relevant assessment criteria into four groups: *Strategic Value*, *Human*, *Performance*, and *Financial*. The present work operationalizes each of these factor groups as a computable sub-score, which are subsequently combined into a composite viability score as detailed in section 4.2.4.

These four factor group scores are combined pairwise into four dimensions, following the two-axis decision matrix of Hirsch and Riehle (2022). As shown in Table 4.1, each dimension reflects a distinct perspective on the viability decision by weighting the contribution of two factor groups.

Dimension	Group A	Group B
Quality Dimension	Strategic Value	Performance
Company Dimension	Strategic Value	Human
Project Dimension	Performance	Financial
Resource Dimension	Human	Financial

Table 4.1: Decision matrix of the four viability dimensions from the underlying factor groups. (Hirsch & Riehle, 2022)

Each group consists of four factors, as shown in Table 4.2. In the following sections (sections 4.2.1 to 4.2.3), we describe which GitHub-derived repository signals and practitioner-supplied questionnaire inputs are used to determine a score $s \in [0, 10]$ for each factor, or, in the case of the financial factor group, specific cost estimates across best, average, and worst-case scenarios.

Strategic Value	Human	Performance	Financial
Urgency	Human Resource Availability	Exp. Impact on Quality	Contribution Cost
Potential Sales Growth	Skills & Know-how Availability	Exp. Impact on Flexibility	Conversion & Integration Cost
Technical Differentiation	Potential Skills Development	Exp. Impact on Innovation	Exp. Maintenance Contribution
Profitability	Inner Source Experience	Exp. Impact on Delivery Time	Cost of Alternative Development

Table 4.2: Decision factors of each factor group. (Hirsch & Riehle, 2022)

4.2.1 Strategic and Human Factors

Strategic value factors capture the degree to which an inner source collaboration aligns with the strategic objectives of the participating organization. Four factors are evaluated: *Urgency*, *Potential Sales Growth*, *Technical Differentiation*, and *Profitability*. These factors resist direct quantification from repository data alone, as they reflect organizational intent and market positioning rather than observable development activity. Each factor is therefore rated from the practitioner on a rating scale of $[0, 10]$, together with a relative importance weight $w_i \in [0, 1]$ that allows the framework to reflect differing strategic priorities.

Human factors assess the organizational readiness and resource capacity required for inner source collaboration. The four constituent factors—*Human Resource Availability*, *Skills and Know-how Availability*, *Potential Skills Development*, and *Inner Source Experience*—are similarly not derivable from repository signals. Each is therefore elicited via practitioner questionnaire on a rating scale of $[0, 10]$, with equal weighting applied across all four factors.

4.2.2 Performance Factors

Performance factors quantify the expected impact of inner source adoption across four dimensions of software development quality: *Quality*, *Flexibility*, *Innovation*, and *Delivery Time*. Unlike the strategic and human factors, performance scores are derived primarily from objective repository metrics, supplemented by targeted practitioner inputs where repository data cannot fully capture business context. In the following, we describe how a score ranging from 0 to 10 is calculated for each factor and which data is required for this calculation.

Expected Impact on Quality (EIoQ)

The quality score estimates the degree to which an inner source collaboration would address existing quality deficiencies in the project. Its computation draws on two sources of input: repository-derived signals that quantify the current quality gap, and practitioner-supplied assessments that contextualize the relevance of that gap for the organization.

Practitioner Inputs. Two questionnaire items are collected:

Q1 Quality Criticality: How critical is software quality for this project? The practitioner selects one of four levels, which are mapped to a numeric modifier q_c:

- Low $\rightarrow q_c = 0.25$
- Medium $\rightarrow q_c = 0.50$
- High $\rightarrow q_c = 0.75$
- Business-critical $\rightarrow q_c = 1.00$

Q2 Review Openness: Are external code reviews welcomed and realistic within the team? The practitioner provides a rating $q_r \in [1, 10]$, which is normalized to $[0.1, 1.0]$ prior to computation.

Repository-Derived Signals. The repository-side quality gap is decomposed into four sub-metrics, each normalized to $[0, 1]$. Table 4.3 lists the GitHub signals used and their definitions.

Signal	Definition
bug_ratio	Share of issues labelled as bugs over total issues
bug_fix_time	Average time from bug open to close, in days
reopened_ratio	Share of issues that were reopened after closing
hotfix_commits	Number of commits including `fix` or `hotfix`
pr_review_rate	Share of pull requests with more than one review
review_time	Average time until first review, in days
$\lvert\mathcal{R}\rvert$	Number of distinct reviewers
$\overline{\text{LOC}}$	Average lines of code per file
$\lvert\mathcal{M}\rvert$	Number of active maintainers

Table 4.3: Repository signals used in the Expected Impact on Quality score.

Each signal is normalized to $[0, 1]$ as follows:

$$S_{\text{bug}} = \min\left(\frac{\text{bug\_ratio}}{0.30}, 1\right) \tag{4.1a}$$
$$S_{\text{fix}} = \min\left(\frac{\text{bug\_fix\_time}}{30}, 1\right) \tag{4.1b}$$
$$S_{\text{reopen}} = \min\left(\frac{\text{reopened\_ratio}}{0.05}, 1\right) \tag{4.1c}$$
$$S_{\text{hotfix}} = \min\left(\frac{\text{hotfix\_commits}}{20}, 1\right) \tag{4.1d}$$

$$G_{\text{coverage}} = \min(1 - \text{pr\_review\_rate}, 1) \tag{4.2a}$$
$$G_{\text{time}} = \min\left(\frac{\text{review\_time}}{14}, 1\right) \tag{4.2b}$$
$$G_{\text{diversity}} = \min\left(1 - \frac{|\mathcal{R}|}{10}, 1\right) \tag{4.2c}$$

$$R_{\text{LOC}} = \min\left(\frac{\overline{\text{LOC}}}{500}, 1\right) \tag{4.3a}$$
$$R_{\text{maintainer}} = \min\left(1 - \frac{|\mathcal{M}|}{10}, 1\right) \tag{4.3b}$$

These signals are then grouped into four sub-metrics:

$$\text{QualityPain} = 0.4 \cdot S_{\text{bug}} + 0.3 \cdot S_{\text{fix}} + 0.2 \cdot S_{\text{reopen}} + 0.1 \cdot S_{\text{hotfix}} \tag{4.4}$$
$$\text{ReviewGap} = 0.5 \cdot G_{\text{coverage}} + 0.2 \cdot G_{\text{diversity}} + 0.3 \cdot G_{\text{time}} \tag{4.5}$$
$$\text{MaintainabilityRisk} = 0.4 \cdot R_{\text{LOC}} + 0.6 \cdot R_{\text{maintainer}} \tag{4.6}$$
$$\text{TestGap} = \begin{cases} 1.0 & \text{if no CI available} \\ 0.0 & \text{if } f_{\text{ci}} \leq 0.05 \\ 1.0 & \text{if } f_{\text{ci}} \geq 0.40 \\ \frac{f_{\text{ci}} - 0.05}{0.35} & \text{otherwise} \end{cases} \tag{4.7}$$

where f_{ci} denotes the CI pipeline failure rate. These four sub-metrics are then aggregated into a composite repository-derived impact score:

$$\begin{aligned} \text{GitHub\_Impact} = {} & 0.30 \cdot \text{QualityPain} + 0.25 \cdot \text{ReviewGap} \\ & + 0.25 \cdot \text{TestGap} + 0.20 \cdot \text{MaintainabilityRisk} \end{aligned} \tag{4.8}$$

Score Computation. The final quality score is obtained by combining the repository-derived impact with the two practitioner-supplied modifiers and scaling to $[0, 10]$:

$$\text{EIoQ} = \text{GitHub\_Impact} \times q_c \times q_r \times 10 \tag{4.9}$$

The multiplicative structure ensures that even a substantial repository-derived quality gap yields a low score if the organization is neither critically dependent on quality nor open to external review, thereby preventing misleadingly optimistic recommendations in unfavourable adoption contexts.

Expected Impact on Flexibility (EIoF)

The flexibility score combines an objective measurement of current code inflexibility with a practitioner assessment of business-side flexibility demand.

Practitioner Inputs. Five questionnaire items are collected:

- **Q3 Modularity:** How modular is the codebase? The practitioner provides a self-assessment $m \in [0, 10]$.
- **Q4 Requirement Change Frequency:** How often do requirements change? Rated on $[0, 10]$.
- **Q5 Number of Dependent Teams:** How many teams require adaptations or extensions? Rated on $[0, 10]$.
- **Q6 Variance of Use Cases:** How diverse are the requirements across users and teams? Rated on $[0, 10]$.
- **Q7 Time-to-Market Pressure:** How critical is fast implementation for business success? Rated on $[0, 10]$.

Repository-Derived Signals. The change scope is derived from the average number of files affected per pull request. To account for diminishing returns at large scales, a logarithmic normalization is applied:

$$\text{ChangeScope} = \left(1 - \frac{\ln(\text{avg\_files\_changed})}{\ln(50)}\right) \times 10 \tag{4.10}$$

where `avg_files_changed` is clamped to $[1, 50]$ prior to computation.

Score Computation. The code flexibility gap combines the practitioner-supplied modularity self-assessment with the repository-derived change scope:

$$\text{CodeFlexGap} = \frac{(10 - m) + \text{ChangeScope}}{2} \tag{4.11}$$

Business flexibility demand is aggregated from the four practitioner ratings $q_4, \ldots, q_7$:

$$\text{FlexNeed} = \frac{q_4 + q_5 + q_6 + q_7}{40} \times 10 \tag{4.12}$$

The final flexibility score is then:

$$\text{EIoF} = \frac{\text{CodeFlexGap} \times \text{FlexNeed}}{10} \tag{4.13}$$

Expected Impact on Innovation (EIoI)

Innovation potential is assessed through four additive components, each reflecting a distinct dimension of a project's receptiveness to novel contributions.

Practitioner Inputs. Two questionnaire items are collected:

Q8 Project Age: The practitioner provides the age of the project in months $a \geq 0$.

