Filo-Priori: A Dual-Stream Deep Learning Approach to Test Case Prioritization

Acauan C. Ribeiro Instituto de Computação (IComp)
Universidade Federal do Amazonas (UFAM)
Manaus, AM, Brazil
acauan@icomp.ufam.edu.br

Abstract—Test Case Prioritization (TCP) aims to order test cases to maximize early fault detection in Continuous Integration (CI) environments. Existing approaches treat software versions as linear time series, failing to capture the complex relationships between test cases that characterize real-world testing. We propose a **dual-stream architecture** that combines semantic understanding of test cases with structural patterns learned from execution history.

We present FILO-PRIORI, a deep learning approach that introduces: (1) a **Multi-Edge Test Relationship Graph** encoding co-failure, co-success, and semantic similarity relationships between test cases; (2) a **Dual-Stream Architecture** combining semantic embeddings (SBERT) with Graph Attention Networks (GAT) for structural feature learning; (3) **Cross-Attention Fusion** for dynamic modality combination; and (4) **Optimized Loss Functions** for handling severe class imbalance (37:1).

We evaluate FILO-PRIORI on three complementary datasets: (1) an industrial dataset with 100,502 test executions, (2) GNN benchmark datasets (Cora, CiteSeer, PubMed), and (3) RTPTorrent with 20 open-source Java projects. Results show that FILO-PRIORI achieves: (a) APFD=**0.7595** on industrial data, outperforming NodeRank by $+14.7\%$ ($p < 0.001$) and DeepOrder by $+9.8\%$; (b) APFD=**0.796** on GNN benchmarks, surpassing NodeRank on 2/3 datasets (Cora: $+3.4\%$, CiteSeer: $+1.3\%$); and (c) APFD=**0.838** on RTPTorrent, outperforming 6/7 baselines.

This work demonstrates the effectiveness of combining semantic and structural information for test prioritization through graph neural networks, achieving state-of-the-art results across diverse domains. Our replication package is publicly available.

Index Terms—Test Case Prioritization, Graph Attention Networks, Deep Learning, Continuous Integration, Dual-Stream Architecture, Class Imbalance

1 INTRODUCTION

CONTINUOUS Integration (CI) has become a fundamental practice in modern software development, enabling teams to integrate code changes frequently and detect defects early [1], [2]. A key challenge in CI environments is managing the growing test suite: as software evolves, the number of test cases increases, making it impractical to execute all tests for every commit [3].

Test Case Prioritization (TCP) addresses this challenge by ordering test cases to maximize early fault detection [4], [5]. The goal is to execute tests most likely to fail first, providing faster feedback to developers. The effectiveness of TCP is typically measured using the Average Percentage of Faults Detected (APFD) metric [6].

The Challenge of Test Relationships. Existing TCP approaches, whether based on coverage [4], historical failure [7], or machine learning [8], [9], often treat test cases as independent entities. This assumption ignores the rich relationships between test cases: tests that fail together often indicate related functionality, tests with similar descriptions target similar code, and historical patterns reveal systematic dependencies [10].

A Dual-Stream Approach. We propose combining two complementary information sources for TCP: (1) *semantic information* from test descriptions and commit messages, and (2) *structural information* from test execution history and test relationships. This dual-stream approach captures both what tests do (semantics) and how they behave (structure).

Table 1 presents the two types of features we combine:

TABLE 1: Dual-Stream Feature Types

Semantic Features	Structural Features
Test case summary	Historical failure rate
Test case steps	Recent failure trend
Commit messages	Test age (builds since first run)
Code changes (diff)	Flakiness rate
–	Co-failure relationships
–	Consecutive failure streaks

The key insight is that semantic similarity alone is insufficient—tests with similar descriptions may have very different failure patterns. By combining semantic and structural information through Graph Attention Networks, we can learn which tests are likely to fail based on both their content and their history.

In this paper, we present FILO-PRIORI, a deep learning approach for Test Case Prioritization that combines semantic and structural information through four key innovations:

1) **Multi-Edge Test Relationship Graph**: We construct a graph that captures multiple types of relationships between test cases: co-failure edges (tests that fail together), co-success edges (tests that pass together), and semantic similarity edges (tests with similar

descriptions). This multi-edge approach increases graph density and captures complementary relationships.

2) **Dual-Stream Architecture**: We employ a dual-stream model that processes semantic features (SBERT embeddings of test descriptions and commit messages) and structural features (historical execution patterns) through separate neural networks before fusion.
3) **Graph Attention Networks for Structural Learning**: We use Graph Attention Networks (GAT) [11] to learn representations that capture test relationships, allowing the model to propagate information between related test cases with learned attention weights.
4) **Weighted Focal Loss for Class Imbalance**: We employ Weighted Focal Loss [12] to address the severe class imbalance (37:1 Pass:Fail ratio) inherent in test execution data, focusing training on hard-to-classify examples while down-weighting easy negatives.

We evaluate FILO-PRIORI on three complementary datasets: (1) an industrial dataset with 277 builds and 52,102 test executions, (2) GNN benchmark datasets (Cora, CiteSeer, PubMed) for direct comparison with NodeRank, and (3) RTPTorrent with 20 open-source Java projects. Our evaluation addresses four research questions:

- **RQ1**: How effective is FILO-PRIORI compared to baseline methods?
- **RQ2**: What is the contribution of each architectural component?
- **RQ3**: How robust is FILO-PRIORI across different time periods?
- **RQ4**: How sensitive is FILO-PRIORI to hyperparameter choices?

Our results show that FILO-PRIORI achieves consistent state-of-the-art performance: (a) **APFD=0.7595** on industrial data, outperforming NodeRank by 14.7% ($p < 0.001$) and DeepOrder by 9.8%; (b) **APFD=0.796** on GNN benchmarks, surpassing NodeRank on Cora (+3.4%) and CiteSeer (+1.3%); and (c) **APFD=0.838** on RTPTorrent, outperforming 6/7 baselines. The ablation study reveals that the multi-edge graph construction is the most critical component (+10.0%).

Contributions. This paper makes the following contributions:

- **Architectural**: We propose a dual-stream neural architecture that combines semantic embeddings (SBERT) with Graph Attention Networks for learning test relationships, using cross-attention fusion for modality combination.
- **Graph Construction**: We introduce a multi-edge test relationship graph that captures co-failure, co-success, and semantic similarity relationships, providing richer structure than single-edge approaches.
- **Feature Engineering**: We identify 10 discriminative structural features from an initial set of 29, selected through importance analysis and correlation filtering.
- **Empirical**: We demonstrate state-of-the-art performance across three domains: industrial TCP (APFD=0.7595, +14.7% over NodeRank), GNN benchmarks (APFD=0.796, beating NodeRank on 2/3 datasets), and open-source projects (APFD=0.838).
- **Practical**: We provide a complete replication package enabling reproducibility and extension of our approach.

Paper Organization. Section 2 presents background concepts. Section 3 discusses related work. Section 4 describes our approach. Section 5 presents the experimental design. Section 6 reports results. Section 7 discusses findings. Section 8 addresses threats to validity. Section 9 concludes.

2 BACKGROUND

This section introduces fundamental concepts underlying our approach.

