A Process Mining-Based System For The Analysis and Prediction of Software Development Workflows

Antía Dorado[1], Iván Folgueira[1], Sofía Martín[1], Gonzalo Martín[1] ,
Álvaro Porto[1], Alejandro Ramos[1], John Wallace[1]
[1]Inverbis Analytics SL, Lugo, Spain

November 3, 2025

Abstract

CodeSight is an end-to-end system designed to anticipate deadline compliance in software development workflows. It captures development and deployment data directly from GitHub, transforming it into process mining logs for detailed analysis. From these logs, the system generates metrics and dashboards that provide actionable insights into PR activity patterns and workflow efficiency. Building on this structured representation, CodeSight employs an LSTM model that predicts remaining PR resolution times based on sequential activity traces and static features, enabling early identification of potential deadline breaches. In tests, the system demonstrates high precision and F1 scores in predicting deadline compliance, illustrating the value of integrating process mining with machine learning for proactive software project management.

1 Introduction

In modern software engineering, DevOps practices have become the cornerstone of achieving continuous integration and continuous delivery (CI/CD). However, as software systems and teams scale, the complexity of development pipelines increases, making it difficult to maintain visibility over the entire workflow, from initial code contributions to deployment in production environments. While DevOps metrics, such as those popularized by the DORA framework [DR23], provide a valuable high-level overview of performance, they fail to capture the fine-grained process dynamics that lead to inefficiencies or bottlenecks in practice.

Process mining has emerged as a data-driven discipline capable of discovering, monitoring, and improving processes based on event logs extracted from operational systems. Although it has been widely applied in business process management and enterprise systems, its use in software development and DevOps environments remains relatively limited. However, the potential is significant: By transforming development events (commits, pull requests, builds, and deployments) into analyzable process logs, organizations can gain an increased level of transparency over how their software delivery workflows actually behave.

In this work, we present **CodeSight**, a system designed to bridge this gap by integrating process mining and predictive analytics within a real DevOps context. The platform extracts, structures, and governs data from GitHub repositories, building event logs that capture the full life cycle of software delivery, from the first commit associated with a pull request to its successful deployment in a development environment. These logs are analyzed using process mining techniques to uncover workflow patterns, identify process variants, and quantify deviations from expected paths.

Beyond descriptive analysis, CodeSight incorporates predictive modeling to estimate development times and anticipate potential delays. By experimenting with machine learning models, we demonstrate how combining process mining with predictive analytics can transform raw DevOps data into actionable insights that improve software delivery performance and foster a data-driven development culture.

1

2   Related Work

DevOps research has been largely focused on measuring performance and organizational outcomes through key indicators such as deployment frequency, lead time for changes, change failure rate, and time to restore service [FHK18]. These so-called DORA metrics have become the industry standard for assessing software delivery performance. However, while providing high-level benchmarks, these can be misused [DR23] and are limited in explaining the underlying causes of inefficiencies or delays. They are primarily outcome-based and lack the granularity required to understand the behavioral and procedural factors that shape these outcomes.

Process Mining [vdA16] has proven to be an effective method for discovering, monitoring, and improving processes using event data extracted from information systems. Its application in the domain of software engineering is still emerging but promising. Recent studies [NZR24] have shown that process mining can uncover complex workflows within software development environments, revealing handovers, rework loops, and deviations from expected paths. Despite this potential, practical applications in CI/CD pipelines remain limited due to the lack of standardized event data structures and the fragmented nature of development toolchains.

In recent years, research on *Predictive Process Monitoring* (PPM) has explored the use of machine learning techniques - such as gradient boosting, random forests, and deep neural networks - to predict process outcomes such as remaining time, success probability or compliance violations [TVLRD17, CCDW$^+$24]. These techniques have demonstrated the ability to anticipate process behavior by learning from historical execution data. When applied to DevOps environments, similar approaches can be leveraged to forecast delivery delays or detect process anomalies [BP24]. Nevertheless, few works have integrated predictive analytics with real, event-level DevOps data in a production context.

Overall, current research provides a strong foundation for measuring and improving DevOps performance, but the integration of process mining and predictive modeling remains underexplored. Existing work either focuses on high-level metrics (e.g., DORA) or on isolated predictive models without contextual process information. **CodeSight** addresses this gap by unifying event data from GitHub repositories, applying process mining techniques to discover real development workflows, and augmenting these insights with machine learning models for predictive analysis.

3   System Overview and Architecture

CodeSight is an integrated system designed to predict deadline compliance in software development processes by combining data collection, process mining, analytics, and deep learning. The system transforms raw development and deployment traces from GitHub into structured process data, enabling predictive insights over ongoing Pull Requests (PRs).

3.1   System Components

The architecture of CodeSight is divided into four main components:

1. **Data Acquisition Layer:** retrieves development and deployment data from the GitHub REST API, including PRs, commits, and workflow runs. This component ensures traceability by linking all artifacts (branches, commits, and actions) through their SHAs.

2. **Data Transformation Layer:** converts the raw data into event logs compliant with a standard process mining *CSV* format. Each PR is represented as a case, and activities correspond to discrete events (e.g., PR creation, commit, merge). This enables reconstruction of actual process flows and the timing between activities.