Q9 Openness to External Ideas: How much innovation input is expected from contributors outside the core team? The practitioner selects one of five levels, mapped to a numeric score e:

- Very low $\rightarrow e = 0.0$
- Low $\rightarrow e = 0.5$
- Medium $\rightarrow e = 1.0$
- High $\rightarrow e = 1.5$
- Very high $\rightarrow e = 2.0$

Repository-Derived Signals. Three signals are derived from the repository. Documentation completeness d is determined by the presence of a `README`, a `CONTRIBUTING` file, and additional documentation artifacts:

$$d = \begin{cases} 1.5 & \text{if README, CONTRIBUTING, and additional docs present} \\ 1.0 & \text{if README and CONTRIBUTING present} \\ 0.5 & \text{if README present} \\ 0.0 & \text{otherwise} \end{cases} \tag{4.14}$$

The feature ratio score f captures the share of feature- or enhancement-type issues relative to all issues:

$$f = \begin{cases} 4 & \text{if feature\_ratio} \geq 0.40 \\ 3 & \text{if feature\_ratio} \geq 0.25 \\ 2 & \text{if feature\_ratio} \geq 0.10 \\ 1 & \text{if feature\_ratio} > 0.00 \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

Score Computation. The practitioner-supplied project age a is mapped to a maturity score p, reflecting the premise that projects in an active growth phase exhibit the greatest innovation potential:

$$p = \begin{cases} 2 & \text{if } a < 6 \\ 3 & \text{if } 6 \leq a \leq 36 \\ 1 & \text{if } 36 < a \leq 72 \\ 0 & \text{if } a > 72 \end{cases} \tag{4.16}$$

The four components are summed and clipped to the range $[0, 10]$:

$$\text{EIoI} = \min(p + f + d + e,\ 10) \tag{4.17}$$

The theoretical maximum of $p + f + d + e$ is $3 + 4 + 1.5 + 2 = 10.5$, intentionally exceeding the upper bound of 10, so that a project need not perform perfectly across all components to achieve a full score.

Expected Impact on Delivery Time (EIoDT)

Delivery time improvement is estimated from three repository-derived indicators of current process bottlenecks: issue cycle time, pull request merge time, and contributor concentration. Elevated values on any of these metrics indicate that the project would benefit from the additional parallel capacity and reduced bus-factor risk associated with inner source collaboration. As no practitioner input is required for this factor, all signals are derived exclusively from the repository.

Repository-Derived Signals. The three sub-metrics are each normalized to $[0, 1]$, where higher values indicate a more severe bottleneck:

$$S_{\text{cycle}} = \max\left(1 - \frac{\text{issue\_cycle\_time}}{90},\ 0\right) \tag{4.18}$$

$$S_{\text{PR}} = \max\left(1 - \frac{\text{pr\_time}}{30},\ 0\right) \tag{4.19}$$

$$S_{\text{conc}} = \max(1 - \text{top\_2\_share},\ 0) \tag{4.20}$$

where `issue_cycle_time` denotes the average number of days from issue open to close, `pr_time` the average number of days from pull request creation to merge, and `top_2_share` the share of total commits attributable to the two most active contributors.

Score Computation. The three sub-metrics are aggregated into the final delivery time score and scaled to $[0, 10]$:

$$\text{EIoDT} = (0.40 \cdot S_{\text{cycle}} + 0.30 \cdot S_{\text{PR}} + 0.30 \cdot S_{\text{conc}}) \times 10 \tag{4.21}$$

4.2.3 Financial Factors

Financial factors translate the operational implications of inner source adoption into monetary estimates, providing the quantitative basis for cost-benefit comparison. All financial sub-scores are computed as three-point estimates (best, average, worst case) to reflect estimation uncertainty, following a convention analogous to the three-point estimation applied in the viability model of Hirsch and Riehle (2022).

Contribution Cost

Contribution cost captures the monetary overhead incurred by external contributors when joining an inner source project. It decomposes into three additive

components, each expressed as an estimated number of developer hours multiplied by a configurable hourly rate r:

$$C_{\text{contrib}} = C_{\text{onboarding}} + C_{\text{coordination}} + C_{\text{quality}} \tag{4.22}$$

Practitioner Inputs. Three project-level parameters are supplied by the practitioner:

Q10 Expected Contributors: The anticipated number of external contributors $n \geq 1$.

Q11 Expected Pull Request Count: The anticipated number of pull requests $p \geq 0$ over the project duration.

Q12 Hourly Rate: The personnel cost per developer hour $r > 0$, in the currency of the organization.

Repository-Derived Signals. The following signals are derived from the repository:

- Documentation completeness d, computed from the presence of a `README`, a `CONTRIBUTING` file, and a test suite[1]:

$$d = 0.4 \cdot 1_{\text{readme}} + 0.4 \cdot 1_{\text{contributing}} + 0.2 \cdot 1_{\text{tests}} \tag{4.23}$$

- Codebase size penalty $s = \min(\text{LOC}/200{,}000,\ 1.0)$, reflecting the additional ramp-up effort required for large codebases.
- Average review wait time per pull request t_{review}, in hours.
- CI failure rate f_{ci} and pull request reopen rate f_{ro}, both expressed as fractions of total pull requests.

Score Computation. Onboarding cost accounts for the ramp-up effort a new contributor must invest before becoming productive:

$$C_{\text{onboarding}} = \left[(1 - d) \cdot 12 + s \cdot 4\right] \cdot n \cdot r \tag{4.24}$$

Coordination cost represents the latency overhead arising from review wait times across all expected pull requests:

$$C_{\text{coordination}} = t_{\text{review}} \cdot p \cdot r \tag{4.25}$$

[1] Binary indicator variables $1_x \in \{0, 1\}$, where 1 indicates presence and 0 absence of the respective signal x.

Quality friction cost accounts for rework overhead due to CI pipeline failures and pull request reopens, with remediation constants of 1 hour per CI failure and 3 hours per reopen:

$$C_{\text{quality}} = \left(f_{\text{ci}} \cdot p \cdot 1.0 + f_{\text{ro}} \cdot p \cdot 3.0\right) \cdot r \tag{4.26}$$

The likely total cost C_{contrib} is then used to derive a three-point scenario estimate to account for inherent uncertainty in cost projections:

$$C_{\text{best}} = 0.8 \cdot C_{\text{contrib}}, \quad C_{\text{avg}} = C_{\text{contrib}}, \quad C_{\text{worst}} = 1.3 \cdot C_{\text{contrib}} \tag{4.27}$$

Conversion and Integration Cost

Conversion and integration cost captures both the one-time expenditure required to prepare the codebase for inner source collaboration and the recurring overhead of managing external contributions over the project lifetime.

Practitioner Inputs. Two project-level parameters are supplied by the practitioner:

Q13 Remaining Project Duration: The anticipated remaining lifetime of the project $T > 0$, in months.

Q14 Hourly Rate: The personnel cost per developer hour $r > 0$, in the currency of the organization.

Repository-Derived Signals. The following signals are derived from the project repository:

- Presence indicators 1_{tests}, 1_{ci}, 1_{readme}, reflecting whether a test suite, CI configuration, and README are present.
- Dependency flag 1_{deps}, indicating whether the project has many external dependencies.
- Average pull request review time t_{review}, in hours.
- Average number of review comments per pull request N_{comments}.
- Average number of CI failures per pull request N_{ci}.
- Average number of pull requests per month N_{PR}.

Score Computation. Conversion cost is a one-time expenditure to bring the codebase to a standard suitable for inner source collaboration. Base conversion hours scale with codebase size, and are adjusted upward by a multiplicative penalty factor reflecting the absence of quality prerequisites:

$$M = \prod_{i=1}^{4} m_i, \quad m_i = \begin{cases} 1.5 & \text{if } 1_{\text{tests}} = 0 \\ 1.2 & \text{if } 1_{\text{ci}} = 0 \\ 1.1 & \text{if } 1_{\text{readme}} = 0 \\ 1.3 & \text{if } 1_{\text{deps}} = 1 \\ 1.0 & \text{otherwise} \end{cases} \tag{4.28}$$

$$C_{\text{conversion}} = \frac{\text{LOC}}{100} \cdot M \cdot r \tag{4.29}$$

Ongoing integration cost accumulates over the remaining project duration and captures the recurring overhead per pull request, comprising review time, comment handling, and CI failure remediation:

$$H_{\text{PR}} = t_{\text{review}} + N_{\text{comments}} \cdot 0.25 + N_{\text{ci}} \cdot 0.5 \tag{4.30}$$

$$C_{\text{integration}} = H_{\text{PR}} \cdot N_{\text{PR}} \cdot T \cdot r \tag{4.31}$$

Both components are expressed as three-point scenario estimates to account for inherent uncertainty in cost projections:

$$C_{\text{conversion}}^{\text{best/avg/worst}} = C_{\text{conversion}} \cdot \{0.8,\ 1.0,\ 1.3\} \tag{4.32}$$

$$C_{\text{integration}}^{\text{best/avg/worst}} = C_{\text{integration}} \cdot \{0.8,\ 1.0,\ 1.3\} \tag{4.33}$$

The total conversion and integration cost per scenario is then the sum of the two corresponding component estimates:

$$C_{\text{CI}}^{s} = C_{\text{conversion}}^{s} + C_{\text{integration}}^{s}, \quad s \in \{\text{best},\ \text{avg},\ \text{worst}\} \tag{4.34}$$

Expected Maintenance Contribution

This factor estimates the additional maintenance effort incurred by the organization when opening a project to inner source collaboration. All inputs are provided by the practitioner because maintenance demand cannot be derived from repository signals alone.