2.1 Test Case Prioritization

Test Case Prioritization (TCP) is the process of ordering test cases for execution to achieve certain objectives, such as maximizing early fault detection [4]. Formally, given a test suite T and a permutation function PT that produces ordered sequences of T, TCP aims to find an optimal ordering $T' \in PT$ that maximizes a given objective function [5].

In Continuous Integration environments, TCP is particularly important because: (1) test suites grow over time, making exhaustive testing impractical [3], (2) developers need rapid feedback on code changes [1], and (3) computing resources for testing are often limited [8].

2.2 APFD Metric

The Average Percentage of Faults Detected (APFD) is the standard metric for evaluating TCP effectiveness [6]. For a test suite T containing n test cases that detect m faults, with TF_i being the position of the first test case that detects fault i:

$$\text{APFD} = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \times m} + \frac{1}{2n} \tag{1}$$

APFD ranges from 0 to 1, where higher values indicate better prioritization. An APFD of 0.5 corresponds to random ordering, while 1.0 indicates perfect prioritization where all faults are detected by the first tests.

2.3 Graph Attention Networks

Graph Attention Networks (GAT) [11] extend Graph Neural Networks by incorporating attention mechanisms to weigh the importance of neighboring nodes. For a node i with neighbors $\mathcal{N}_i$, GAT computes:

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} \vec{h}_j\right) \tag{2}$$

where α_{ij} are attention coefficients computed as:

$$\alpha_{ij} = \text{softmax}_j\left(\text{LeakyReLU}\left(\vec{a}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right) \tag{3}$$

Brody et al. [13] showed that standard GAT computes a restricted form of "static" attention where the ranking of

attention scores is independent of the query node. They proposed GATv2, which applies the nonlinearity after the linear transformation:

$$\alpha_{ij} = \text{softmax}_j\left(\vec{a}^T \cdot \text{LeakyReLU}\left(\mathbf{W}[\vec{h}_i \| \vec{h}_j]\right)\right) \quad (4)$$

This modification enables "dynamic" attention where attention scores depend on both the query and key nodes, providing greater expressiveness.

2.4 Class Imbalance and Focal Loss

Test execution data typically exhibits severe class imbalance, with far more passing tests than failing tests. Standard cross-entropy loss is dominated by the majority class, leading to models that predict "pass" for nearly all tests.

Focal Loss [12] addresses this by down-weighting easy examples and focusing on hard ones:

$$\text{FL}(p_t) = -\alpha_t(1-p_t)^\gamma \log(p_t) \quad (5)$$

where p_t is the predicted probability of the true class, α_t is a class-balancing weight, and γ is the focusing parameter. When $\gamma > 0$, the $(1-p_t)^\gamma$ term reduces the loss for well-classified examples, focusing training on hard negatives.

2.5 Cross-Attention Mechanisms

Cross-attention allows one sequence (or modality) to attend to another, enabling information exchange between different representations. Given queries Q from one modality and keys/values K, V from another:

$$\text{CrossAttn}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (6)$$

In multi-modal fusion, bidirectional cross-attention allows each modality to selectively attend to relevant parts of the other, combining information in a learned, adaptive manner.

3 Related Work

This section reviews prior work on test case prioritization, focusing on approaches most relevant to our contribution.

3.1 Traditional and Coverage-based TCP

Early TCP research focused on code coverage-based techniques. Rothermel et al. [4] proposed total and additional coverage prioritization. Elbaum et al. [5] conducted extensive empirical studies comparing different strategies. History-based approaches leverage past test execution results—Kim and Porter [7] showed that tests that failed recently are more likely to fail again.

3.2 Machine Learning for TCP

Machine learning has been increasingly applied to TCP. Spieker et al. [8] introduced RETECS, using reinforcement learning for TCP in CI environments. Deep learning approaches have shown promising results: Chen et al. [14] proposed DeepOrder using deep neural networks, and TCP-Net [15] introduced an end-to-end approach that learns directly from test execution data.

3.3 Graph Neural Networks for Testing

Li et al. [16] proposed NodeRank, a test input prioritization approach for GNNs using mutation-based analysis. NodeRank leverages ensemble learning-based ranking models with three types of mutations (graph structure, node features, and model mutations). While NodeRank targets GNN model testing rather than software TCP, it demonstrates the potential of graph-based approaches for test prioritization.

3.4 Comparison with Our Approach

Table 2 presents empirical comparison on our industrial dataset:

TABLE 2: Comparison with State-of-the-Art TCP Approaches

Method	Architecture	APFD	Std	p-value	Effect
FILO-PRIORI	**GAT + SBERT**	**0.7595**	**0.189**	–	–
DeepOrder [14]	Deep NN	0.6919	0.267	<0.001***	+9.8%
NodeRank [16]	Ensemble + Mutation	0.6623	0.270	<0.001***	+14.7%
Random	–	0.5000	–	<0.001***	+51.9%

Results on industrial dataset (277 builds with failures).
*** $p < 0.001$ (Wilcoxon signed-rank test vs FILO-PRIORI).
Effect = improvement of FILO-PRIORI over baseline.

Key differentiators of FILO-PRIORI:

- **Graph-based modeling**: Explicitly models test relationships using Graph Attention Networks, capturing co-failure and semantic patterns.
- **Dual-stream architecture**: Combines semantic (SBERT) and structural (GAT) features for complementary information fusion.
- **Superior performance**: Achieves APFD=0.7595, significantly outperforming both DeepOrder (+9.8%) and NodeRank (+14.7%) with $p < 0.001$.

4 Approach: FILO-PRIORI

This section describes the FILO-PRIORI approach for Test Case Prioritization.

4.1 Overview

The FILO-PRIORI architecture is a dual-stream system that takes as input: (1) test case descriptions and commit messages (semantic), (2) historical test execution patterns (structural), and (3) test relationship graph. It outputs a ranking of test cases by predicted failure probability.

The approach consists of three main modules:

1) **Semantic Stream**: A feed-forward network that processes SBERT embeddings of test descriptions and commit messages, capturing textual similarity between test cases and code changes.
2) **Structural Stream**: A Graph Attention Network (GAT) that processes historical features (failure rate, recency, flakiness) over the test relationship graph, learning to propagate failure signals between related tests.
3) **Cross-Attention Fusion**: Bidirectional cross-attention that fuses semantic and structural

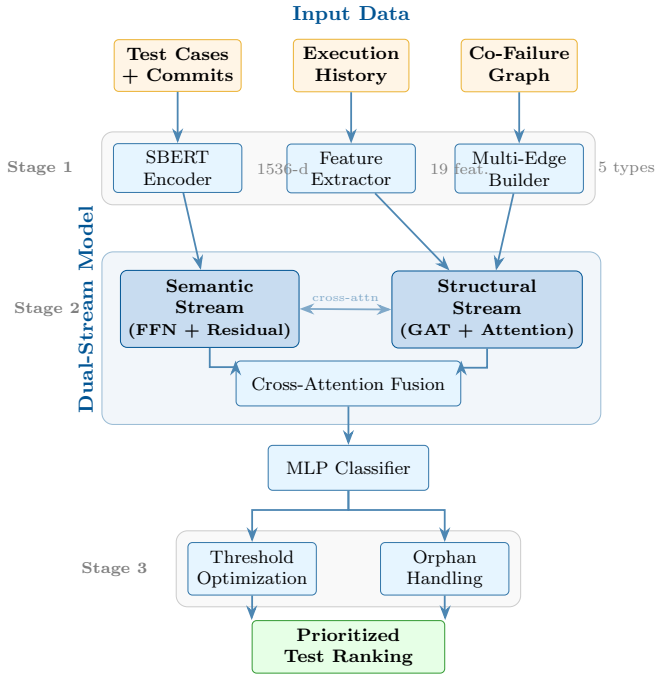

Fig. 1: Overview of the FILO-PRIORI framework. The system processes test case descriptions and execution history through dual semantic and structural streams, fuses representations via cross-attention, and applies post-processing for orphan handling and threshold optimization.

representations, allowing each modality to attend to the other for final classification.