3. **Process Mining and Visualization Layer:** performs process mining, extracting elements such as workflow variants and rework patterns, as well as metrics such as lead time, review duration, or workflow success rates. These results are displayed through interactive dashboards and can be used for continuous improvement or benchmarking between repositories or teams.

4. **Predictive Layer:** implements the LSTM-based model that anticipates the remaining time of a PR, using both sequential (activity traces) and static features.

3.2 Workflow Overview

The complete workflow is summarized in Figure 1. Raw GitHub data are collected and normalized, transformed into event logs, analyzed via process mining, and finally used to feed metrics dashboards and as training input for the LSTM predictor. This pipeline supports both descriptive and predictive analytics, closing the loop between process observation and actionable forecasting.
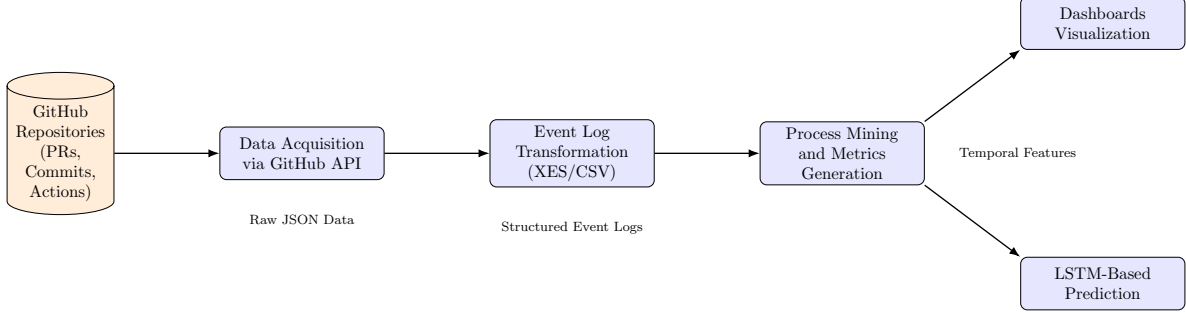

Figure 1: CodeSight architecture and data processing workflow.

Beyond predictive analytics, the architecture of **CodeSight** supports a continuous improvement loop for software development workflows. Predictions and process performance indicators are fed back into the organization through dashboards, alerts, and periodic reports. These insights enable development teams to identify potential deadline violations early, rebalance workloads, and refine branching or review strategies.

By combining *process mining*, *machine learning*, and *real-time feedback*, CodeSight acts not only as a monitoring system but also as a decision-support tool that enhances both operational efficiency and software quality across development pipelines.

4 Data Acquisition and Preparation

The **CodeSight** system collects and transforms information directly from *GitHub* repositories through its public REST API. The goal of this stage is to build a complete record of development and deployment activities that can be analyzed using process mining techniques. The following subsections describe the acquisition and transformation workflow.

4.1 Data Extraction from GitHub

Data are retrieved via authenticated API calls, focusing on *pull requests* (PRs), their associated *commits*, and *workflow runs* (CI/CD executions) from a specific repository branch.

4.1.1 Pull Requests

The initial extraction is performed using the endpoint:

```
https://api.github.com/repos/{OWNER}/{REPO}/pulls
```

All PRs are requested (`state=all`), including open, closed, and draft ones. The response provides core information about each PR:

- **id, number, title**: unique identifiers and PR description.
- **login**: user who created the PR.
- **head_ref**, **head_sha**, **merge_commit_sha**: source branch and related commit SHAs.
- **base_ref**: target branch (typically the main branch).
- **created_at**, **merged_at**, **closed_at**: key temporal milestones.

- **state**, **draft**: operational status and draft flag.

- **assignees**, **requested_reviewers**: developers and reviewers involved.

- **commits_url**: link to detailed commit information.

Additional metadata are obtained through an individual PR call:

```
https://api.github.com/repos/{OWNER}/{REPO}/pulls/{pr_number}
```

This endpoint provides complementary attributes such as **labels**, **merged_by**, **commits**, **additions**, **deletions**, and **changed_files**, offering a more granular view of the PR's scope and authoring context.

4.1.2 Commits Associated with Pull Requests

Commit-level information is retrieved via:

```
https://api.github.com/repos/{OWNER}/{REPO}/pulls/{pr_number}/commits
```

For each commit, the following attributes are collected: **commit_sha**, **date**, **message**, and **author**. Since this service does not include file details, an additional request is made for each commit:

```
https://api.github.com/repos/{OWNER}/{REPO}/commits/{commit_sha}
```

This allows extraction of modified file extensions and types, enabling analysis of the technical impact of each change (e.g., code, documentation, or configuration edits).

4.1.3 Workflow Runs (GitHub Actions)

To capture continuous integration and deployment (CI/CD) events, workflow run data are extracted through:

```
https://api.github.com/repos/{OWNER}/{REPO}/actions/runs
```

For each run, the following information is retrieved: **id**, **name**, **head_sha**, **event**, **status**, **conclusion**, **run_attempt**, **run_started_at**, **updated_at**, **created_at**, and **triggering_actor**. These data link workflow executions with corresponding commits or PRs and enable calculation of metrics such as duration, success rate, and failure patterns.