Practitioner Inputs. Three questionnaire items are collected:

Q15 Baseline Maintenance Demand (BMD): How much time is currently spent on maintenance per month? The practitioner selects one of five levels, mapped to a monthly hour estimate:

- Very low → BMD = 8 h/month
- Low → BMD = 16 h/month
- Medium → BMD = 32 h/month
- High → BMD = 48 h/month
- Very high → BMD = 64 h/month

Q16 Inner Source Overhead Factor (ISOF): How much additional effort is expected from opening the project (reviews, documentation, coordination)? The practitioner selects one of five levels, mapped to a multiplicative factor:

- Very low → ISOF = 1.10
- Low → ISOF = 1.15
- Medium → ISOF = 1.25
- High → ISOF = 1.30
- Very high → ISOF = 1.40

Q17 Expected External Maintenance Share (EEMS): What share of maintenance could realistically be absorbed by external teams? The practitioner selects one of five levels, mapped to a fractional share:

- Very low → EEMS = 0.15
- Low → EEMS = 0.25
- Medium → EEMS = 0.35
- High → EEMS = 0.50
- Very high → EEMS = 0.65

Score Computation. The Future Maintenance Demand (FMD) under inner source conditions is first projected by scaling the baseline with the overhead factor:

$$\mathrm{FMD} = \mathrm{BMD} \times \mathrm{ISOF} \tag{4.35}$$

The Expected Maintenance Contribution (EMC) is then computed over the remaining project duration T at hourly rate r:

$$\text{EMC} = \text{FMD} \times \text{EEMS} \times T \times r \tag{4.36}$$

The EMC represents the projected maintenance cost under inner source and is included in the total cost used in the final viability calculation. A three-point scenario estimate accounts for uncertainty in the practitioner-supplied inputs:

$$\text{EMC}_{\text{best/avg/worst}} = \text{EMC} \cdot \{0.9,\ 1.0,\ 1.1\} \tag{4.37}$$

Cost of Alternative Development

The cost of pursuing traditional closed development serves as the financial baseline against which inner source expenditure is compared. As this figure cannot be derived from repository metrics, it is supplied directly by the practitioner via a single questionnaire item:

Q18 Cost of Alternative Development: What is the estimated total cost of developing this component independently, without inner source collaboration? The practitioner provides a three-point estimate directly:

- Best case $\rightarrow C_{\text{alt}}^{\text{best}}$
- Average case $\rightarrow C_{\text{alt}}^{\text{avg}}$
- Worst case $\rightarrow C_{\text{alt}}^{\text{worst}}$

These three values are used as the reference baseline in the financial score computation detailed in the following section.

4.2.4 Dimension Aggregation and Overall Score

This section describes how the factor-level results computed in sections 4.2.1 to 4.2.3 are aggregated into a single composite viability score. The aggregation proceeds in three stages: (1) the financial factor group scores are collapsed into a single financial score via Program Evaluation and Review Technique (PERT) weighting, (2) the four factor group scores are combined pairwise into four dimension scores, and (3) the dimension scores are averaged into the overall viability score.

Financial Score

The three cost components (*Contribution Cost*, *Conversion and Integration Cost*, and *Expected Maintenance Contribution*) as well as the *Cost of Alternative De-*

velopment are each represented as three-point estimates $(C^{\text{best}}, C^{\text{avg}}, C^{\text{worst}})$. To collapse each estimate into a single expected value, the PERT weighted mean, originally introduced by Malcolm et al. (1959) for research and development program evaluation, is applied:

$$\mu_{\text{PERT}}(C) = \frac{C^{\text{best}} + 4 \cdot C^{\text{avg}} + C^{\text{worst}}}{6} \tag{4.38}$$

This formulation assigns four times the weight to the most likely estimate relative to the extreme scenarios, reflecting the assumption that the average case is the most reliable estimate while still accounting for tail risk. The total expected inner source cost is then:

$$\begin{aligned} \mu_{\text{IS}} &= \mu_{\text{PERT}}(C_{\text{contrib}}) + \mu_{\text{PERT}}(C_{\text{CI}}) + \mu_{\text{PERT}}(\text{EMC}) \\ \mu_{\text{alt}} &= \mu_{\text{PERT}}(C_{\text{alt}}) \end{aligned} \tag{4.39}$$

The financial score is computed as the relative cost advantage of inner source over alternative development, normalized to $[0, 10]$:

$$\Delta = \frac{\mu_{\text{alt}} - \mu_{\text{IS}}}{\mu_{\text{IS}}} \tag{4.40}$$

$$\text{FS} = \frac{\text{clamp}(\Delta,\ -1,\ 1) + 1}{2} \times 10 \tag{4.41}$$

A score of $\text{FS} = 10$ indicates that inner source is substantially cheaper than the alternative, $\text{FS} = 5$ indicates cost parity, and $\text{FS} = 0$ indicates that inner source costs at least twice as much as independent development. If $\mu_{\text{IS}} = 0$, a neutral score of 5.0 is returned as a fallback.

Dimension Scores

The strategic value score SV and human factors score HF are computed as weighted and unweighted averages of their respective factor ratings, as described in section 4.2.1. The performance score PF is the equally weighted mean of the four performance factor scores.Together with the financial score FS, these

four group scores are combined pairwise into four dimension scores, following the decision matrix of Hirsch and Riehle (2022) introduced in table 4.1:

$$D_{\text{quality}} = 0.5 \cdot \text{SV} + 0.5 \cdot \text{PF} \tag{4.42}$$
$$D_{\text{company}} = 0.5 \cdot \text{SV} + 0.5 \cdot \text{HF} \tag{4.43}$$
$$D_{\text{project}} = 0.5 \cdot \text{PF} + 0.5 \cdot \text{FS} \tag{4.44}$$
$$D_{\text{resource}} = 0.5 \cdot \text{HF} + 0.5 \cdot \text{FS} \tag{4.45}$$

Overall Viability Score

The four dimension scores are aggregated with equal weight into the composite viability score $\mathcal{V} \in [0, 10]$:

$$\mathcal{V} = \frac{1}{4} \sum_{d \in \{\text{quality},\ \text{company},\ \text{project},\ \text{resource}\}} D_d \tag{4.46}$$

This score constitutes the training label for the machine learning pipeline described in section 4.4. Higher values indicate a more favorable inner source viability assessment across all four dimensions, while lower values signal that one or more dimensions present significant obstacles to successful collaboration.

4.3 Synthetic Dataset Design

As established in section 4.1, no real-world dataset of inner source projects annotated with ground-truth viability scores exists, as the viability scoring framework introduced in section 4.2 is itself a novel artifact. The supervised training of the ML model therefore requires a purpose-built synthetic dataset whose records are generated programmatically and subsequently labeled by the scoring algorithm. The design of this dataset is governed by two requirements: (1) the generated data must be informationally complete w.r.t. the scoring framework, i.e., every record must supply all inputs required by the algorithm to produce a deterministic label; and (2) the dataset must exhibit sufficient archetype diversity to prevent the trained model from overfitting to a single viability regime.

4.3.1 Project Type and Scope

The synthetic dataset models the *Platform Component* project type, i.e., a reusable software component developed collaboratively across organizational silo boundaries within a single enterprise. Platform components represent the canonical inner source use case: they are consumed by multiple business units, benefit from

pooled maintenance effort, and generate the cross-team contributor dynamics that inner source is designed to harness (Riehle et al., 2016). Alternative project types—such as federation models spanning multiple enterprises or single-team tools with no intended cross-organizational adoption—are structurally distinct and out of scope for this work.

Each synthetic project is characterized by a fixed set of practitioner-supplied organizational parameters (e.g., strategic value ratings, human factor assessments, financial estimates) and a sequence of monthly GitHub-derived repository metric snapshots. The practitioner inputs are drawn at project initialization and remain structurally stable across the observation window, reflecting the assumption that organizational parameters change on longer timescales than repository activity. Repository metrics, by contrast, evolve month-to-month according to the lifecycle model described in section 4.3.3. Each record is therefore informationally complete: the scoring algorithm can compute a fully determined viability score $\mathcal{V}$ for every record without imputation or approximation, yielding the ground-truth training label for the ML model. The feature space is consequently not an independent design decision but a direct consequence of the scoring framework's input specification.

4.3.2 Project Archetypes

To ensure that the dataset spans the realistic outcome space of inner source viability, four project archetypes (referred to as *Fate Profiles*) are defined. Each profile specifies the distributional parameters governing the initial state and temporal evolution of observable signals for projects of that type. The archetypes differ in their organizational context, repository activity patterns, and expected viability outcomes.

The *thriving*, *average*, and *struggling* archetypes represent different positions along a performance axis while assuming broadly comparable organizational scale. By contrast, the *small company* archetype models a structurally distinct context in which the economic benefits of inner source are constrained by the limited size of the development organization. Even if repository metrics appear neutral, the financial sub-score tends to be unfavorable in this scenario because the coordination overhead required to open a project internally exceeds the potential savings from cross-team collaboration. Including this archetype ensures that the dataset contains cases in which inner source viability is low for financial rather than performance-related reasons. This is important for learning the interaction between organizational scale and the financial scoring sub-dimension. During dataset generation, archetypes are sampled with approximately equal probability in order to obtain a balanced distribution of viability outcomes for model training.

Archetype	Organizational Context	Repository Dynamics	Typical Viability Outcome
Thriving	High strategic importance, experienced teams, strong inner source readiness	Increasing collaboration, rising pull request activity, improving quality metrics	High viability scores
Average	Moderate inner source experience with mixed organizational support	Mostly stable repository signals with moderate fluctuations	Medium viability scores
Struggling	High delivery pressure, limited experience with collaborative development	Declining quality indicators, limited reviewer diversity, unstable activity patterns	Low viability scores
Small Company	Small development organization with limited scale advantages for inner source	Neutral technical signals but high coordination overhead relative to team size	Low financial viability

Table 4.4: Synthetic project archetypes used during dataset generation.

4.3.3 Lifecycle Phases and Temporal Dynamics

Each synthetic project is simulated over a fixed observation window of N_{months} months, subdivided into three sequential lifecycle phases whose boundaries are defined proportionally to the total duration: an *incubation* phase ($\approx$ the first 30 %), an *adoption* phase ($\approx$ the next 40 %), and a *maturity* phase ($\approx$ the final 30 %). This tripartite structure is a modeling decision motivated by the need to capture temporal signal heterogeneity. During incubation, all archetypes exhibit broadly similar signal distributions, as cross-team contributions have not yet materialized. Fate divergence accelerates during adoption, which constitutes the primary window in which archetype-consistent patterns emerge in reviewer diversity, pull request volume, and quality metrics (Stol et al., 2014). In the maturity phase, dynamics stabilize along archetype-consistent trajectories. This structure ensures that the synthetic data is temporally informative for the ML model, which must learn to distinguish early-phase patterns predictive of long-term viability from short-term fluctuations.