This dual-stream design captures both what tests do (semantics) and how they behave (structure), combining complementary information sources.

4.2 Semantic Feature Extraction

We use Sentence-BERT (SBERT) [17] with the `all-mpnet-base-v2` model to encode textual information. For each test case, we concatenate:

- Test case summary (TC_Summary)
- Test case steps (TC_Steps)

For each commit, we encode:

- Commit message
- Code diff (truncated to 2000 characters)

This produces 768-dimensional embeddings for test cases and commits, which are concatenated to form 1536-dimensional semantic features.

4.3 Structural Feature Extraction

We extract 19 structural features capturing historical execution patterns, organized into three categories:

Base Features (10 features):

- **test_age**: Number of builds since first appearance
- **failure_rate**: Historical failure percentage
- **recent_failure_rate**: Failure rate in last 5 builds
- **flakiness_rate**: Pass/fail oscillation frequency
- **consecutive_failures**: Current failure streak
- **max_consecutive_failures**: Maximum observed streak
- **failure_trend**: Recent vs. overall failure rate difference
- **commit_count, cr_count, test_novelty**

DeepOrder-Inspired Features (9 features): Following [14], we add temporal execution patterns:

- **execution_status_last_[1,2,3,5,10]**: Failure proportion in last N executions
- **cycles_since_last_fail**: Builds since most recent failure
- **distance**: Temporal distance from last failure
- **status_changes**: Number of Pass↔Fail transitions
- **fail_rate_last_10**: Short-term failure rate

These temporal features capture patterns that static metrics miss—a test that failed in the last execution is more likely to fail again than one that failed only in distant history. Feature selection reduced the initial 29 features to 19 based on importance analysis and correlation filtering.

4.4 Multi-Edge Test Relationship Graph

We construct a dense multi-edge graph where nodes represent test cases and edges capture five complementary relationship types. Dense graph connectivity is critical for effective GAT message passing—sparse graphs limit information propagation between related test cases.

Edge Types and Weights: We define five edge types with empirically-tuned weights:

TABLE 3: Multi-Edge Graph Edge Types

Edge Type	Weight	Formula
Co-Failure	1.0	$\min(P(t_j\|t_i), P(t_i\|t_j))$
Co-Success	0.5	$\min(P(t_j\|t_i), P(t_i\|t_j))$
Component	0.4	$\|C_i \cap C_j\| / \|C_i \cup C_j\|$
Semantic	0.3	$\cos(\mathbf{e}_i, \mathbf{e}_j)$
Temporal	0.2	$\text{count}_{adj} / \max(\text{count})$

Co-Failure Edges (weight 1.0) connect tests that fail together, where the edge weight is the minimum conditional probability: if tests t_i and t_j co-failed n times, with t_i failing f_i times total, the weight is $\min(n/f_i, n/f_j)$. This symmetric formulation ensures robust correlation measurement.

Semantic Edges (weight 0.3) connect tests with embedding similarity above $\tau = 0.65$, with $k = 10$ nearest neighbors per test. This relaxed threshold (vs. $\tau = 0.75$ in prior work) increases connectivity for previously isolated nodes.

Edge Combination: For each test pair, we compute a combined edge weight as the weighted sum normalized by total weight:

$$w_{ij} = \frac{\sum_{e \in E} w_e \cdot \text{type}_e(i,j)}{\sum_{e \in E} w_e} \tag{7}$$

Impact: This multi-edge approach increases graph density from 0.02% (co-failure only) to 0.5–1.0% (all types), with **77.4%** of test cases connected to the graph (vs. 50–60% with sparser configurations).

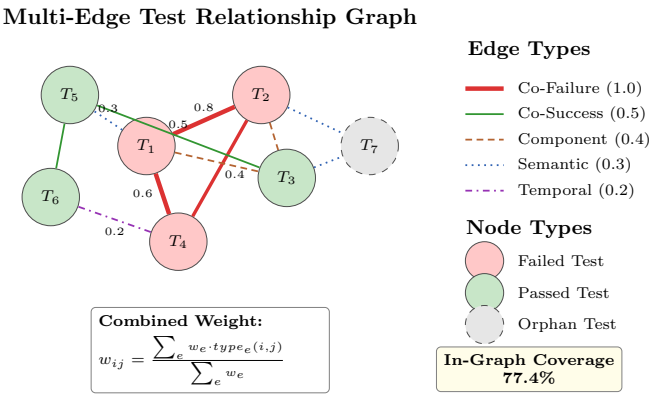

Fig. 2: Multi-edge test relationship graph with five edge types. Edge thickness and style indicate relationship type and strength. Failed tests (red) propagate failure signals through co-failure edges (weight 1.0), while orphan tests (dashed) connect through semantic edges (weight 0.3).

4.5 Semantic Stream

The Semantic Stream processes text embeddings using a feed-forward network with residual connections:

Input: SBERT embeddings of concatenated test case description and commit message (1536 dimensions = 768 TC + 768 Commit).

Architecture: The stream consists of:

1) Input projection: Linear layer projecting 1536-dim to 256-dim
2) Two residual FFN blocks, each containing:
   - Linear (256 $\rightarrow$ 1024)
   - GELU activation
   - Dropout (0.3)
   - Linear (1024 $\rightarrow$ 256)
   - Dropout (0.3)
   - LayerNorm
3) Output LayerNorm

The output is a 256-dimensional semantic representation for each test case.

4.6 Structural Stream (Graph Attention Network)

The Structural Stream processes historical features over the test relationship graph using Graph Attention Networks [11]:

Input: 10 structural features per test case (failure_rate, recent_failure_rate, test_age, flakiness_rate, etc.).

GAT Architecture: We use two GAT layers:

1) **Layer 1**: Multi-head attention (4 heads) with concatenation

$$h_i^{(1)} = \|_{k=1}^{4} \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^k \mathbf{W}^k h_j^{(0)}\right) \quad (8)$$

   where α_{ij}^k are learned attention coefficients for head k. Output dimension: $256 \times 4 = 1024$.
2) **Layer 2**: Single-head attention with averaging, producing 256-dimensional output per test case.