4.2 Data examples

For our own internal tests, we extracted data from our own process mining platform development repositories in Github, namely *frontend* and *backend*, which were loaded into three CSV files that we describe as follows.

Dataset: exported_commits_inverbisanalytics_frontend.csv

This file contains the commit history of the frontend repository. Each row represents an individual commit performed in a branch associated with a Pull Request (PR) identified by `pr_id`. The data allow analysis of source code evolution, change authorship, file types modified, and the temporal sequence of development, as Table 1 shows.

- **Size:** 465 rows × 6 columns

- **Temporal coverage:** 2020-11-03 – 2025-08-12

Subsequent use: this dataset provides the "Commit" events in the process mining trace, linked to the same `pr_id` as PR opening or merging events.

Table 1: Commit dataset structure (frontend repository)

Column	Type	Description
`pr_id`	ID	Unique identifier of the Pull Request associated with the commit. Groups multiple commits belonging to the same integration request.
`commit_id_sha`	text	SHA hash of the Git commit. Unique per commit, links to the full repository history.
`commited_at`	datetime	Date and time when the commit was recorded in the repository. Used as the primary event timestamp.
`commit_title`	text	Short commit message summarizing the purpose or main change.
`commit_author`	text	User or author who performed the commit. May contain null values for automated or missing commits.
`filetypes`	text	List or set of modified file types (e.g., `.js`, `.vue`, `.css`, `.yaml`).

Dataset: exported_inv_frontend_some_missing.csv

This dataset contains Pull Request (PR) and workflow execution information from the frontend repository. Each row corresponds to a combination between a PR (`pr_id`) and a possible associated pipeline execution (`run_id`). It includes authorship, state, key timestamps (creation, merge, closure), and change metrics. Table 2 lists all the data fields we extract.

- **Size:** 1179 rows × 32 columns

- **Temporal coverage:** 2023-02-13 – 2025-08-27

Subsequent use: this dataset provides the "PR Opening", "PR Merge", "PR Closure", and workflow-related events.

Dataset: exported_inv_backend.csv

This dataset has the same structure as the previous one but corresponds to the backend repository. It also includes individual commits associated with PRs.

- **Size:** 1375 rows × 37 columns

- **Temporal coverage:** 2024-07-29 – 2025-06-04

Subsequent use: this dataset enables tracing of PR opening, closure, and merge events, as well as workflow executions and individual commits.

4.3 Data Transformation and Event Log Generation

The transformation process aims to structure the raw data exported from GitHub into a structured event log compatible with process mining tools.

This conversion is performed using a set of Python scripts organized into three functional blocks:

Trace Structure Generation. Date columns are converted to `datetime` format, and relevant attributes are selected for each type of table (commitments, PRs, and workflows). Then, the tabular data are transformed into a sequential event model, where each case (`pr_id`) can have multiple activities: PR opening, commits, workflow executions, merge, or closure. The procedure identifies all columns with the prefix `"Fch"` (event dates), groups records by `pr_id`, and for each detected date column creates a new row duplicating the PR metadata.

For each new row:

- The date value is stored in the `DATE` column.

Table 2: PR and workflow dataset structure (frontend repository)

Column	Type	Description
pr_id	ID	Internal identifier of the Pull Request (primary case key).
pr_number	numeric	PR number assigned by GitHub.
pr_title	text	Descriptive title summarizing the purpose of the PR.
pr_author	text	User who created the PR.
from_branch	text	Source branch from which the merge was proposed (author's branch).
head_sha	text	SHA hash of the last commit before merging.
merge_commit_sha	text	SHA of the merge commit, when the PR was successfully merged.
into_branch	text	Target branch where the PR is merged.
created_at_x	datetime	PR creation timestamp.
merged_at	datetime	Merge timestamp; null if closed without merge.
closed_at	datetime	PR closure date (either merged or cancelled).
state	text	Final state of the PR (open, closed, merged).
is_draft	boolean	Indicates whether the PR was marked as draft.
assignees	text	List of users assigned to the PR.
reviewers	text	List of requested reviewers.
merged_by	text	User who performed the merge action.
commits	integer	Total number of commits included in the PR.
additions	integer	Number of added lines.
deletions	integer	Number of deleted lines.
changed_files	integer	Number of modified files.
labels	text	Labels or tags applied to the PR.
run_id	ID	Identifier of the pipeline execution.
run_name	text	Workflow name.
run_pr_commit _head_sha	text	SHA of the commit associated with the workflow run.
event_trigger	text	Event type that triggered the pipeline execution.
status	text	Execution status.
conclusion	text	Final result (success, failure, etc.).
created_at_y	datetime	Creation timestamp of the run record.
run_attempt	integer	Attempt number of the execution.
run_started_at	datetime	Start time of the pipeline run.
duration_ms	numeric	Total pipeline duration in milliseconds.
actor_trigger	text	User or process that triggered the pipeline.

- The original column name is assigned to ACTIVITY, which is later translated into a readable activity name.

Finally, the resulting table is sorted chronologically by pr_id and DATE and reindexed. Each row thus represents an activity with its timestamp linked to the corresponding case.

Attribute Configuration and Selection In this phase, the columns for the final dataset are defined (identifier, date, activity type, and relevant attributes). Activity names extracted from the date columns are translated into more understandable terms (e.g., Fch commit → Commit, Fch apertura PR → PR Opening).

