mathematics

MDPI

Article

SeqRankFL: Sequence-Aware Ranking of LLM-Based Code Representations for Statement-Level Fault Localization

Dong An [1,2], Shihai Wang [1,2,*], Bin Liu [1,2] and Liandie Zhu [1,2,3]

[1] School of Reliability and Systems Engineering, Beihang University, 37 Xueyuan Road, Haidian District, Beijing 100191, China; andong@buaa.edu.cn (D.A.); liubin@buaa.edu.cn (B.L.); zy1914232@buaa.edu.cn (L.Z.)
[2] National Key Laboratory of Reliability and Environmental Engineering, 37 Xueyuan Road, Haidian District, Beijing 100191, China
[3] Beijing Electromechanical Research Institute, Beijing 100074, China
* Correspondence: wangshihai@buaa.edu.cn

Abstract

Software reliability is an important part of reliability assurance for complex engineering systems, and timely fault diagnosis supports safe and continuous operation. After a test failure, statement-level fault localization ranks source-code lines for early inspection. Representation-based approaches can operate without a coverage matrix by extracting line-level hidden states from a frozen large language model for code (code LLM), but their downstream readout and training objective are not necessarily aligned with source-window ranking and buggy-version-level evaluation. We propose SeqRankFL, a sequence-aware ranker for LLM-based code representations. It combines a bidirectional long short-term memory network (BiLSTM), which follows code order within a source window, with a Hybrid objective that joins binary cross-entropy (BCE) and the listwise learning-to-rank loss ListNet. On the source-window ranking task using windows of at most 128 physical lines within known buggy files from BugsInPy and Defects4J, relative to a matched LLMAO-style Transformer+BCE reference that shares the same representations, splits, and evaluation protocol, SeqRankFL raises the equally weighted Top-1 from 51.31% to 58.58%, an absolute gain of 7.27 percentage points, with consistent improvements on both datasets. Further controlled analyses show that multi-depth layer mixing and syntax scope mainly improve average ranks, whereas control-flow/data-flow graphs and failure behavior have condition-dependent effects across models, languages, and project partitions. The controlled results identify the listwise objective as the primary driver of the Top-1 improvement, with the BiLSTM readout providing an additional architecture-dependent gain under the ranking-aware objective.

Keywords: software reliability; large language models; software fault localization; learning to rank; deep learning

MSC: 68T07; 68N30

Academic Editor: Michael Voskoglou

Received: 6 August 2026
Revised: 31 August 2026
Accepted: 5 September 2026
Published: 11 September 2026

1. Introduction

Software is a critical component of complex engineering systems, and software reliability directly affects dependable system operation. As a diagnostic activity within reliability assurance, software fault localization ranks program locations that may explain an observed failure, helping developers inspect the most relevant code before repair and

regression testing. Classical spectrum-based fault localization derives statement suspiciousness from coverage differences between passing and failing tests; Tarantula, similarity coefficients, and DStar are representative formulations [1–4]. These methods require executable tests, instrumentation, and a reliable coverage matrix. Such a dynamic-analysis front end is unavailable or too costly in settings with incomplete test infrastructure or source-only access.

Learning-based and graph-based methods broaden the available evidence by combining coverage, code changes, semantic features, and control-flow or data-flow structure in a learned ranking function [5–10]. They demonstrate the value of program semantics and structure but generally still depend on coverage, test outcomes, or additional analysis tools. This motivates a complementary setting in which a frozen code model supplies line-level signals and a lightweight downstream model performs local ranking without receiving a coverage matrix.

LLMAO established the feasibility of this route by freezing a left-to-right code LLM, extracting line-level hidden states offline, and training a lightweight bidirectional adapter with line-wise binary cross-entropy [11]. Other LLM-based fault-localization studies have explored broader input contexts and direct model inference [12–14]. Together, these works provide the representation-based foundation for the present study and leave open a downstream design question: how should frozen line representations be read and trained when the output is a source-window ranking whose metrics are aggregated at the buggy-version level rather than a set of independent line labels?

Three downstream questions follow from this setting. First, do the localizer and loss reflect a ranking problem? BCE treats lines independently, whereas the localizer ranks candidates within each source window, and the resulting metrics are aggregated by buggy version; positive lines account for only 3.10% of valid lines in BugsInPy and 2.47% in Defects4J. Second, is the final layer the only useful readout? Fault-relevant cues may be distributed across model depth and may interact with local syntax and scope. Third, do additional inputs transfer across conditions? Control-flow graphs (CFGs), data-flow graphs (DFGs), and failure-behavior summaries are plausible evidence, but they may be redundant with the frozen representation or behave differently across models and languages.