4.4 Machine Learning Model Design

The ML component constitutes the *Viability Predictions* stage of the pipeline illustrated in fig. 4.1. It receives the labeled dataset produced by the scoring algorithm and is designed to predict the viability score $\mathcal{V}$ of an ongoing inner source project for the months ahead, given only its observed history up to the current point in time. The following subsections describe the four central design decisions: the temporal feature representation (section 4.4.1), the train-test split strategy (section 4.4.2), the choice of regressor (section 4.4.3), and the recursive multi-step forecasting procedure (section 4.4.4).

4.4.1 Temporal Feature Representation

The raw dataset provides one record per project per month. To expose temporal dynamics to the model, each monthly snapshot is augmented with lagged values and three-month rolling means of all input features, computed exclusively from prior observations. This strict backward-looking constraint prevents data leakage, ensuring that the model never sees future information during training or inference. The same transformation is applied to the target variable itself, since recent viability scores are expected to be the strongest predictor of near-term viability. The current month index is retained as an additional feature so the model can account for lifecycle effects, as the signal patterns of projects in the incubation and maturity phases differ systematically (section 4.3.3).

4.4.2 Train-Test Split

The dataset is split at the project level, stratified by fate archetype. Approximately 10 % of the projects from each archetype are held out as the test set, with the remaining 90 % used for training. This design choice reflects the intended deployment scenario: The model should generalize to entirely unseen projects, not merely extrapolate known ones forward in time. Stratification by archetype ensures that all four project types (thriving, average, struggling, small company) are proportionally represented in both partitions, preventing the test set from being dominated by a single viability regime.

4.4.3 Regressor

LightGBM (Ke et al., 2017), a gradient-boosted decision tree framework, is selected as the primary model. The choice is motivated by the characteristics of the problem setting: The input data is tabular with mixed numerical and categorical features, and the scoring framework introduces features of varying predictive relevance. Gradient-boosted tree ensembles are well-suited to precisely this regime. Empirical benchmarks demonstrate that they remain state-of-the-art on tabular

datasets, in part because they are inherently robust to uninformative features and require no feature-specific preprocessing (Grinsztajn et al., 2022). LightGBM's native support for early stopping further mitigates the risk of overfitting to the synthetic training distribution. A Linear Regression baseline is trained on the same feature set to quantify the benefit of the non-linear model and to establish a minimum performance threshold.

4.4.4 Recursive Multi-step Forecasting

At inference time, the model operates in a *recursive multi-step forecasting* mode (Bontempi et al., 2013): a practitioner supplies the observed history of a project for an initial window of k months, and the model then predicts all subsequent months by substituting its own prior predictions into the lag features required for each next step. This strategy reflects the realistic use case in which a project controller observes an ongoing project and wishes to obtain a medium-term viability forecast before the full observation window has elapsed. Its principal limitation is error accumulation: deviations in early predictions propagate into later steps as lag inputs, amplifying forecast errors over longer horizons (Bontempi et al., 2013). The evaluation in chapter 7 quantifies this effect across all test projects and archetype categories.

5 Implementation

This chapter describes the concrete realization of the artifact whose design was established in chapter 4. The implementation follows the pipeline introduced in section 4.1 and is organized accordingly: Section 5.1 motivates the technology choices that underpin the entire system. Section 5.2 presents the software architecture and maps its components onto the design pipeline of fig. 4.1. Section 5.3 details the realization of the viability scoring algorithm, with emphasis on the data model that bridges the raw input interface and the factor-group structure of the scoring framework. Section 5.4 describes the implementation of the synthetic dataset generator and the labeling procedure. Finally, section 5.5 covers the machine learning pipeline, including feature engineering, model configuration, and the recursive multistep forecasting procedure introduced in section 4.4.4.

5.1 Used Technologies and Libraries

Data Storage – JSON. All intermediate and final datasets produced by the pipeline are persisted as JavaScript Object Notation (JSON) files. Each monthly project observation is a heterogeneous record combining flat scalar values, nested sub-objects, and metadata fields, for which JSON′s schema-free hierarchical structure is a natural fit. Human-readability further simplifies debugging of intermediate pipeline outputs.

Implementation Language – Python. The entire pipeline is implemented in Python 3 (https://www.python.org). Python is the de facto standard language for scientific computing and machine learning research, offering a mature ecosystem of numerical and data-science libraries directly applicable to the components of this thesis.

Libraries. The implementation relies on the following third-party libraries, each selected for a specific role within the pipeline:

NumPy (https://numpy.org) provides the foundational array operations used throughout all pipeline components, including the noise sampling and clamping

routines in the synthetic data generator and the array manipulations required during feature engineering.

Pandas (https://pandas.pydata.org) is used for all tabular data operations. The labeled JSON dataset is loaded into a `DataFrame` with one row per project-month observation; all subsequent transformations, including lag construction, rolling-mean computation, and the project-level stratified split, are implemented as `DataFrame` operations. Its `groupby` and `shift` primitives are particularly well-suited to the time-series structure of the data.

LightGBM (https://lightgbm.readthedocs.io) provides the gradient-boosted decision tree regressor that constitutes the primary ML model, as motivated in section 4.4.3. Its `LGBMRegressor` interface integrates directly with the scikit-learn Application Programming Interface (API), and its native support for early stopping via a validation set is used to prevent overfitting to the synthetic training distribution.

scikit-learn (https://scikit-learn.org) supplies the `LinearRegression` baseline model, the `StandardScaler` used to normalize the feature matrix for the linear model, and the evaluation metrics Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 used throughout the evaluation.

Matplotlib (https://matplotlib.org) is used for all result visualizations.

5.2 Component Architecture

The component architecture shown in fig. 5.1 maps the three pipeline stages introduced in section 4.1 onto five concrete Python modules. The *Data Generation* stage is realized by the `SyntheticDataGenerator`. The *Data Labeling* stage is split across the `DataTransformer`, which restructures each raw record into the factor-group schema expected by the scorer, the `ViabilityScorer`, which computes $\mathcal{V}$, and the `DatasetLabeler`, which orchestrates both. The *Viability Predictions* stage is handled by the `MLPipeline`. Separating `DataTransformer` and `ViabilityScorer` ensures that the scoring algorithm operates on a well-defined, format-stable input structure independent of how the raw data was produced—whether by the synthetic generator or, in future work, by a real-world data collection tool.

5.3 Viability Assessment Algorithm

This section describes the implementation of the viability scoring framework designed in section 4.2, realized across the `DataTransformer` and `ViabilityScorer` modules. Since the individual factor scoring functions are direct implementations

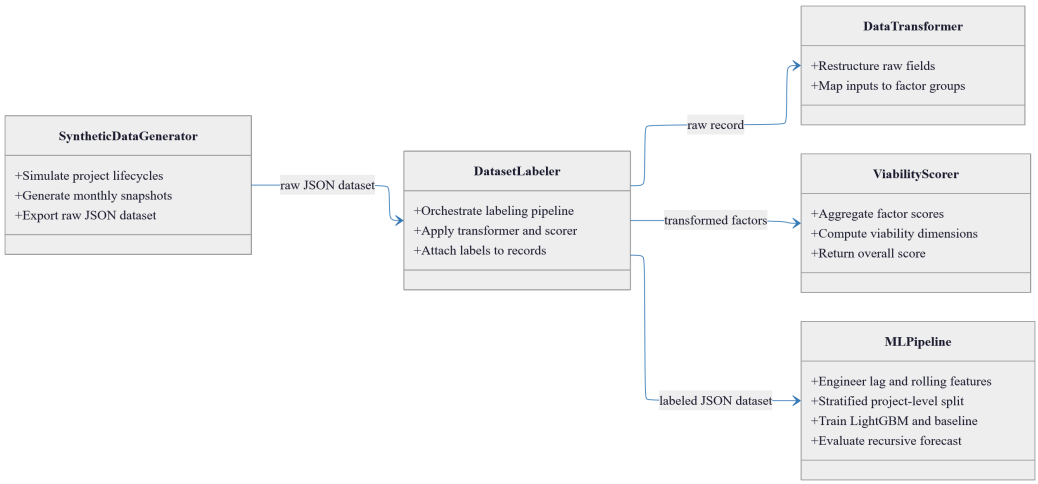

Figure 5.1: Software component architecture of the inner source viability prediction pipeline, realized across five Python modules.

of the formulations presented there, the focus here is on the two implementation-specific aspects: the input data layout and the final aggregation procedure.

Scoring Pipeline and Input Data Layout. The `DataTransformer` maps each raw record onto the nested factor-group schema required by the scoring algorithm, organized into five top-level keys: `project_meta`, `strategic_value_factors`, `human_factors`, `performance_factors`, and `financial_factors`. A representative excerpt of the full transformed schema is provided in the Appendix (listing A.1).

Score Calculation and Aggregation. Once the transformed record has been passed through the individual factor scoring functions, each factor group is reduced to a scalar score in $[0, 10]$ following the aggregation rules of section 4.2. The four group scores are then combined into the overall score $\mathcal{V}$ via the dimension composition of table 4.1 and returned as a `ViabilityScore` dataclass instance (see listing 5.1), which the `DatasetLabeler` attaches to each record as its training label.

5.4 Synthetic Dataset Generation and Labeling

This section describes the implementation of the synthetic dataset generator and the subsequent labeling procedure, realizing the design established in section 4.3.

```python
@dataclass
class ViabilityScore:
    quality_dimension: float
    company_dimension: float
    project_dimension: float
    resource_dimension: float
    overall_score: float
```

Listing 5.1: `ViabilityScore` dataclass

Dataset Layout. Each record corresponds to a single monthly observation of one synthetic project. The raw layout separates the two input categories into a flat `github_data` object and a flat `user_input` object, keeping the generator agnostic of the scoring framework's internal factor-group structure (c.f. listing A.2). The `DataTransformer` acts as the explicit boundary between the two representations. Each record additionally carries `project_id` and `timestamp` for identification, `phase` and `fate` as generation metadata never exposed to the ML model, and a `label` field initially set to `null` and populated by the labeling step.