Execution and Integration of the Data Model The three original CSV files are loaded and the date columns are renamed with descriptive labels. The transformations described previously are applied and the resulting traces are merged into a single unified data set containing the following main fields:
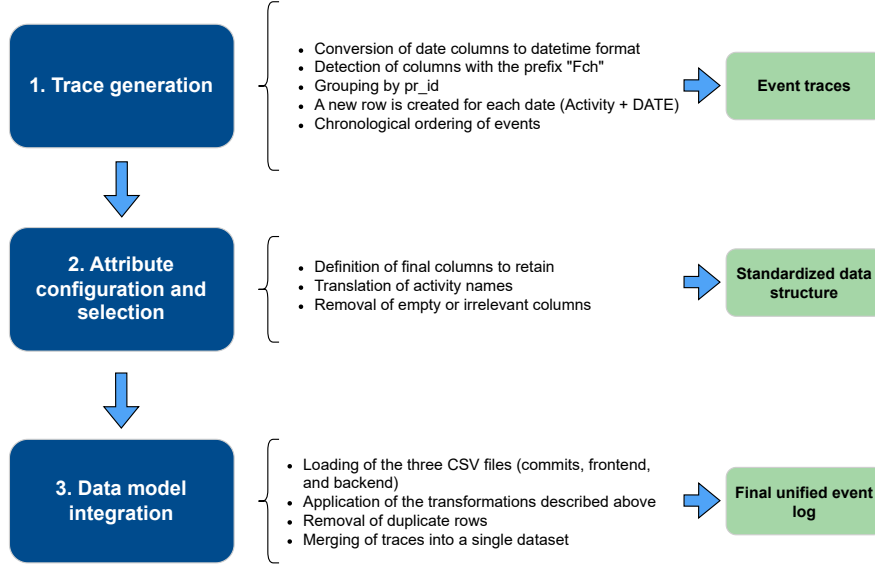
Figure 2: Conversion of raw data

- `pr_id` (case identifier)

- `ACTIVITY` (event)

- `DATE` (timestamp)

- User attributes: `commit_author`, `pr_author`, `merged_by`, etc.

- Additional context attributes: `from_branch`, `filetypes`, `state`, etc.

This process, summarized by Figure 2, standardizes heterogeneous structures into a coherent single model, translates and contextualizes activities for interpretation, and produces an event log directly exportable to process mining tools.

4.4 Resulting Dataset

After applying the transformations, a unified event log dataset is obtained, where each row represents an activity performed during a Pull Request (PR) lifecycle. The identifier `pr_id` acts as the case ID, grouping all activities associated with the same PR from opening to closure.

5 Process Mining and Dashboards

This section details the use of process mining and dashboarding techniques to analyze pull request event data. The aim is to reconstruct workflow structures, identify bottlenecks, and visualize key performance indicators through a Power BI dashboard. These insights serve both process optimization and as input for the predictive modeling presented in Section 6.

5.1 Mining the event log

The resulting event log is analyzed using our own **process mining** platform to reconstruct workflow structures and compute operational metrics, including:

- Activity and transition durations (e.g., time from PR creation to merge).

- Waiting and review times.

- Identification of bottlenecks and rework patterns.

- Performance indicators such as throughput and deadline compliance.

The process we obtained from our own data comprises a total of 835 cases and 271 process variants, with a temporal range from 2020-11-03 to 2025-08-27. The high number of different paths indicates substantial variability in process execution, which is typical in software development projects with multiple branches, authors, and automation workflows. The average process duration is 7 days and 14 hours, highlighting notable differences between cases depending on complexity or number of iterations.
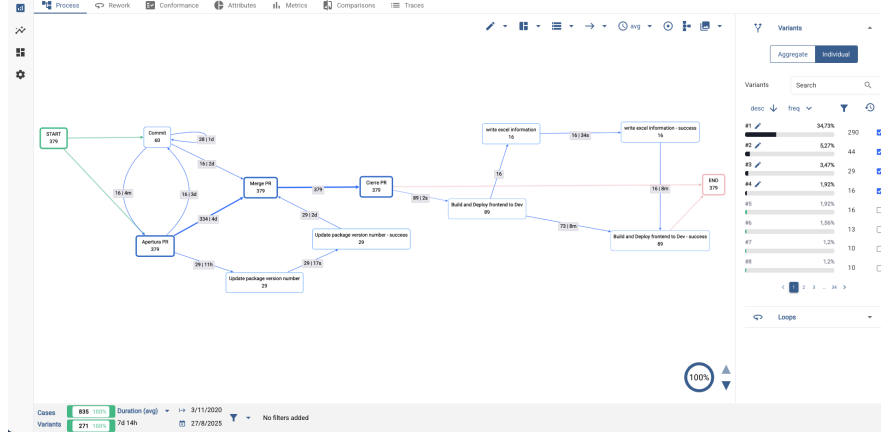

Figure 3: Simplified process model.

This process model, shown in Figure 3, enables identification of improvement patterns, analysis of dependencies between activities, and detection of potential bottlenecks in code review or automation phases. Likewise, the temporally structured traces enriched with derived metrics serve as the direct input for both the insights dashboard and the LSTM-based predictive model described in the next section.