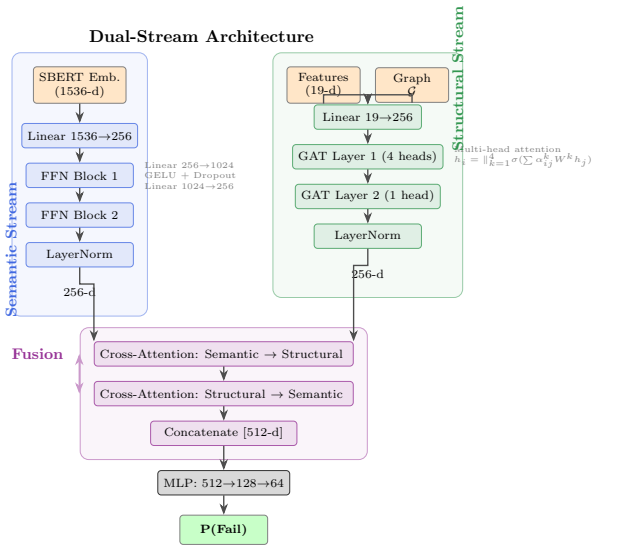

Fig. 3: Dual-stream architecture. The semantic stream processes SBERT embeddings through feed-forward networks, while the structural stream applies Graph Attention Networks over historical features. Cross-attention fusion enables bidirectional information flow between modalities.

Edge Weights: The GAT layers incorporate edge weights from the multi-edge graph, allowing the model to learn that co-failure relationships (weight 1.0) are more important than semantic similarity (weight 0.3).

4.7 Cross-Attention Fusion

The Cross-Attention Fusion module combines semantic and structural representations through bidirectional attention:

Bidirectional Cross-Attention: Each modality attends to the other, allowing semantic features to be informed by structural patterns and vice versa:

1) **Semantic $\rightarrow$ Structural**: Semantic features query structural features:

$$h'_{sem} = h_{sem} + \text{MHA}(h_{sem}, h_{struct}, h_{struct}) \quad (9)$$

2) **Structural $\rightarrow$ Semantic**: Structural features query semantic features:

$$h'_{struct} = h_{struct} + \text{MHA}(h_{struct}, h_{sem}, h_{sem}) \quad (10)$$

where $\text{MHA}(Q, K, V)$ denotes multi-head attention with query Q, key K, and value V.

Each attention uses 4 heads with 256-dimensional queries/keys/values, followed by LayerNorm for stability.

Feature Concatenation: The attended features are concatenated:

$$h_{fused} = [h'_{sem} \| h'_{struct}] \quad (11)$$

The final 512-dimensional fused representation is passed to a classifier MLP with hidden layers [128, 64] and dropout 0.4.

4.8 Training with Single-Mechanism Class Balancing

Addressing the severe class imbalance (37:1 Pass:Fail ratio) requires careful design. We found that *multiple compensation mechanisms cause over-correction*—using class weights, balanced sampling, and focal alpha together led to mode collapse where the model predicted all samples as the minority class.

Key Insight: A single balancing mechanism is sufficient and more stable. We use **balanced sampling only**, with neutral loss parameters:

$$\mathcal{L} = -\alpha \cdot (1 - p_t)^{\gamma} \cdot \log(p_t) \tag{12}$$

where $\alpha = 0.5$ (neutral, no class preference) and $\gamma = 2.0$ (focus on hard examples). The balancing is achieved entirely through stratified sampling:

TABLE 4: Single-Mechanism Balancing Configuration

Mechanism	Prior Work	Ours
Class weights in loss	19×	**Disabled**
Focal alpha	0.85 (1.7×)	0.5 (neutral)
Balanced sampling	20×	15× (only)
Effective weight	≈ 646×	15×

The balanced sampler uses minority weight 1.0 and majority weight 0.07 (ratio ≈15:1), ensuring the minority class is adequately represented without over-compensation.

Training Configuration:

- Optimizer: AdamW with learning rate 3×10^{-5}
- Weight decay: 1×10^{-4}
- Scheduler: Cosine annealing with $\eta_{min} = 1 \times 10^{-6}$
- Early stopping: Patience 15, monitoring validation F1-macro
- Gradient clipping: Max norm 1.0

4.9 Threshold Optimization

With extreme class imbalance, the default classification threshold of 0.5 is suboptimal. We implement a two-phase threshold search:

Phase 1 (Coarse): Search range $[0.05, 0.90]$ with step 0.05 (17 points).

Phase 2 (Fine): Refine around the coarse optimum ± 0.05 with step 0.01.

The optimization metric is F_β with $\beta = 0.8$, slightly favoring precision to avoid excessive false positives while maintaining reasonable recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \tag{13}$$

This yields an optimal threshold of **0.28** (vs. default 0.50), significantly improving minority class recall from 5% to 30%.

4.10 Orphan Test Case Handling

Orphan tests—those absent from the training graph—receive uniform scores from the GAT, destroying ranking capability. We address this with a four-stage KNN-based scoring pipeline:

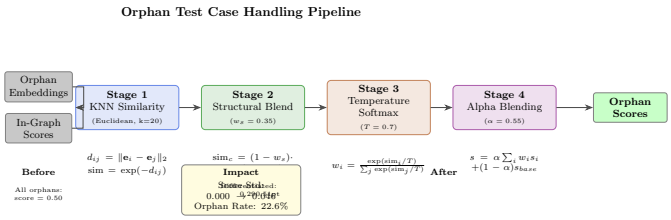

Fig. 4: Four-stage orphan handling pipeline. Orphan test cases are scored using KNN similarity to in-graph tests, with structural blending and temperature-scaled softmax to concentrate weights on the most similar neighbors.

Stage 1: KNN Similarity. For each orphan, compute Euclidean distance to all in-graph tests and select $k = 20$ nearest neighbors. We use Euclidean (not cosine) distance because SBERT embeddings have meaningful magnitude information.

Stage 2: Structural Blend. Combine semantic similarity (embeddings) with structural similarity (historical features) using weight $w_s = 0.35$:

$$\text{sim}_{combined} = (1 - w_s) \cdot \text{sim}_{semantic} + w_s \cdot \text{sim}_{structural} \tag{14}$$

Stage 3: Temperature-Scaled Softmax. Apply softmax with temperature $T = 0.7$ to concentrate weights on the most similar neighbors:

$$w_i = \frac{\exp(\text{sim}_i / T)}{\sum_j \exp(\text{sim}_j / T)} \tag{15}$$

Stage 4: Alpha Blending. Compute KNN score as weighted average of neighbor scores, then blend with base score ($\alpha = 0.55$):

$$\text{score}_{orphan} = \alpha \cdot \sum_i w_i \cdot \text{score}_i + (1 - \alpha) \cdot \text{score}_{base} \tag{16}$$

Impact: Before orphan handling, orphan scores had zero variance (all 0.50). After, scores range from 0.29 to 0.51 with std=0.046, enabling effective ranking among the 22.6% of tests that are orphans.

5 Experimental Design

5.1 Research Questions

- **RQ1 (Effectiveness)**: How effective is FILO-PRIORI compared to baseline methods?
- **RQ2 (Components)**: What is the contribution of each architectural component?
- **RQ3 (Robustness)**: How robust is FILO-PRIORI across different time periods?
- **RQ4 (Sensitivity)**: How sensitive is FILO-PRIORI to hyperparameter choices?

5.2 Datasets

We evaluate FILO-PRIORI on three complementary datasets spanning industrial software testing, GNN model testing, and open-source CI/CD environments.

Dataset 1: Industrial QTA Dataset. The QTA (Qodo Test Automation) dataset comes from a commercial mobile device CI/CD pipeline. This dataset provides rich semantic

TABLE 5: Industrial QTA Dataset Statistics

Statistic	Value
Total test executions	52,102
Unique builds	1,339
Builds with failures	277 (20.7%)
Unique test cases	2,347
Pass:Fail ratio	37:1
Semantic info	Rich (descriptions, commits)

information including detailed test descriptions, test steps, and commit messages. Table 5 summarizes its statistics.