Dataset Generation. The generator simulates 150 projects over 12 monthly snapshots each, yielding 1 800 records in total. Time-evolving signals such as `bug_ratio`, `reviewer_diversity`, and `ci_fail_rate` drift according to archetype-specific rates and are perturbed by Gaussian noise at each step. Several fields are derived from correlated signals (`prs_with_review` from `reviewer_diversity`, `hotfix_commits` from `bug_ratio`) to reflect the interdependencies observable in real repository histories rather than producing implausible combinations of independently sampled values.

Dataset Labeling. The `DatasetLabeler` iterates over the unlabeled dataset and applies the `DataTransformer` and `ViabilityScorer` to each record in sequence. The resulting score is attached to the `label` field, replacing the initial `null` value. Records for which scoring raises an exception are marked with a `label_error` field and excluded from model training.

5.5 Machine Learning Model Implementation

This section describes the implementation of the ML pipeline, realizing the design established in section 4.4. The pipeline is structured as a sequential series of steps: feature engineering, train-test splitting, model training, and recursive forecast evaluation.

Feature Engineering. The labeled dataset is loaded into a flat `DataFrame` with one row per project-month observation, where all `github_data` and `user_input` fields are prefixed with `gh_` and `ui_` respectively to avoid name collisions. The four viability dimension scores are explicitly excluded from the feature matrix to prevent sub-score leakage into the target. For every remaining numeric column, three lag features and one three-month rolling mean are constructed by shifting the grouped time series strictly backward, ensuring that no future information is visible at any step. The same transformation is applied to the target variable `overall_score` itself, and the current month index `month_idx` is retained to allow the model to account for lifecycle phase effects. Rows with incomplete lag windows are dropped after this step.

Train-Test Split. The dataset is split at the project level, stratified by fate archetype, with approximately 10% of projects per archetype held out as the test set. The `fate` field is never carried into the feature matrix, ensuring it cannot act as a proxy feature during training or evaluation.

Model Training. The LightGBM regressor is trained on the full training partition with early stopping monitored on a held-out validation slice consisting of the final month of each training project. The upper bound of 500 estimators is set conservatively, with early stopping terminating training after 50 rounds without improvement in RMSE on the validation slice. Table 5.1 lists the hyperparameter configuration used for training. The linear regression baseline is trained on the same feature matrix after standardization with `StandardScaler`, providing a minimum performance threshold against which the non-linear model is assessed.

Recursive Forecast Evaluation. At evaluation time, the first 7 months of each test project are provided as real observed input. All subsequent months are predicted recursively by substituting the model's own prior predictions into the lag and rolling-mean features of the target variable, while all remaining input features retain their true observed values—reflecting the deployment scenario described in section 4.4.4. The core substitution procedure is shown in listing 5.2. Per-project and aggregate MAE, RMSE, and R^2 scores are reported across all four archetype categories.

Hyperparameter	Value	Rationale
`n_estimators`	500	Upper bound; early stopping prevents overfit
`learning_rate`	0.05	Small steps for more robust generalization
`num_leaves`	31	Default complexity; balances bias and variance
`min_child_samples`	10	Minimum samples per leaf for regularization
`subsample`	0.8	Row sampling per tree to reduce variance
`colsample_bytree`	0.8	Feature sampling per tree to reduce variance
`random_state`	42	Fixed seed for reproducibility

Table 5.1: LightGBM hyperparameter configuration used during training.

```

for month in forecast_months:
    row = feature_vector(month)

    # substitute predicted scores into lag features
    for lag in [1, 2, 3]:
        source_month = month - lag
        if source_month >= known_months:
            row[f"overall_score_lag{lag}"] = predictions[source_month]

    # update rolling mean from mixed real/predicted history
    recent = [predictions.get(month - i,
              score_history.get(month - i))
              for i in [1, 2, 3]]
    row["overall_score_roll3"] = mean(recent)

    predictions[month] = model.predict(row)

```

Listing 5.2: Recursive multi-step forecast procedure.

6 Demonstration

This chapter demonstrates the artifact within its intended context by tracing concrete execution paths through the two analytically central pipeline stages: the viability scoring algorithm and the ML forecasting model.

The scoring demonstration in section 6.1 traces the aggregation chain for two representative synthetic snapshots, one from the thriving archetype and one from the struggling archetype, as defined in section 4.3.2. The ML demonstration in section 6.2 uses two held-out test projects, P_043 from the small company archetype and P_105 from the struggling archetype. All input values are drawn directly from records produced by the `SyntheticDataGenerator`. The statistical properties of the full synthetic dataset are analyzed in depth in chapter 7.

6.1 Viability Scoring Demonstration

Table 6.1 traces the complete aggregation chain for one representative snapshot from each of the two archetypes at month 6, from factor group scores through dimension scores to the viability score $\mathcal{V}$, following the three-stage aggregation defined in section 4.2.4.

The two snapshots produce scores that illustrate contrasting patterns of viability. The thriving snapshot attains $\mathcal{V} = 7.29$, with the resource dimension ($D_{\text{resource}} = 8.29$) as the strongest contributor, driven primarily by a high financial sub-score (FS = 9.84). All four dimensions contribute positively and no single dimension acts as a bottleneck, reflecting favorable organizational and project conditions. The struggling snapshot yields $\mathcal{V} = 3.84$, with the resource dimension ($D_{\text{resource}} = 2.54$) as the most depressed contributor. The particularly low human factor score (HF = 1.25) drives the company dimension down to $D_{\text{company}} = 3.38$, capturing a situation in which limited personnel availability and weak inner source experience jointly constrain collaboration viability regardless of other conditions.

The project dimension of the struggling snapshot ($D_{\text{project}} = 4.30$) is comparatively less significant than the resource dimension. This is a consequence of the scoring framework's design: a high time-to-market pressure entered by the

Score	Thriving	Struggling
Factor Group Scores		
Strategic Value (SV)	7.00	5.50
Human Factors (HF)	6.75	1.25
Performance (PF)	5.55	4.75
Financial (FS)	9.84	3.84
Dimension Scores		
Quality (D_{quality})	6.28	5.13
Company (D_{company})	6.88	3.38
Project (D_{project})	7.70	4.30
Resource (D_{resource})	8.29	2.54
Overall Score		
$\mathcal{V}$	**7.29**	**3.84**

Table 6.1: Scoring chain trace for representative thriving and struggling archetype snapshots at month 6.

practitioner elevates the EIoDT component, as the slow issue cycle time and high contributor concentration of the struggling snapshot constitute a process bottleneck that inner source collaboration could partially mitigate. The scoring algorithm therefore avoids a uniformly negative assessment of the project dimension despite unfavorable repository signals, reflecting the intended nuance of the decision model of Hirsch and Riehle (2022).

6.2 Machine Learning Model Demonstration

The ML pipeline is demonstrated through the recursive forecast trajectories produced for the two selected test projects by the LightGBM. The model observes months 0–6 of each project as the known input window and predicts months 7–11 recursively, as described in section 4.4.4.

Figures 6.1 and 6.2 show the ground-truth viability trajectory alongside the model's recursive forecast for each project. The lower panel of each figure plots the absolute forecast error per step, with a dashed line indicating the per-project MAE over the forecast horizon.

For project P_043, the ground-truth trajectory is structurally flat, oscillating narrowly around $\mathcal{V} \approx 4.0$ throughout the observation window. This behavior

is characteristic of the small company archetype, in which a dominant financial penalty produces a stable, low-variance score sequence throughout the project lifecycle. The model tracks this plateau accurately, with absolute errors remaining well below 0.10 for most forecast steps and only a minor divergence at month 11, yielding a per-project MAE of 0.111.

Project P_105 presents a more demanding forecasting context. The ground-truth trajectory is irregular within the range $\mathcal{V} \approx 4.6$–5.7, exhibiting a pronounced dip at month 9 that is not present in the observation window. The model maintains a near-constant prediction in the range $\mathcal{V} \approx 5.1$–5.4, correctly identifying the broad stability of the trajectory but failing to anticipate the transient excursion at month 9. The error at that step is the largest in the forecast horizon (≈ 0.49), and the subsequent recovery in the ground truth causes the error to decrease again at month 10. The per-project MAE of 0.260 remains within one score unit, confirming that the model captures the trajectory's central tendency despite being unable to resolve short-term fluctuations from lag features alone.

Taken together, the two cases illustrate both the strength and the boundary of the recursive forecasting approach. On structurally regular trajectories such as P_043, the autoregressive model extrapolates accurately from the observed window. On trajectories with transient unexptected disturbances such as in P_105, the model produces directionally sound but smoothed forecasts in which event-driven deviations are not resolved. Both outcomes are consistent with the design intent of the artifact: the forecasts serve as a continuous viability signal for project controllers rather than a point prediction of individual monthly values.

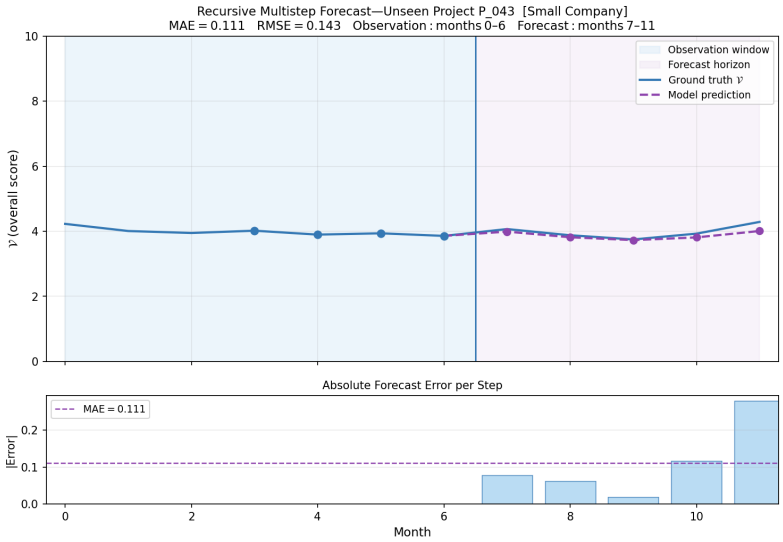

Figure 6.1: Recursive multistep forecast for the small company demonstration project P_043. The model observes months 0–6 and predicts months 7–11 recursively. Ground-truth scores are shown for reference only and are not available to the model during inference.