5.2 Insights Dashboard

The CodeSight system includes a comprehensive Power BI dashboard designed to monitor software development and deployment processes through key performance indicators and process mining results. This dashboard feeds directly from the Inverbis Analytics process mining platform's API. The dashboard is organized into seven main sections, each providing a distinct analytical perspective:

1. **DORA Metrics:** This first tab presents the core DORA metrics (Deployment Frequency, Lead Time for Changes, Change Failure Rate, and Mean Time to Restore). These indicators provide a high-level view of the software delivery performance and reliability. These are shown in Figure 4.

2. **General Development Indicators:** The second tab, shown in Figure 5, provides general insights into development activity, including deployment frequency, lead time, average implementation time, and post-merge duration. It also includes distributions of merge events by author, offering visibility into developer participation and workload.

3. **Pull Request Activity:** This section focuses on Pull Request (PR) dynamics, presenting the number of PRs created, average review times, and distributions by author and duration. Additional visualizations highlight the average PR lifetime per author, supporting analysis of collaboration and code review efficiency.

4. **Process Variants and Visualization:** The fourth tab integrates process mining outputs, displaying the discovered process variants and an interactive visualization of the PR lifecycle. This allows users to identify deviations, redundant paths, or bottlenecks in the workflow.
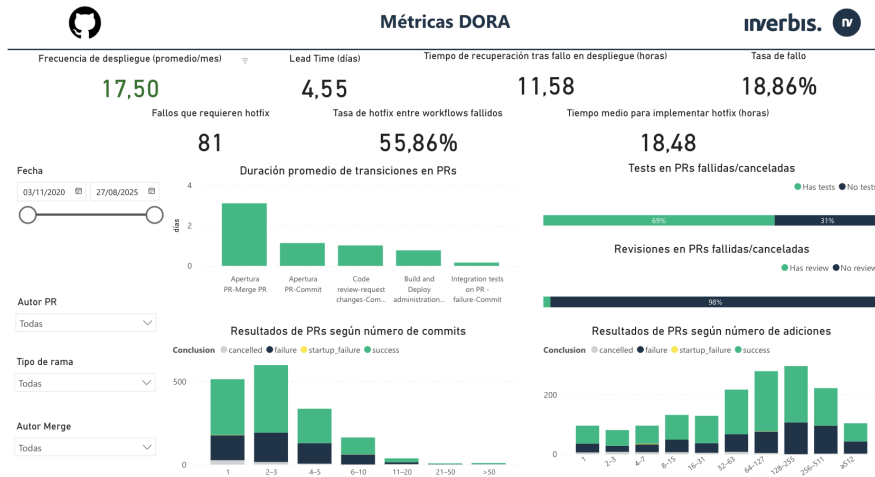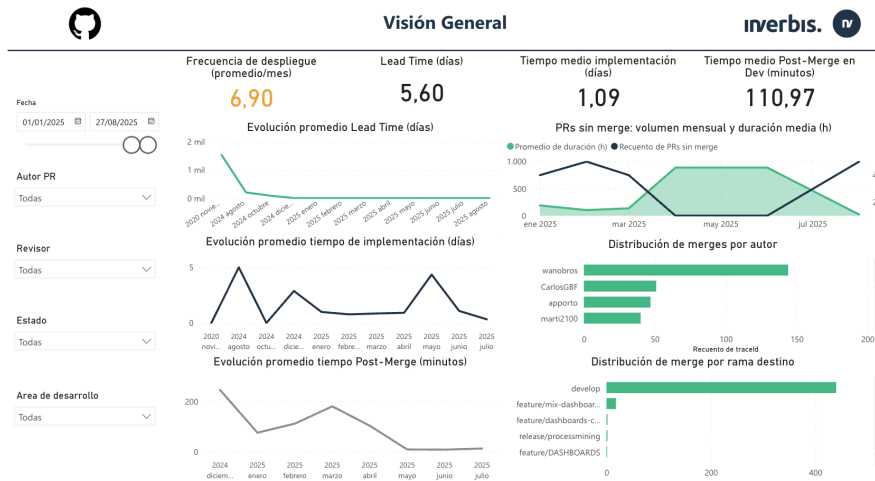
8

Figure 4: DORA Metrics view

Figure 5: General development indicators view.

5. **User-based Analysis:** This view provides a per-user analysis of activity across PRs, commits, and workflows, enabling identification of development patterns, workload balance, and team-level performance.

6. **Temporal Evolution of PRs:** The sixth tab explores the temporal trends of PR creation and integration, showing the evolution of activity over time along with complementary indicators such as average completion time and throughput.

7. **Deployment and Incident Metrics:** The final section focuses on deployment-related information, including failed workflow counts, incident rates, breakdowns by author and repository type, and the monthly evolution of incidents. This view connects development dynamics with operational outcomes, closing the DevOps feedback loop.

Together, these dashboards provide an integrated analytical environment for continuous monitoring of software development processes, combining process mining insights with operational metrics for decision support and process improvement.

6 LSTM for Deadline Compliance Prediction in Pull Requests

This section describes the methodology used to develop and evaluate a predictive model for deadline compliance in pull requests (PRs). The goal is to determine whether a PR will meet its assigned deadline based on historical process data and contextual attributes.