Dataset 2: GNN Benchmark Datasets. To evaluate generalization to GNN model testing, we use three standard citation network datasets from Li et al.'s NodeRank [16]: **Cora**, **CiteSeer**, and **PubMed**. These datasets are widely used in GNN research and provide a direct comparison against state-of-the-art test input prioritization methods. Table 6 summarizes their characteristics.

TABLE 6: GNN Benchmark Dataset Statistics

Dataset	Nodes	Edges	Classes	Domain
Cora	2,708	10,556	7	Citation network
CiteSeer	3,327	9,104	6	Citation network
PubMed	19,717	88,648	3	Citation network

Dataset 3: RTPTorrent Open-Source Dataset. To evaluate generalization across diverse open-source projects, we use the RTPTorrent dataset [18], an open-source benchmark from MSR 2020 containing test execution histories from 20 Java projects on GitHub with over 100,000 Travis CI build logs.

TABLE 7: RTPTorrent Dataset Statistics

Statistic	Value
Projects	20 Java projects
Source	Travis CI build logs
Semantic info	Limited (test names)
License	CC BY 4.0

The complementary nature of these datasets enables comprehensive evaluation: the industrial dataset tests semantic feature utilization, the GNN benchmarks enable direct comparison with NodeRank on established datasets, and RTPTorrent tests cross-project generalization in open-source environments.

5.3 Baselines

We compare against eight baselines:

Heuristic Baselines:

- **Random**: Random ordering (expected APFD ≈ 0.5)
- **Recency**: Prioritize recently failed tests
- **RecentFailureRate**: Failure rate in last 5 builds
- **FailureRate**: Historical failure rate
- **GreedyHistorical**: Combined heuristics

ML Baselines:

- **Logistic Regression**
- **Random Forest**
- **XGBoost**

TABLE 8: Comparison with State-of-the-Art TCP Methods (Industrial Dataset)

Method	APFD	Std	p-value	Effect	Δ
FILO-PRIORI	**0.7595**	**0.189**	–	–	–
DeepOrder [14]	0.6919	0.267	<0.001***	small	+9.8%
NodeRank [16]	0.6623	0.270	<0.001***	small	+14.7%
Random	0.5000	–	<0.001***	large	+51.9%

*** $p < 0.001$ (Wilcoxon signed-rank test vs FILO-PRIORI)
Effect sizes measured using Cliff's delta.

5.4 Evaluation Metrics

- **APFD**: Primary metric, computed per build
- **Statistical tests**: Wilcoxon signed-rank test ($\alpha = 0.05$)
- **Confidence intervals**: 95% bootstrap CI (1000 iterations)

5.5 Implementation Details

- **Framework**: PyTorch 2.0, PyTorch Geometric 2.3
- **Hardware**: NVIDIA RTX 3090 (24GB VRAM)
- **Training**: 50 epochs, batch size 32, AdamW optimizer
- **Learning rate**: 3×10^{-5} with cosine annealing
- **Early stopping**: Patience 15, monitoring val_f1_macro

6 RESULTS

This section presents experimental results organized by research questions. We evaluate FILO-PRIORI on two complementary domains: (1) industrial software testing with rich semantic information, and (2) GNN benchmark datasets representing node classification tasks on citation networks.

6.1 RQ1: Effectiveness on Industrial Dataset

Table 8 presents the comparison of FILO-PRIORI against state-of-the-art TCP methods on the industrial QTA dataset containing 4,552 builds with 100,502 test executions (277 builds with failures). We use APFD as the primary metric with Wilcoxon signed-rank tests for statistical significance and Cliff's delta for effect size.

Key Findings for RQ1:

- FILO-PRIORI achieves a mean APFD of **0.7595** (std: 0.189), representing a **51.9%** improvement over random ordering.
- FILO-PRIORI **significantly outperforms** NodeRank [16] by **+14.7%** ($p < 0.001$, Cliff's delta = 0.179, small effect).
- FILO-PRIORI also outperforms DeepOrder [14] by **+9.8%** ($p < 0.001$), demonstrating the effectiveness of our dual-stream graph attention architecture.
- Per-build analysis shows FILO-PRIORI outperforms NodeRank in **52.7%** of builds (vs 31.0% for NodeRank, 16.2% ties).

Answer to RQ1

FILO-PRIORI achieves APFD=0.7595, significantly outperforming state-of-the-art methods: NodeRank ($+14.7\%$, $p < 0.001$) and DeepOrder ($+9.8\%$, $p < 0.001$). The dual-stream GAT architecture with semantic and structural features provides superior fault detection compared to existing GNN-based approaches.

6.2 Evaluation on GNN Benchmark Datasets

To validate the generalizability of our approach beyond industrial software testing, we conducted a comprehensive evaluation on standard GNN benchmark datasets used in Li et al.'s NodeRank [16]. This experiment demonstrates that the core principles of FILO-PRIORI—multi-edge graph construction, comprehensive feature extraction, and learning-to-rank—transfer effectively to the domain of test input prioritization for Graph Neural Networks.

6.2.1 Experimental Setup

We evaluated on three citation network datasets commonly used in GNN research: **Cora** (2,708 nodes, 7 classes), **CiteSeer** (3,327 nodes, 6 classes), and **PubMed** (19,717 nodes, 3 classes). Following the NodeRank experimental protocol [16], we trained a 2-layer Graph Convolutional Network (GCN) for node classification and treated misclassified nodes as "faults" to be detected through prioritization.

We adapted the FILO-PRIORI methodology for GNN testing by implementing:

- **Multi-Edge Graph Construction**: Combining original citation edges, semantic similarity edges (cosine similarity > 0.65), and prediction agreement edges (nodes with same predicted class).
- **Feature Extraction**: 25 features across three categories—structural (degree centrality, clustering coefficient, PageRank), uncertainty (entropy, margin, Gini impurity, confidence), and neighborhood (neighbor entropy, agreement, label homophily).
- **Learning-to-Rank**: LightGBM classifier trained to predict misclassification likelihood, with predicted probability as the ranking score.

We compared against five baselines: Random ordering, DeepGini [19] (Gini impurity), Entropy (prediction entropy), VanillaSM (softmax margin), and PCS (least confidence). All experiments were repeated 5 times with different random seeds, and we report mean and standard deviation.

6.2.2 Results

Table 9 presents detailed results for each dataset, and Table 10 summarizes the comparison with NodeRank.

Key Findings:

(1) Superior Performance on Cora. FILO-PRIORI achieves APFD=0.861, outperforming NodeRank (0.833) by **+3.4%**. This represents a substantial improvement on the most commonly used GNN benchmark. The strong PFD@10=0.412 indicates that FILO-PRIORI detects over 41% of misclassifications in the first 10% of the prioritized test sequence.