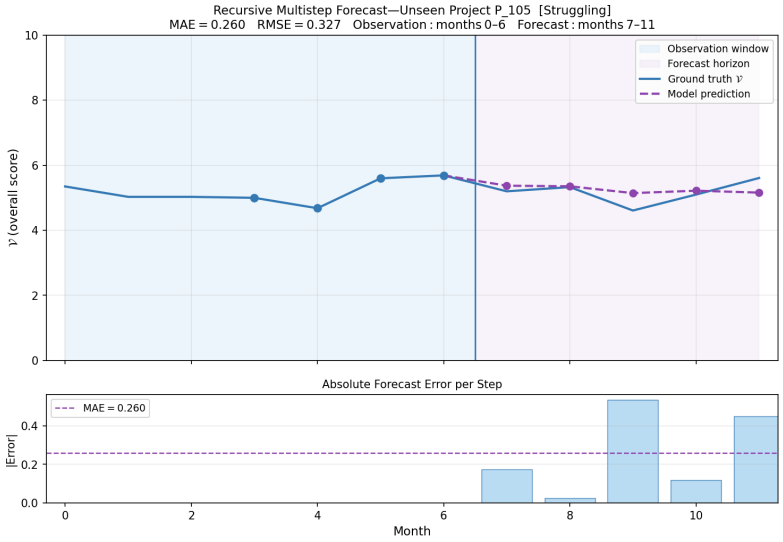

Figure 6.2: Recursive multistep forecast for the struggling demonstration project P_105.

7 Evaluation

This chapter evaluates the artifact developed in chapters 4 and 5 against the objectives established in chapter 3. It is structured as four sections: section 7.1 establishes the evaluation scope, setup, and metrics; section 7.2 analyses the statistical properties of the synthetic dataset; section 7.3 reports model performance; and section 7.4 synthesizes the findings, with addressing the principal validity threats.

7.1 Evaluation Methodology

Scope and Questions. The artifact under evaluation is the end-to-end pipeline consisting of the viability scoring algorithm, the synthetic dataset generator, and the ML forecasting model. Two questions guide the evaluation: first, whether the scoring algorithm produces a dataset that is statistically well-formed and suitable for supervised learning; second, whether the trained model generalizes to unseen inner source projects and produces accurate multi-step viability forecasts.

Experimental Setup. For each held-out test project, months 0–6 are provided as the known observation window and months 7–11 are predicted recursively, as described in section 4.4.4. The observation window of seven months was chosen because it corresponds approximately to the boundary between the adoption and maturity phases defined in section 4.3.3, the point at which a project controller has gathered sufficient history to assess trajectory while retaining meaningful time to act on a forecast. The test set consists of 14 projects unseen during training, drawn from an archetype-stratified held-out split as detailed in section 4.4.2 and summarized in table 7.1.

Performance Metrics. Model performance is quantified using three complementary metrics evaluated on the forecast horizon, i.e. months 7 through 11. Mean Absolute Error (MAE) measures the average absolute deviation between predicted and true viability scores in the original score unit ($[0, 10]$), providing a directly interpretable error magnitude. Root Mean Squared Error (RMSE) pe-

Archetype	Total	Train	Test
Thriving	44	40	4
Average	40	36	4
Struggling	44	40	4
Small Company	22	20	2
Total	**150**	**136**	**14**

Table 7.1: Archetype-stratified train-test split: project counts per category.

nalizes large deviations more heavily, making it sensitive to catastrophic forecast failures that MAE may understate. The coefficient of determination R^2 quantifies the proportion of variance in the true scores explained by the model, where zero corresponds to predicting the training-set mean and one to a perfect fit.

7.2 Dataset Quality and Score Distribution

This section assesses two properties: the overall statistical distribution of the computed viability scores and the temporal dynamics of score trajectories across archetypes.

Score Distribution. Table 7.2 reports summary statistics of the overall viability score $\mathcal{V}$ computed by the scoring algorithm for all 1 800 observations. The mean score of 5.41 and median of 5.41 indicate a symmetric distribution centered close to the midpoint of the scale, suggesting that the scoring framework does not exhibit a systematic upward or downward bias across the full dataset. The standard deviation of 1.09 corresponds to roughly 11 % of the full scale range, indicating moderate spread rather than concentration around a degenerate value.

Archetype	N	Min	Max	Mean	Std
Thriving	528	5.07	7.81	6.485	0.837
Average	480	3.84	6.67	5.379	0.809
Struggling	528	3.09	5.98	5.009	0.684
Small Company	264	3.20	5.63	4.128	0.561
Overall	**1800**	**3.09**	**7.81**	**5.411**	**1.091**

Table 7.2: Viability score distribution by archetype.

The per-archetype breakdown confirms that the scoring algorithm correctly resolves the structural differences between the synthetic archetypes: thriving projects attain the highest mean score (6.49), followed by average (5.38) and struggling (5.01) projects, while small company projects occupy the lowest range (4.13). The absence of overlap in the means, together with modest standard deviations within each archetype, indicates that the dataset provides sufficient discriminability for supervised regression. It is noteworthy that the small company archetype shows the lowest variance (0.56), consistent with the design decision described in section 4.3.2: low viability in this archetype is primarily attributable to the structural financial penalty rather than to fluctuating performance signals.

Trajectory Shape and Linearity. A characteristic of the synthetic generation procedure that has direct implications for model selection is the temporal dynamics of the score trajectories. Pearson correlation coefficients between `month_idx` and `overall_score` were computed per project to assess linearity. The median correlation is $R = -0.887$, and 71 of 150 projects (47 %) exhibit $|R| > 0.9$, indicating that a large proportion of synthetic trajectories are strongly linear with a negative slope. Figure 7.1 illustrates these trajectories grouped by archetype. The downward trend has a structural cause rooted in the financial scoring component (section 4.2.3): as the remaining project duration T decreases month by month, the ongoing integration cost $C_{\text{integration}}$ shrinks proportionally, while the one-time conversion cost $C_{\text{conversion}}$ remains fixed. Consequently, the total expected inner source cost approaches the cost of alternative development from above, and the relative financial advantage of inner sourcing narrows continuously toward project completion. A switch to inner source with only a few months remaining is thus penalized financially, which is the intended behavior of the scoring model. Only 21 projects (14 %) exhibit $|R| < 0.5$, and no project produces a monotonically increasing score trajectory. This pronounced linear structure is a consequence of the additive noise-perturbed drift formulation used in the synthetic generator and represents an important boundary condition for interpreting model performance results, as it systematically favors linear models over non-linear approaches. This issue is discussed in depth in section 7.4.

7.3 Machine Learning Model Performance

This section reports the quantitative performance of the two evaluated models on the held-out test set. Results are presented at two levels of granularity: overall aggregate metrics and per-archetype breakdowns.

Overall Aggregate Performance. Table 7.3 reports aggregate MAE, RMSE, and R^2 across all 126 forecast steps from the 14 held-out test projects. Linear Regression achieves the best performance overall, attaining an MAE of 0.131, an

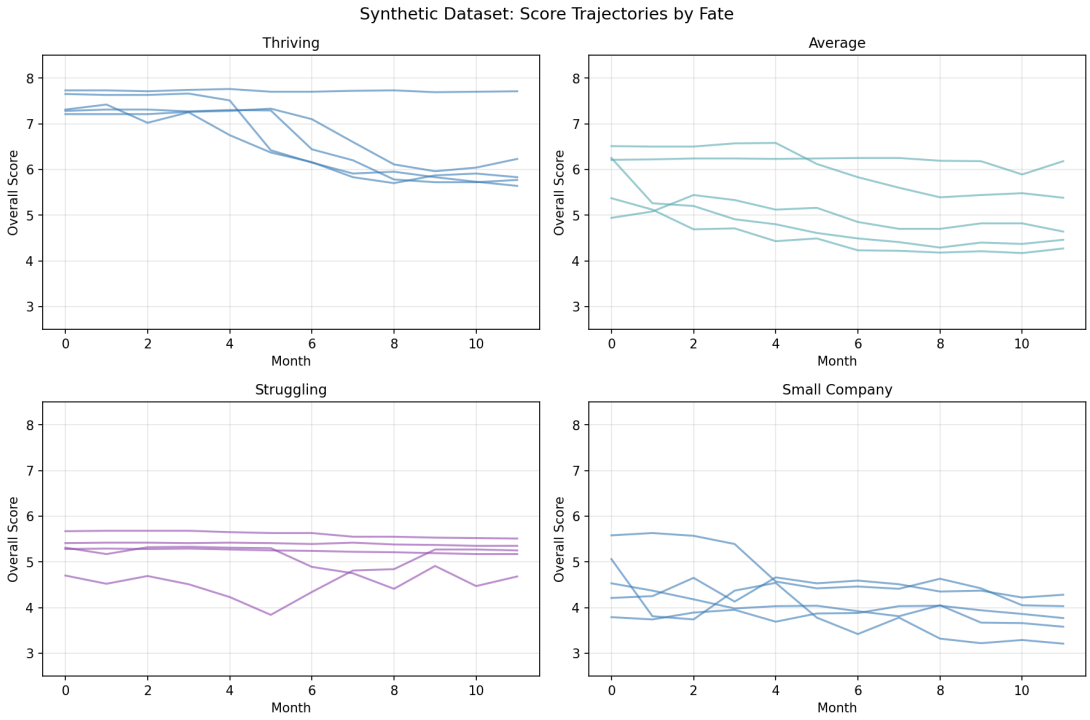

Figure 7.1: Overall viability score $\mathcal{V}$ trajectories of all 150 synthetic projects grouped by fate archetype. Archetype separation becomes visible from approximately month 4 onward, coinciding with the onset of the adoption phase (c.f. section 4.3.3).

RMSE of 0.171, and an R^2 of 0.980. The LightGBM model yields an MAE of 0.414 and an RMSE of 0.532, corresponding to an R^2 of 0.802. An illustration of the difference between the two models can be found in fig. 7.2.

Model	**MAE**	**RMSE**	R^2
LightGBM	0.4139	0.5323	0.8023
Linear Regression	0.1307	0.1709	0.9796

Table 7.3: Aggregate model performance on 14 held-out test projects (126 forecast steps, months 7–11).