To achieve this, a Long Short-Term Memory (LSTM) neural network is implemented, as it can effectively capture the temporal dependencies within sequences of PR activities. We adopted the use of LSTM networks since these are well-suited for modeling temporal dependencies in sequential data, such as the ordered activities within a pull request lifecycle. The section covers all stages of the modeling pipeline, from data preparation to model evaluation.

First, the feature generation and preprocessing steps are described, including the creation of categorical, binary, and numerical variables, as well as methods to prevent data leakage and transform target variables. Then, the training pipeline is presented, encompassing dataset splitting, sequence padding, and temporal normalization. Finally, the LSTM model architecture, training configuration, and evaluation metrics are detailed, followed by the presentation and discussion of the results.

6.1 Feature Generation and Selection

Branch Attributes

- `from_branch_type`: type of the PR's source branch (fix, hotfix, bug, feature).

- `into_branch_type`: type of the target branch (feature, develop, release, staging).

- `process`: distinguishes between backend/frontend.

File Types

A binary column `has_X` is generated for each file type found in the original `filetypes` column, with a value of 1 if that type is modified and 0 otherwise.

Activity Encoding

Each activity name is converted into an integer (reserving 0 for padding). For each PR, a sequential list of indices of the executed activities is obtained.

Temporal Attributes

From the `DATE` column, the following features are created: `year`, `month`, `day`, `weekday`, and `is_weekend`.

Durations Between Activities

`transition_seconds` is calculated, a list of elapsed times between consecutive activities.

Labels and Deadline

Each PR is classified by complexity level: S, M, L, or XL. Each level is assigned a deadline in hours:

- S: 8 h

- M: 24 h

- L: 48 h

- XL: 72 h

6.2 Trace Truncation

To simulate the "in-progress" state of a PR and predict its evolution, each trace is truncated at a random position, generating new attributes:

- `cut`: index at which the trace is truncated.

- `truncated_activity_list`: activities up to the truncation point.

- `remaining_activity_list`: remaining activities.

- `truncated_transitions` and `remaining_transitions`: durations corresponding to each segment.

- `elapsed_time`: total time elapsed up to the truncation.

- `remaining_time`: remaining time until the actual closure.

- `activity`: index of the last activity recorded before truncation.

Other metrics derived from the prefix are also calculated:

- `prefix_len`: length of the prefix.

- `trunc_dt_mean`: mean time between events in the prefix.

6.3 Data Preparation and Feature Engineering

Preparation of Variables for Training

- **Prevention of data leakage:** Variables that reveal future information or are not constant along the trace (e.g., `trace_total_duration` or `remaining_transitions`) are removed.

- **Target variable transformation:** A logarithmic transformation (`log1p`) is applied to `remaining_time` to reduce skewness and stabilize training.

Feature Selection

- **Continuous numerical:** `elapsed_time`, `prefix_len`, and `trunc_dt_mean`.

- **Categorical:** calendar variables, branch type, and activity type.

- **Binary:** `has_` columns associated with events occurring before the cut.

A complete list of features (`all_features`) is compiled, and categorical variables are typed as strings for proper encoding.

Column Preprocessing

- **Numerical:** missing values are imputed with the median and scaled using `StandardScaler`.

- **Binary:** left unchanged.

- **Categorical:** encoded using `OneHotEncoder`.

These steps are integrated into a `ColumnTransformer`, which is later included in a general pipeline (`prep_pipeline`).

6.4　Dataset Splitting

The dataset is split approximately into:

- 70% training

- 15% validation

- 15% test

Explanatory variables (X_train, X_val, X_test) and transformed target labels (y_train_log, y_val_log, y_test_log), along with their original versions in seconds (y_train, y_val, y_test), are extracted.

The pipeline is fitted on the training set (fit_transform) and then applied to validation and test sets (transform) without retraining, ensuring consistency and avoiding data leakage.

6.5　Sequence Padding

To ensure uniform sequence lengths, the end of the activity list is padded with 0. The maximum length is set at the 95th percentile to prevent excessively long sequences from unnecessarily increasing the tensor size. For the transition list, a 0 is added at the beginning of each list (representing time before the first event).

6.6　Duration Transformation

Durations between activities are positive values that can vary greatly. They are transformed using a logarithm and standardized. The transformation also returns parameters (mu, sd) to apply later to the validation and test sets.

6.7　Model architecture

The architecture of the LSTM model is shown in Figure 6.