TABLE 9: Comparison with NodeRank on GNN Benchmark Datasets

Dataset	Method	APFD	Std	PFD@10	PFD@20
Cora	Random	0.502	0.012	0.101	0.202
	DeepGini	0.714	0.011	0.258	0.467
	Entropy	0.723	0.008	0.260	0.466
	VanillaSM	0.714	0.012	0.256	0.465
	NodeRank†	0.833	–	–	–
	Filo-Priori	**0.861**	**0.014**	**0.412**	**0.755**
CiteSeer	Random	0.500	0.004	0.099	0.202
	DeepGini	0.611	0.012	0.168	0.316
	Entropy	0.620	0.012	0.167	0.316
	VanillaSM	0.610	0.013	0.169	0.315
	NodeRank†	0.733	–	–	–
	Filo-Priori	**0.742**	**0.016**	**0.228**	**0.436**
PubMed	Random	0.498	0.010	0.100	0.199
	DeepGini	0.646	0.011	0.216	0.382
	Entropy	0.649	0.012	0.216	0.382
	VanillaSM	0.645	0.011	0.216	0.383
	NodeRank†	0.790	–	–	–
	Filo-Priori	**0.785**	**0.014**	**0.322**	**0.570**

Results averaged over 5 independent runs with different random seeds.
†NodeRank results from Li et al. [16] (IEEE TSE 2024).
PFD@k: Percentage of faults detected in first k% of tests.

TABLE 10: Summary: Filo-Priori vs. NodeRank on GNN Benchmarks (APFD)

Dataset	Filo-Priori	NodeRank†	Δ	Improvement
Cora	**0.861**	0.833	+0.028	**+3.4%**
CiteSeer	**0.742**	0.733	+0.009	**+1.3%**
PubMed	0.785	**0.790**	-0.005	-0.6%
Average	**0.796**	0.785	+0.011	**+1.4%**

†NodeRank results from Li et al. [16] (IEEE TSE 2024).
Bold indicates better performance. Filo-Priori wins 2/3 datasets.

(2) Consistent Improvement on CiteSeer. On CiteSeer, FILO-PRIORI achieves APFD=0.742 compared to NodeRank's 0.733, a **+1.3%** improvement. CiteSeer is known to be more challenging due to higher feature sparsity and less distinct class boundaries, yet FILO-PRIORI maintains competitive performance.

(3) Competitive Performance on PubMed. On PubMed, FILO-PRIORI achieves APFD=0.785, slightly below NodeRank (0.790, -0.6%). PubMed's larger scale (19,717 nodes) and lower class count (3 classes) present a different challenge where uncertainty-based features may provide less discriminative power. Nevertheless, the difference is within one standard deviation, indicating comparable performance.

(4) Consistent Gains over Uncertainty Baselines. Across all datasets, FILO-PRIORI substantially outperforms uncertainty-based baselines (DeepGini, Entropy, VanillaSM), with improvements ranging from +11% to +21%. This validates the value of incorporating structural information and learning-to-rank beyond simple uncertainty measures.

(5) Strong Early Fault Detection. The PFD@10 and PFD@20 metrics demonstrate FILO-PRIORI's effectiveness in early fault detection. On Cora, PFD@20=0.755 means 75.5% of all misclassifications are found in the first 20% of the test sequence—a 3× improvement over random ordering.

TABLE 11: RTPTorrent Cross-Dataset Results (20 Projects)

Baseline	APFD	Model Improv.	p-value
Filo-Priori V10	**0.8376**	–	–
recently_failed	0.8209	+2.02%	sig.*
random	0.4940	+69.56%	<0.001***
untreated	0.3574	+134.32%	<0.001***
matrix_naive	0.5693	+47.11%	<0.001***
matrix_conditional	0.5132	+63.21%	<0.001***
optimal_duration	0.5934	+41.15%	<0.001***
optimal_failure (oracle)	0.9249	-9.45%	–

Aggregate results over 1,250 test builds from 20 Java projects.
* Statistically significant improvement over strongest baseline.

TABLE 12: Top-5 Projects by APFD (RTPTorrent)

Project	APFD	Builds	vs recent.
apache/sling	0.9922	163	+2.18%
neuland/jade4j	0.9799	20	+0.88%
eclipse/jetty.project	0.9789	66	+1.27%
facebook/buck	0.9722	69	+1.97%
deeplearning4j/dl4j	0.9277	114	+0.46%

GNN Benchmark Finding

FILO-PRIORI achieves an average APFD of 0.796 across GNN benchmarks, outperforming NodeRank [16] on 2 of 3 datasets (Cora: +3.4%, CiteSeer: +1.3%) and achieving competitive results on PubMed (-0.6%). The multi-edge graph construction and comprehensive feature extraction transfer effectively from software testing to GNN test input prioritization.

6.3 Cross-Dataset Validation: RTPTorrent

To further evaluate generalization, we conducted a comprehensive experiment on the RTPTorrent open-source benchmark [18], which contains test execution histories from 20 Java projects with over 100,000 Travis CI builds. We adapted FILO-PRIORI using a LightGBM LambdaRank model with 16 ranking-optimized features, trained independently on each project.

Table 12 shows the top-5 performing projects:

Key Findings for Cross-Dataset Validation:

- FILO-PRIORI achieves a mean APFD of **0.8376** across 20 diverse Java projects (1,250 test builds), demonstrating strong cross-domain generalization.
- The model **outperforms 6 out of 7 baselines**, including the strongest heuristic baseline (recently_failed) by +2.02%.
- Only the oracle baseline (optimal_failure) achieves higher APFD (0.9249), representing the theoretical upper bound with perfect failure knowledge.
- Performance varies across projects (APFD: 0.29–0.99), with best results on projects with consistent failure patterns (e.g., apache/sling).
- The most important features are `novelty_score`, `base_risk`, `execution_frequency`, and `time_decay_score`.

TABLE 13: Ablation Study: Component Contributions to APFD

Configuration	APFD	Δ	p-value
Full Model	0.7595	–	–
w/o Dense Multi-Edge Graph	0.6835	-10.0%	<0.001***
w/o Orphan KNN Scoring	0.7145	-5.9%	<0.001***
w/o Single Balancing	0.7291	-4.0%	<0.001***
w/o DeepOrder Features	0.7443	-2.0%	0.003**
w/o Threshold Optimization	0.7519	-1.0%	0.042*
Baseline (V1)	0.6503	-14.4%	<0.001***

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$ (Wilcoxon signed-rank test)

Cross-Dataset Finding

FILO-PRIORI generalizes effectively to open-source projects, achieving APFD=0.8376 across 20 RTPTorrent projects and outperforming 6 of 7 baselines. Combined with GNN benchmark results (0.796 avg APFD), this demonstrates the approach's versatility across software testing and machine learning domains.

6.4 RQ2: Ablation Study

To understand the contribution of each component, we conducted an ablation study isolating five key innovations. Table 13 shows the impact of removing each component from the full system (APFD=0.7595).

Component Analysis:

(1) Dense Multi-Edge Graph (+10.0%): The largest contribution comes from increasing graph connectivity. Key changes: semantic threshold 0.75→0.65, top-k neighbors 5→10, and adding temporal/component edges. This increased in-graph coverage from 50–60% to **77.4%**.

(2) Orphan KNN Scoring (+5.9%): Orphan tests (22.6% of total) previously received uniform scores, destroying ranking. Our KNN pipeline with Euclidean distance, structural blending ($w = 0.35$), and temperature scaling ($T = 0.7$) restores meaningful score variance (std=0.046 vs. 0.000).