The gap between LightGBM and the linear baseline (0.361 RMSE points) is substantial in the context of the overall score range and warrants careful interpretation. As analyzed in section 7.2, the synthetic dataset exhibits predominantly linear score trajectories, which directly advantages linear models. This structural mismatch between the training distribution and the non-linear inductive bias of gradient-boosted trees is the primary explanatory factor for the performance

differential.

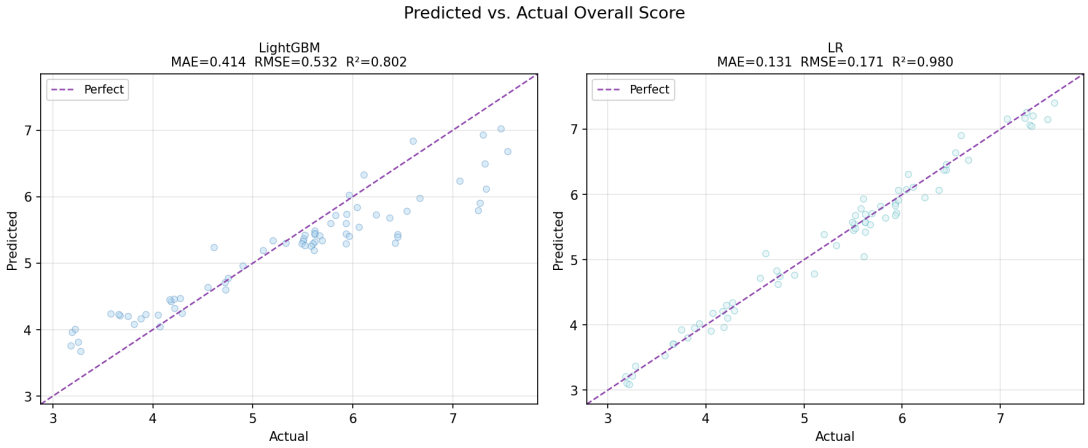

Figure 7.2: Predicted vs. ground-truth viability scores for all 126 forecast steps on the test set. Points lying on the diagonal indicate perfect predictions.

Performance by Archetype. Figure 7.3 shows the LightGBM performance broken down by archetype. The small company archetype yields the worst result ($R^2 = -2.533$), reflecting the model's inability to extrapolate the structurally flat, financially-dominated trajectories of that group. Struggling projects produce the best relative performance ($R^2 = 0.838$), as their moderately non-linear declining trajectories offer more learnable signal for the gradient-boosted model.

Forecast Error by Horizon. A characteristic failure mode of recursive multi-step forecasting is error accumulation: each prediction step relies on lag features that are themselves predictions, causing deviations from early steps to propagate

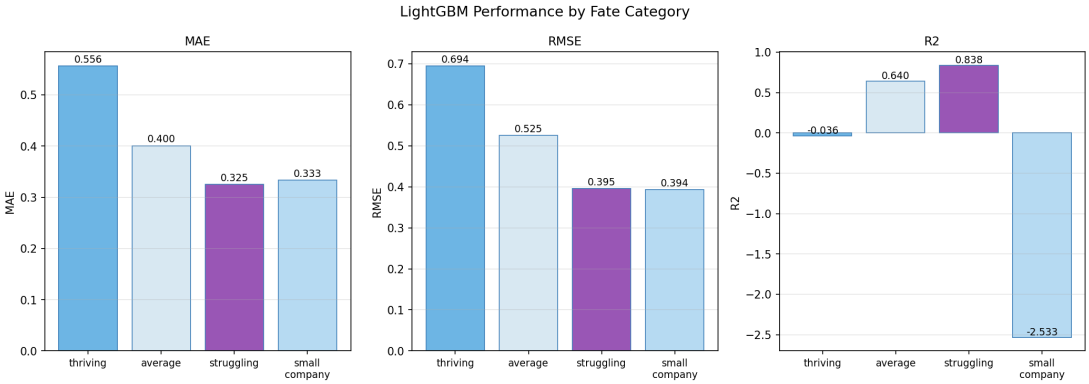

Figure 7.3: LightGBM performance by archetype. Negative R^2 values indicate that the model performs worse than the mean predictor for those archetypes.

into later forecast steps. The evaluation confirms this effect (see fig. 7.4). For the linear model, the MAE remains largely stable across forecast steps, consistent with the observation that linear trajectory extrapolation does not suffer from the same compounding behavior. The LightGBM model exhibits a more pronounced increase in forecast error as the horizon extends beyond step three, the point at which all three lag features are derived exclusively from predicted rather than observed values. This structural threshold at step three is an expected consequence of the lag depth of three used during feature engineering, as formalized in section 5.5.

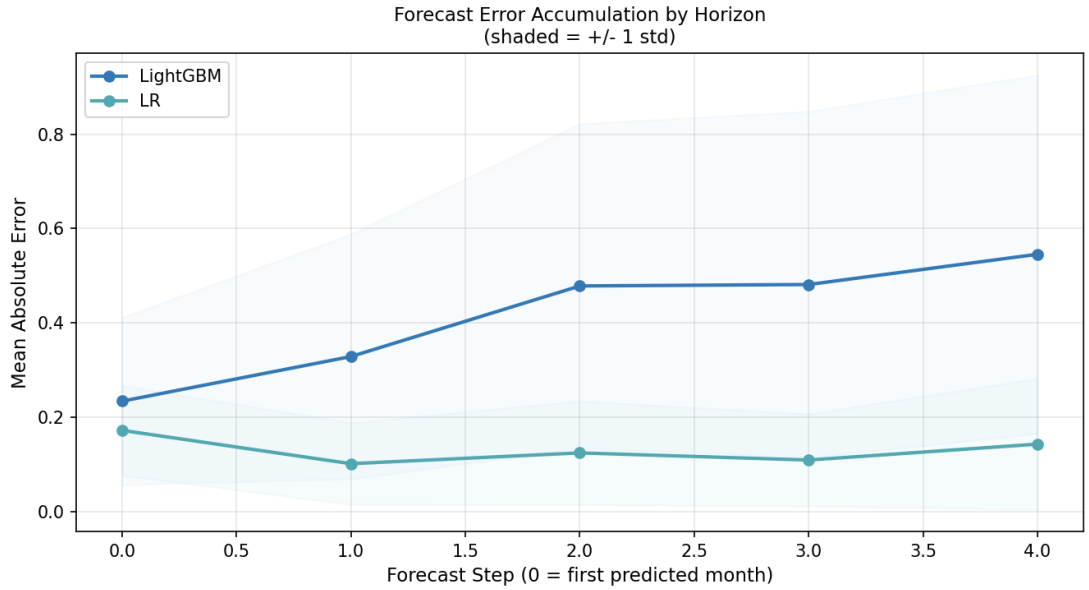

Figure 7.4: Mean absolute forecast error per step (months 7–11) for both models, averaged across all 14 held-out test projects.

7.4 Discussion and Limitations

The evaluation results demonstrate that the artifact fulfills its core research objective of producing an end-to-end pipeline capable of generating a labeled inner source viability dataset and training a predictive model from it. The pipeline successfully produces 1 800 well-formed, labeled observations that exhibit the intended archetype-consistent score distributions and temporal dynamics. Linear Regression achieves $R^2 = 0.980$ and MAE $= 0.131$ on the test set, with MAE values well below one score point confirming that the recursive forecasting procedure is operationally functional within the linear regime.

The central finding is the performance inversion between LightGBM and the linear baseline. This is not a general finding about inner source viability fore-

casting but a specific consequence of the synthetic data generation process: as documented in section 7.2, the score trajectories produced by the generator are predominantly linear , which artificially advantages linear models. Real-world inner source projects are expected to exhibit substantially more complex, non-linear trajectories driven by organizational changes, budget cycles, and shifts in contributor engagement. Under such conditions, a gradient-boosted tree model with sufficient training data would be expected to outperform the linear baseline, as it can capture interactions and threshold effects that a linear model cannot represent (Mouli et al., 2024). This interpretation is further supported by the archetype-level breakdown: LightGBM performs best on struggling projects, whose trajectories include more non-linear perturbations, and worst on the small company archetype, whose structurally flat trajectories offer no non-linear signal to exploit.

The evaluation is subject to four principal limitations:

- **Synthetic Dataset Linearity.** The dominant linearity of the synthetic trajectories artificially advantages linear models and prevents a fair evaluation of LightGBM. Future generator iterations should incorporate non-linear phase transitions and event-driven shocks to produce more realistic trajectory distributions.
- **Small Dataset Size.** With only 150 projects and 14 held out for testing, the dataset is a marginal sample size for a gradient-boosted tree model. The two small company test projects in particular provide insufficient statistical power to draw firm conclusions about that archetype.
- **Recursive Forecast Error Accumulation.** After forecast step three, all lag inputs rely on previous predictions. This causes errors to accumulate and increases errors in later steps. A sensitivity analysis varying the observation window k over $[3, 10]$ months would characterize this trade-off.
- **No Real-World Validation.** The dataset is entirely synthetic and the artifact has not been validated against observed industry projects. The reported performance is therefore not directly transferable to real-world deployment without empirical validation against a labeled dataset from industry partners.

8 Conclusions

This thesis presented a novel system for predicting the inner source viability of software projects using GitHub signals, user-provided inputs, and ML. The following sections summarize the main contributions of this work, answer the research questions posed at the outset, and outline directions for future research.

8.1 Summary of Contributions

As described in the Introduction (chapter 1), this thesis follows the DSRM proposed by Peffers et al. (2007), structured around three research questions. The following summarizes how each was addressed and what it contributed.

RQ1. *What are the objectives for an inner source viability prediction system?*

In chapter 2, we established that despite the well-documented benefits of inner source adoption, organizations consistently lack data-driven instruments for assessing project viability prior to resource commitment. Grounding our work in the framework of Hirsch and Riehle (2022), we identified its central limitation—reliance on manual expert input—and consolidated three open problems in the research gap (section 2.4). Chapter 3 then derived the specific system objectives, which directly influenced all subsequent design decisions.