Model Inputs

- seq_in: sequence of activity IDs

- dt_in: transition times associated with those activities, of the same length

- stat_in: processed static features (numerical and categorical)

Sequential Branch: Activities + Transition Times

- **Embedding layer** (64 dimensions, ignores padding)

- **Transition mask**: ignores zeros

- **Concatenation** of embedding + duration

- **BiLSTM**: reads the full sequence (forward and backward) and learns temporal patterns. Each direction has 64 neurons, combined into a final vector of 128 values

- **Dropout = 0.15** randomly disables some connections during training to prevent overfitting

Sequential Information Compression

- Dense layer with 64 neurons and ReLU activation

Static Feature Branch

- Small dense network processing non-sequential information (numerical and categorical)
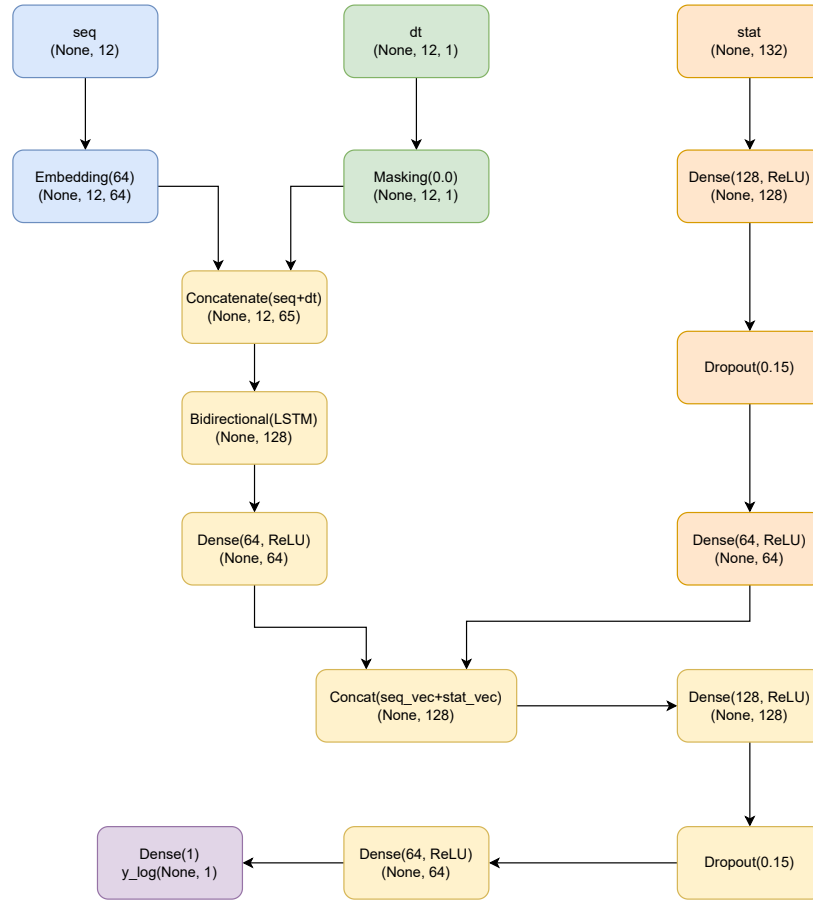
- Dense(128) - Dropout(0.15) - Dense(64)

Figure 6: Architecture of the LSTM model

Fusion of Both Branches

- The features are merged into a layer of 128 neurons.

- A more compact dense layer with 64 neurons is applied.

- A Dropout layer with a 15% rate is applied.

Prediction Head

- Combines all learned signals to predict the final value

- Outputs a continuous value representing the logarithm of the remaining time

6.8   Compilation and Training

- **Optimizer**: Adam with learning rate $1e-3$

- **Loss function**: mean squared error (MSE) on log-transformed values, penalizing proportionally large relative errors in long durations; mean absolute error (MAE) is also added for an intuitive reference during training.

- **Callbacks**:

    – `EarlyStopping`: stops training if validation loss does not improve for 8 consecutive epochs, restoring best weights.

- ReduceLROnPlateau: halves the learning rate when validation loss stagnates for 3 epochs, with a minimum of 1e−5, helping fine-tune the final stages of learning.

- The model is trained using all three input types (activity sequence, transition sequence, and static features).

- Maximum of 40 epochs, batch size of 64, validating on the validation set and applying defined callbacks to control convergence and overfitting.

6.9   Model Performance Evaluation

- Predictions are generated and the logarithmic transformation is reversed to obtain durations in seconds.

- MAE is calculated between predictions and actual values, converting to hours.

- Metrics are evaluated on the log scale; the coefficient of determination (R^2) is calculated to indicate the percentage of explained variance.

- Deadline compliance is evaluated by checking whether total elapsed time (`elapsed_time + predicted_duration`) stays within the maximum allowed limit (`Deadline_hours * 3600`).

- Compared with actual outcomes to obtain binary metrics: accuracy and F1-score, indicating the model's ability to correctly anticipate whether a case will meet the deadline.

6.10   Results

Regression Metrics

The target variable was trained on a logarithmic scale. The following metrics are reported in Table 3:

- **MAE(h)**: mean absolute error in hours (original scale)
- **R2(log)**: coefficient of determination on the log scale

Table 3: Regression performance metrics.

Set	MAE(h)	R2(log)
Train	10.108	0.938
Validation	11.627	0.843
Test	8.801	0.781

The model explains between 80% and 93% of the variance. The MAE of 8.8 hours on the test set represents roughly one working day of deviation, which is acceptable for practical deadline forecasting purposes.

Deadline Compliance Evaluation

Predicted remaining durations are transformed to seconds and compared with the deadline limit (`deadline_hours` × 3600). Accuracy and F1-score are reported in Table 4.

A case is predicted as deadline-compliant if:

$$\texttt{elapsed\_time} + \texttt{predicted\_duration} \leq \texttt{deadline\_hours} \times 3600$$

On the test set, the model achieves an accuracy of 0.944 and F1-score of 0.963, indicating a high ability to anticipate deadline compliance from incomplete traces.