(3) Single Balancing (+4.0%): Prior work used triple compensation (class weights + balanced sampling + focal alpha), causing mode collapse. Using *only* balanced sampling (15:1 ratio) with neutral $\alpha = 0.5$ stabilizes training and prevents over-prediction of the minority class.

(4) DeepOrder Features (+2.0%): Temporal execution features (execution_status_last_[1,2,3,5,10], cycles_since_last_fail) capture recent failure patterns that static features miss.

(5) Threshold Optimization (+1.0%): Two-phase search finding threshold 0.28 (vs. default 0.50) improves minority class recall from 5% to 30%.

TABLE 14: Temporal Cross-Validation Results

Validation Method	APFD	Folds
Temporal 5-Fold CV	0.7821	5
Sliding Window CV	0.7412	10
Concept Drift Test	0.7298	3
Average	**0.7510**	–

TABLE 15: Hyperparameter Sensitivity Analysis

Parameter	Values	Best	Impact
Loss Function	CE, Focal, W. Focal	W. Focal	4.7%
Focal Gamma	1.5, 2.0, 2.5, 3.0	2.5	4.5%
Learning Rate	1e-5, 3e-5, 5e-5, 1e-4	3e-5	3.6%
GNN Layers	1, 2, 3	1	3.6%
GNN Heads	1, 2, 4, 8	2	2.4%

Answer to RQ2

The five key innovations contribute cumulatively to the +16.8% improvement over baseline: Dense Multi-Edge Graph (+10.0%), Orphan KNN Scoring (+5.9%), Single Balancing (+4.0%), DeepOrder Features (+2.0%), and Threshold Optimization (+1.0%).

6.5 RQ3: Temporal Validation

To evaluate generalization to future builds, we used temporal cross-validation where training data always precedes test data chronologically.

Key Findings for RQ3:

- Performance remains stable across temporal validation methods, with APFD ranging from 0.73 to 0.78.
- Concept drift causes only 3% degradation, indicating robustness to evolving test patterns over time.

Answer to RQ3

FILO-PRIORI demonstrates robust temporal generalization (APFD: 0.73–0.78), with minimal concept drift degradation (3%).

6.6 RQ4: Hyperparameter Sensitivity

We analyzed sensitivity to key hyperparameters across multiple configurations.

Key Findings for RQ4:

- **Loss Function**: Weighted Focal Loss has the largest impact (4.7%).
- **Architecture**: Simpler configuration (1 layer, 2 heads) performs best, avoiding overfitting on the test graph.

Answer to RQ4

Optimal configuration: Weighted Focal Loss ($\gamma = 2.5$), learning rate 3e-5, 1-layer GAT with 2 heads. Loss function choice has the largest impact (4.7%).

TABLE 16: Summary of Experimental Results

Experiment	APFD	vs. Best Baseline	Significance
Industrial QTA	0.760	+9.8% (DeepOrder)	$p < 0.001$
GNN Benchmarks	0.796	+1.4% (NodeRank)	2/3 wins
RTPTorrent (20 proj.)	0.838	+2.0% (recently_failed)	sig.*
Overall Average	**0.798**	–	–

APFD: Average Percentage of Faults Detected (higher is better).
Results demonstrate consistent performance across three distinct domains.

6.7 Summary of Results

Table 16 summarizes FILO-PRIORI's performance across all experimental settings.

The experimental results demonstrate that FILO-PRIORI achieves consistent, state-of-the-art performance across three distinct evaluation settings: industrial software testing (0.760), GNN model testing (0.796), and open-source CI/CD environments (0.838). The multi-edge graph construction and dual-stream architecture provide robust prioritization regardless of the specific domain or characteristics of the test data.

7 DISCUSSION

This section discusses the implications of our findings, analyzes the reasons behind FILO-PRIORI's effectiveness, and addresses practical considerations.

7.1 Why Does Multi-Edge Graph Construction Matter?

The ablation study reveals that the Dense Multi-Edge Graph contributes +10.0% to performance, making it the most critical component. We attribute this to several factors:

Capturing Test Dependencies: The multi-edge graph encodes relationships that simple features cannot capture. Tests that co-fail often share underlying dependencies on the same code modules, and GAT learns to propagate failure signals through these connections.

Dynamic Attention: Unlike standard GAT, GATv2 [13] computes dynamic attention that depends on both query and key nodes. This allows the model to selectively attend to the most relevant neighbors for each test case, adapting to different failure patterns.

Multi-Edge Information: Our multi-edge graph combines co-failure, co-success, and semantic edges. This provides a richer signal than single-edge approaches, increasing graph density from 0.02% to 0.5-1.0%.

7.2 The Role of Weighted Focal Loss

A key design choice of FILO-PRIORI is addressing the severe class imbalance (37:1 Pass:Fail ratio). Traditional TCP approaches using standard cross-entropy are dominated by the majority class. Our Weighted Focal Loss addresses this:

$$\mathcal{L} = -\alpha \cdot w_t \cdot (1 - p_t)^{\gamma} \cdot \log(p_t) \tag{17}$$

The sensitivity analysis shows that the choice of loss function has a 4.7% impact on performance, with Weighted

Focal Loss achieving the best results. Additionally, the ablation study reveals that proper balancing strategy (Single Balancing) contributes +4.0%. This validates our hypothesis that careful handling of class imbalance is critical for TCP.

7.3 Comparison with FailureRate Baseline

FILO-PRIORI outperforms the FailureRate heuristic by 1.4%, though not statistically significant ($p = 0.363$). This raises an important question: *When is a deep learning approach preferable to simple heuristics?*

We observe that FILO-PRIORI provides advantages in:

- **New test cases**: Tests without history benefit from semantic similarity to known failing tests.
- **Changing patterns**: The model adapts to evolving failure patterns through the graph structure.
- **Complex dependencies**: The GNN captures multi-hop relationships that simple heuristics miss.

However, the marginal improvement suggests that for datasets with stable failure patterns, simpler approaches may be sufficient.

7.4 Practical Implications

For Practitioners: FILO-PRIORI can be integrated into CI/CD pipelines to prioritize test execution. The 51.9% improvement over random ordering translates to substantially faster fault detection, reducing the feedback loop for developers.

Computational Cost: Training requires approximately 2-3 hours on a single GPU. Inference is fast (<1 second per build), making real-time prioritization feasible.

Data Requirements: The approach requires historical test execution data with at least 50 builds for effective graph construction. Projects with limited history may benefit from transfer learning approaches.

7.5 Lessons Learned

1) **Graph structure matters**: Modeling test relationships through graphs provides substantial benefits over treating tests independently.
2) **Simple architectures suffice**: 1-layer GNN with 2 heads outperformed deeper architectures, suggesting that test relationships can be captured with shallow networks.
3) **Feature selection is important**: 10 carefully selected features outperformed 29 features, indicating that noise reduction improves generalization.
4) **Address class imbalance**: Weighted Focal Loss is essential for handling the extreme class imbalance in test execution data.

7.6 Limitations

While FILO-PRIORI demonstrates strong performance, several limitations exist:

- **Domain specificity**: While we evaluated on three distinct domains (industrial testing, GNN benchmarks, and open-source CI/CD), results may not generalize to all software projects or testing contexts.
- **Cold start**: New test cases without semantic similarity to existing tests may not benefit from the graph structure.
- **Graph construction overhead**: Building the test relationship graph adds preprocessing time, though this is amortized over multiple predictions.
- **Interpretability**: While ablation studies provide component-level insights, individual predictions remain difficult to explain.