RQ2. *How can an inner source viability prediction system be conceptually designed and implemented?*

In chapters 4 and 5, three interdependent artifact components were developed. A viability scoring algorithm operationalizes the framework of Hirsch and Riehle (2022) into a continuous target variable by mapping repository-observable signals and user-provided inputs onto a scalar overall viability score. Since no labeled real-world inner source data exists, a synthetic dataset of 150 simulated projects was generated to provide a labeled dataset. A LightGBM regressor constitutes the predictive component, using recursive multistep forecasting.

RQ3. *To what extent does the developed system fulfill its objectives, and what are its limitations?*

In chapter 7, the artifact was evaluated against the objectives defined in chapter 3. Regarding fulfillment, the system meets its core objective: the pipeline is end-to-end functional, the scoring algorithm produces archetype-consistent viability trajectories suitable for supervised learning, and the forecasting component generalizes to unseen projects. Regarding limitations, the linear baseline outperforms LightGBM, a finding that reflects the predominantly linear structure of the synthetic dataset rather than a fundamental limitation of the model class. More broadly, the absence of real-world validation data constrains the generalizability of all findings, and empirical data acquisition remains the decisive prerequisite for any production deployment.

8.2 Future Work

The evaluation results and identified limitations directly motivate several directions for future research, ordered by their expected impact on system validity.

Real-World Data Acquisition. The most consequential next step is the replacement of the synthetic dataset with empirical data from actual inner source repositories. This would require instrumenting real projects with the repository-observable signals defined in chapter 5 and collecting ground-truth viability assessments from domain experts or project retrospectives. Even a small set of 30–50 well-documented inner source projects would likely yield more ecologically valid results than the 150 synthetic projects used here, as the non-linear, context-dependent trajectory dynamics absent from the synthetic data would become learnable.

Validation of the Viability Score. Independent of the machine learning component, the viability scoring algorithm itself requires empirical validation. The current score operationalizes the framework of Hirsch and Riehle (2022) into a continuous target variable, but whether this operationalization correctly ranks projects by their actual inner source outcomes has not been tested. A structured expert study could be conducted in which practitioners assess the viability of a set of known projects. Comparing their ratings with the computed scores would help determine whether the scoring model has sufficient construct validity to serve as a supervised learning target.

Non-Linear Model Evaluation on Larger Datasets. The underperformance of LightGBM relative to the linear baseline is a consequence of dataset size and synthetic linearity rather than an inherent property of gradient boosting

on this problem class. Once real-world data becomes available, a systematic comparison of model families—including LightGBM, Random Forest, and regularized linear models—could be conducted on datasets of at larger size. Such an analysis would provide more reliable conclusions about the degree of non-linearity present in viability curves and which model class best captures it.

Appendices

A JSON Schemas

```json
{
  "project_meta": {
    "loc": 18400,
    "hourly_rate_dollar": 95,
    "remaining_project_time": 6,
    "has_readme": true,
    ...
  },
  "strategic_value_factors": {
    "urgency": { "value": 4, "weight": 0.3 },
    ...
  },
  "human_factors": {
    "human_resource_availability": { "value": 7, "weight": 0.25 },
    ...
  },
  "performance_factors": {
    "quality_pain":          { "bug_ratio": 0.18, ... },
    "review_gap":            { "prs_with_review": 0.74, ... },
    "test_gap":              { "ci_fail_rate": 0.12 },
    "maintainability_risk": { "avg_loc": 210, ... },
    "flexibility":           { "modularity": 7, ... },
    "innovation":            { "project_age_months": 18, ... },
    "delivery_time":         { "issue_cycle_time": 14, ... },
    "questionnaire":         { "quality_criticality": 0.75, ... }
  },
  "financial_factors": {
    "contribution_cost":           { "avg_review_time_hours": 2.5, ...
    ↪  },
    "integration_cost_per_month": { "pr_review_time_hours": 1.8, ... },
    "maintenance_contribution":    { "bmd": 3200, ... },
    "alternative_development_cost": {
      "best_case": 42000, "avg_case": 55000, "worst_case": 71000
    }
  }
}
```

Listing A.1: Transformed input schema consumed by the `ViabilityScorer`.

```json
{
  "project_id": "P_001",
  "timestamp": "2024-01-01T00:00:00",
  "phase": "incubation",
  "project_type": "platform_component",
  "fate": "thriving",
  "data": {
    "github_data": {
      "loc": 18400,
      "has_readme": true,
      "has_contributing": true,
      "has_tests": true,
      "has_ci": true,
      "many_dependencies": false,
      "bug_ratio": 0.182,
      "bug_fix_time_days": 8.4,
      "reopened_issues_ratio": 0.031,
      ...
    },
    "user_input": {
      "hourly_rate_dollar": 95,
      "remaining_project_time": 6,
      "expected_contributors": 4,
      "urgency":   { "value": 4, "weight": 0.25 },
      ...
      "human_resource_availability":   7,
      ...
      "alternative_development_cost": {
        "best_case":  42000,
        "avg_case":   55000,
        "worst_case": 71000
      }
    }
  },
  "label": null
}
```

Listing A.2: Raw record layout produced by the `SyntheticDataGenerator`, with representative values. The `label` field is initially `null` and populated by the `DatasetLabeler`.

References

Ajila, S. A., & Wu, D. (2007). Empirical study of the effects of open source adoption on software development economics. *Journal of Systems and Software*, *80*(9), 1517–1529. https://doi.org/https://doi.org/10.1016/j.jss.2007.01.011

Bontempi, G., Ben Taieb, S., & Le Borgne, Y.-A. (2013). Machine learning strategies for time series forecasting. In M.-A. Aufaure & E. Zimányi (Eds.), *Business intelligence: Second european summer school, ebiss 2012, brussels, belgium, july 15-21, 2012, tutorial lectures* (pp. 62–77). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-36318-4_3

Buchner, S., & Riehle, D. (2023). A Research Model for the Economic Assessment of Inner Source Software Development. *Hawaii International Conference on System Sciences 2023 (HICSS-56)*. https://aisel.aisnet.org/hicss-56/cl/cross-org_and_cross-border_collaboration/2

Capraro, M., & Riehle, D. (2016). Inner Source Definition, Benefits, and Challenges. *ACM Comput. Surv.*, *49*(4), 67. https://doi.org/10.1145/2856821

Coelho, J., & Valente, M. T. (2017). Why modern open source projects fail. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 186–196. https://doi.org/10.1145/3106237.3106246

Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2008). Free/libre open-source software development: What we know and what we do not know. *ACM Comput. Surv.*, *44*(2). https://doi.org/10.1145/2089125.2089127

Dillon, C. (2025). *State of InnerSource 2025 Report* (tech. rep.). Zenodo. https://doi.org/10.5281/zenodo.17980340

Edison, H., Carroll, N., Morgan, L., & Conboy, K. (2020). Inner source software development: Current thinking and an agenda for future research. *Journal of Systems and Software*, *163*, 110520. https://doi.org/https://doi.org/10.1016/j.jss.2020.110520

Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data? In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho & A. Oh (Eds.), *Advances in neural information processing systems* (pp. 507–520, Vol. 35). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/

2022/file/0378c7692da36807bdec87ab043cdadc-Paper-Datasets_and_Benchmarks.pdf

Hall, T., & Rasheed, K. (2025). A survey of machine learning methods for time series prediction. *Applied Sciences*, *15*(11). https://doi.org/10.3390/app15115957

Hirsch, J., & Riehle, D. (2022). Management Accounting Concepts for Inner Source Software Engineering. In N. Carroll, A. Nguyen-Duc, X. Wang & V. Stray (Eds.), *Software Business* (pp. 101–116). Springer International Publishing.

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260. https://doi.org/10.1126/science.aaa8415

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014). The promises and perils of mining github. *Proceedings of the 11th Working Conference on Mining Software Repositories*, 92–101. https://doi.org/10.1145/2597073.2597074

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018). Statistical and machine learning forecasting methods: Concerns and ways forward. *PLOS ONE*, *13*(3), 1–26. https://doi.org/10.1371/journal.pone.0194889

Malcolm, D. G., Roseboom, J. H., Clark, C. E., & Fazar, W. (1959). Application of a technique for research and development program evaluation. *Operations Research*, *7*(5), 646–669. Retrieved March 11, 2026, from http://www.jstor.org/stable/167013

Maslej, N., Fattorini, L., Perrault, R., Gil, Y., Parli, V., Kariuki, N., Capstick, E., Reuel, A., Brynjolfsson, E., Etchemendy, J., Ligett, K., Lyons, T., Manyika, J., Niebles, J. C., Shoham, Y., Wald, R., Walsh, T., Hamrah, A., Santarlasci, L., … Oak, S. (2025). Artificial intelligence index report 2025. https://arxiv.org/abs/2504.07139

Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, *33*(1), 2–13. https://doi.org/10.1109/TSE.2007.256941

Mouli, K. C., Raghavendran, C. V., Rao, C. M., Ushasree, D., Indupriya, B., Vatin, N. I., & Negi, A. S. (2024). Performance analysis of linear and non-linear machine learning models for forecasting compressive strength of concrete. *Cogent Engineering*, *11*(1), 2368101. https://doi.org/10.1080/23311916.2024.2368101

Nassif, A. B., Azzeh, M., Capretz, L. F., & Ho, D. (2016). Neural network models for software development effort estimation: A comparative study. *Neural Computing and Applications*, *27*(8), 2369–2381. https://doi.org/10.1007/s00521-015-2127-1

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Riehle, D., Capraro, M., Kips, D., & Horn, L. (2016). Inner Source in Platform-Based Product Engineering. *IEEE Transactions on Software Engineering*, *42*(12), 1162–1177. https://doi.org/10.1109/TSE.2016.2554553

Stol, K.-J., Avgeriou, P., Babar, M. A., Lucas, Y., & Fitzgerald, B. (2014). Key factors for adopting inner source. *ACM Trans. Softw. Eng. Methodol.*, *23*(2). https://doi.org/10.1145/2533685

Toner, W., & Darlow, L. (2024). An analysis of linear time series forecasting models. https://arxiv.org/abs/2403.14587

Wan, Z., Xia, X., Zhang, Y., Lo, D., Zhou, D., Chen, Q., & Hassan, A. E. (2022). What motivates software practitioners to contribute to inner source? *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 132–144. https://doi.org/10.1145/3540250.3549148