The confusion matrix on the test set (Figure 7 shows very strong performance: 91 true positives (TP), 28 true negatives (TN), 3 false positives (FP), and 4 false negatives (FN).

- Predicted "Compliant" precision: $\approx 91/(91 + 3) = 0.968$

Table 4: Deadline compliance classification results (Accuracy and F1-score)

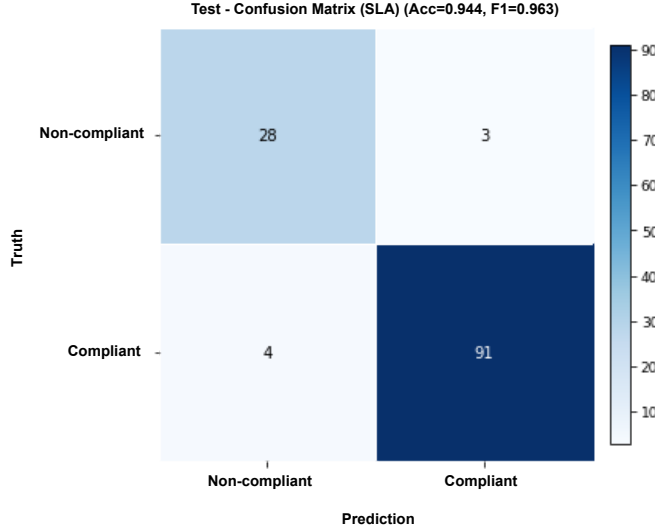| Set	Accuracy	F1
Train	0.940	0.959
Validation	0.881	0.911
Test	0.944	0.963

Figure 7: Test confusion matrix

- Sensitivity (recall) for "Compliant": $\approx 91/(91 + 4) = 0.958$

- Specificity for "Non-compliant": $\approx 28/(28 + 3) = 0.903$

Operationally, only 3 cases were falsely authorized as "Compliant" (low risk), and 4 were marked as "Non-compliant" despite actually meeting the deadline (slightly conservative criterion).

The consistency between the continuous remaining-time predictions and the binary deadline compliance outcomes indicates that the model successfully captures the underlying temporal dynamics of the process. This coherence supports the methodological decision to employ a unified continuous output for both regression and classification perspectives.

7 Conclusions and Future Work

This work presents *CodeSight*, a system that integrates data acquisition, process mining, metric generation, and predictive modeling to analyze software development workflows. The system demonstrates how process-oriented representations of software development activities—derived from GitHub repositories—can be leveraged to build interpretable metrics and predictive models that anticipate project outcomes.

The results obtained confirm that the approach is capable of modeling complex temporal and structural patterns within development processes, achieving high predictive performance when estimating task completion times and deadline compliance. Moreover, the combination of process mining with LSTM-based predictive models provides a comprehensive view that supports both operational monitoring and forward-looking insights.

However, several challenges remain open for future work. First, companies rely on different code repository services (e.g., GitLab, Bitbucket, Azure DevOps), each with its own API structure and data semantics. Extending *CodeSight* to integrate multiple platforms would enhance its generalization and applicability across heterogeneous environments. Second, even within the same platform, development

teams often organize their workflows differently—using distinct branching strategies, labels, or issue linking conventions. Therefore, designing a flexible data extraction layer capable of adapting to diverse repository structures will be essential to ensure the robustness and portability of the approach.

Future work will focus on three main directions: (i) expanding data integration to additional repository services and CI/CD systems; (ii) automating the mapping of repository events into standardized process logs; and (iii) incorporating explainability mechanisms into predictive models to facilitate their adoption in industrial environments. (iv) splitting the prediction approach into development and deployment for more specific estimations of both stages.

Ultimately, *CodeSight* represents a step toward process intelligence for DevOps, enabling organizations to continuously monitor, understand, and predict the behavior of their development pipelines through data-driven insights.

Acknowledgements

References

[BP24]   Peerachai Banyongrakkul and Suronapee Phoomvuthisarn. Deeppull: Deep learning-based approach for predicting reopening, decision, and lifetime of pull requests on github open-source projects. In Hans-Georg Fill, Francisco José Domínguez Mayo, Marten van Sinderen, and Leszek A. Maciaszek, editors, *Software Technologies*, pages 100–123, Cham, 2024. Springer Nature Switzerland.

[CCDW+24] Paolo Ceravolo, Marco Comuzzi, Jochen De Weerdt, Chiara Di Francescomarino, and Fabrizio Maria Maggi. Predictive process monitoring: concepts, challenges, and future research directions. *Process Science*, 1:2, 2024.

[DR23]   DORA Devops and Research. Accelerate state of devops report 2023. Technical report, Google, 2023.

[FHK18]   Nicole Forsgren, Jez Humble, and Gene Kim. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press, 2018.

[NZR24]   Ana F. Nogueira and Mário Zenha Rela. Process mining software engineering practices: A case study for deployment pipelines. *Information and Software Technology*, 168:107392, 2024.

[TVLRD17]   Niek Tax, Igor Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with lstm neural networks. In *Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 477–492. Springer, 2017.

[vdA16]   Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. Springer, 2 edition, 2016.