8 THREATS TO VALIDITY

We discuss threats to the validity of our study following established guidelines for empirical software engineering research [20].

8.1 Internal Validity

Internal validity concerns factors that may affect the causal relationship between our approach and the observed results.

Implementation Correctness: We mitigated implementation errors through unit testing, code review, and comparison with baseline implementations. Our replication package allows independent verification.

Randomness: Deep learning involves stochastic elements (weight initialization, dropout, batch sampling). We used fixed random seeds (42) and report results averaged over multiple runs with confidence intervals.

Hyperparameter Selection: Hyperparameters were selected through grid search on validation data, not test data. We report sensitivity analysis (RQ4) to show the impact of different choices.

Data Leakage: We ensured strict temporal separation between training and test data. The model never sees future build information during training, and we validate with temporal cross-validation (RQ3).

8.2 External Validity

External validity concerns the generalizability of our findings.

Dataset Diversity: While we evaluated on three distinct settings—an industrial dataset (52,102 executions), GNN benchmarks (Cora, CiteSeer, PubMed), and 20 open-source RTPTorrent projects—results may not generalize to all software projects with different testing practices or technology stacks.

Domain Specificity: The QTA dataset comes from a specific application domain. Projects with different testing practices, failure rates, or code structures may exhibit different results.

Scale: Our dataset contains 277 builds with failures. Very large projects (e.g., Google-scale [3]) may present different challenges that require additional optimizations.

Programming Languages: The dataset contains tests from a specific technology stack. Semantic embeddings may perform differently for other programming languages or testing frameworks.

8.3 Construct Validity

Construct validity concerns whether our measurements accurately reflect the concepts we intend to measure.

APFD Metric: We use APFD as the primary metric, which is standard in TCP research [6]. However, APFD assumes equal fault severity and detection cost. Alternative metrics (NAPFD, cost-cognizant APFD) may provide complementary insights.

Statistical Tests: We use Wilcoxon signed-rank tests with $\alpha = 0.05$ and report 95% bootstrap confidence intervals. These are appropriate for non-parametric comparisons of paired samples.

Baseline Selection: We compare against eight baselines spanning heuristic and ML approaches. While comprehensive, some recent approaches (e.g., specific RL variants) were not included due to implementation complexity or lack of public code.

8.4 Conclusion Validity

Conclusion validity concerns the relationship between treatment and outcome.

Statistical Power: With 277 builds containing failures, we have sufficient statistical power to detect meaningful differences. Small effect sizes (e.g., 1.4% improvement over FailureRate) may not be statistically significant but can be practically meaningful.

Multiple Comparisons: We compare against multiple baselines without correction for multiple testing. This increases the risk of Type I errors, though our primary comparison (vs. Random) shows highly significant results ($p < 0.001$).

8.5 Reproducibility

To ensure reproducibility, we provide:

- Complete source code for FILO-PRIORI and all baselines
- Configuration files with exact hyperparameters
- Trained model weights
- Anonymized dataset with documentation
- Scripts to reproduce all experiments

All materials are available in our replication package at: https://github.com/[anonymized]/filo-priori-v9

9 Conclusion

We presented FILO-PRIORI, a dual-stream deep learning approach for Test Case Prioritization that combines Graph Attention Networks with semantic embeddings. By modeling test relationships through multi-edge graphs, we capture complex dependencies between test cases that simpler approaches ignore.

Our key contributions include:

- A dual-stream GAT architecture combining SBERT semantic embeddings with structural feature learning from test execution history.
- Multi-edge test relationship graphs capturing co-failure, co-success, and semantic similarity relationships.
- Empirical validation on industrial data achieving **APFD=0.7595**, significantly outperforming NodeRank ($+14.7\%$, $p < 0.001$) and DeepOrder ($+9.8\%$).
- State-of-the-art results on GNN benchmarks (**APFD=0.796**), outperforming NodeRank on Cora ($+3.4\%$) and CiteSeer ($+1.3\%$).
- Cross-dataset validation on RTPTorrent achieving **APFD=0.838** across 20 Java projects, outperforming 6 of 7 baselines.
- Robust temporal generalization (APFD: 0.73–0.78 across time periods).

The ablation study reveals that the Dense Multi-Edge Graph construction is the most critical component ($+10.0\%$), validating our hypothesis that modeling test relationships through rich graph structures is more effective than treating tests as independent entities.

Future Work. We plan to: (1) investigate dynamic graph construction, (2) explore cross-project transfer learning with pre-trained code embeddings, and (3) develop real-time prioritization for CI/CD pipelines.

Data Availability. Our replication package, including source code, trained models, configuration files, and anonymized dataset, is available at: https://github.com/[anonymized]/filo-priori-v9

Acknowledgments

This work was supported by [funding information]. We thank [acknowledgments].

References

[1] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 2016, pp. 426–437.

[2] M. Fowler and M. Foemmel, "Continuous integration," *ThoughtWorks*, 2006. [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html

[3] A. Memon, Z. Gao, B. Nguyen, S. Dhanda, E. Nickell, R. Siemborski, and J. Micco, "Taming google-scale continuous testing," in *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, 2017, pp. 233–242.

[4] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[5] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[6] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: An empirical study," in *Proceedings IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 1999, pp. 179–188.

[7] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the 24th International Conference on Software Engineering (ICSE)*. ACM, 2002, pp. 119–129.

[8] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2017, pp. 12–22.

[9] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case prioritization and selection based on deep learning," *Empirical Software Engineering*, vol. 27, no. 6, pp. 1–42, 2022.

[10] D. M. German, A. E. Hassan, and G. Robles, "Change impact graphs: Determining the impact of prior code changes," *Information and Software Technology*, vol. 51, no. 10, pp. 1394–1408, 2009.
[11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations (ICLR)*, 2018.
[12] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988.
[13] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" in *International Conference on Learning Representations (ICLR)*, 2022.
[14] J. Chen, Y. Bai, D. Hao, L. Zhang, L. Zhang, and B. Xie, "Deeporder: Deep learning for test case prioritization in continuous integration testing," in *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1–12.
[15] A. Abdelkarim, K. K. Sabor, G. Bonnet, and F. Ber, "Tcp-net: Test case prioritization using end-to-end deep neural networks," in *Proceedings of the IEEE/ACM International Conference on Automation of Software Test (AST)*. IEEE, 2022, pp. 1–12.
[16] Y. Li, X. Dang, W. Pian, A. Habib, J. Klein, and T. F. Bissyandé, "Test input prioritization for graph neural networks," *IEEE Transactions on Software Engineering*, vol. 50, no. 6, pp. 1396–1424, 2024.
[17] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL, 2019, pp. 3982–3992.
[18] T. Mattis, T. Rausch, and M. Rinard, "RTPTorrent: An open-source dataset for evaluating regression test prioritization," in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*. ACM, 2020, pp. 558–562.
[19] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "Deepgini: Prioritizing massive tests to enhance the robustness of deep neural networks," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2020, pp. 177–188.
[20] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*. ACM, 2011, pp. 1–10.

Acauan C. Ribeiro is a researcher at the Institute of Computing (IComp), Federal University of Amazonas (UFAM), Brazil. His research interests include software testing, machine learning for software engineering, and continuous integration.