We investigate these questions in a source-window task with a known buggy file and no coverage matrix at the localizer input. We propose SeqRankFL, which replaces the reference readout with a BiLSTM and adds a within-window ranking constraint through the BCE+ListNet Hybrid objective. We then audit internal representation information, including multi-depth layer mixing, depth residuals, neighborhood residuals, and syntax scope, separately from external CFG/DFG and failure-behavior inputs. Each experiment family changes only its prespecified factor.

The main contributions are as follows.

1. **A sequence-aware downstream ranker.** We match a BiLSTM readout with a BCE+ListNet Hybrid objective for sparse source-window ranking of frozen line representations. The design combines a code-order inductive bias with an explicit listwise constraint.
2. **An audit of representation and input information.** Under shared representations, splits, seeds, and version-level evaluators, we separate internal factors such as layer depth, neighborhood residuals, and syntax scope from external CFG/DFG and failure-behavior inputs. The results identify which information improves average ranking and where its direction depends on the evaluation condition.
3. **Cross-language validation.** We evaluate the same design on real Python and Java faults from BugsInPy and Defects4J (experiments run under Python 3.12), report

the datasets separately, and add leave-one-project-out evidence for within-window generalization to unseen projects.

The remainder of this paper introduces related work, the problem definition, and the method, followed by the experimental protocol, results for four research questions, a mechanistic discussion, the applicability boundary, and threats to validity.

2. Background and Related Work

2.1. Coverage-Driven and Learning-Based Fault Localization

Spectrum-based fault localization computes suspiciousness from differences in statement execution between passing and failing tests. Its rules are simple and interpretable, but its quality depends on coverage completeness and failure discrimination. Mutation-based methods such as Metallaxis add mutation effects as another signal, at the cost of substantially more test executions [15,16]. Learning-based methods combine complexity, coverage, change, and semantic features, while graph-based methods propagate evidence over control-flow, data-flow, or program-dependence relationships [5–10].

These studies share a central property: fault localization produces relative priorities among candidate program elements rather than independent class labels. We study a complementary setting in which the localizer receives no statement coverage matrix and uses frozen code representations, source-computable structure, and historical defect supervision.

2.2. Frozen Code Models and LLM-Based Fault Localization

LLM-based fault localization follows two routes: direct inference and frozen-representation ranking. Direct-inference methods provide code, failure information, and task instructions to a model and request a file, method, or line prediction. AutoFL, AgentFL, and project-context-oriented studies enlarge the visible context through tool calls or staged workflows [13,17,18]. They support interactive and potentially explanatory localization, but their outputs depend on context budgets, prompts, and decoding behavior.

The representation-based route instead treats a code model as a frozen feature extractor. CodeBERT, GraphCodeBERT, and large code-generation models demonstrate that lexical, semantic, and data-flow information can be encoded in hidden vectors [19–22]. LLMAO uses such a frozen model to obtain line-level representations and trains a lightweight bidirectional adapter with line-wise BCE for test-independent localization [11].

Relation to this paper. SeqRankFL uses the same representation-based task boundary and studies three downstream choices that have not been separated in a single controlled protocol: sequence readout, ranking objective, and the conditional value of additional information. The matched reference is a Transformer+BCE configuration trained with the same representations, splits, seeds, and evaluator.

2.3. Hierarchical Representations, Program Structure, and Learning to Rank

Transformer self-attention can represent long-range dependencies within a sequence [23], but different layers tend to encode information at different granularities. Prior work on hierarchical representations shows that intermediate layers may preserve lexical, syntactic, and semantic cues that are useful for downstream tasks, whereas the final layer is optimized primarily for the pretraining or generation objective [24–27]. For code, program-dependence and program graphs provide explicit control-flow, data-flow, and scope relationships [28,29].

Learning-to-rank methods address relative priority at the objective level. RankNet expresses pairwise order constraints, whereas ListNet defines a cross-entropy objective over listwise score distributions [30,31]. We treat multi-depth states, neighborhood residuals, and syntax scope as separable representation factors, and evaluate ListNet alongside BCE

under shared data and evaluators. Which factors enter the main method is decided by those controlled results.

2.4. Conditionality of Additional Input Information

Structured inputs append explicit program relationships, while failure-behavior summaries are closer to observed runtime phenomena. Yet, additional context is not automatically useful: input organization and positional effects can determine whether a model uses the information as intended [32,33]. We therefore evaluate external inputs as conditional factors across the tested models and both language datasets.

3. Problem Definition and Task Boundary

3.1. Task Formulation

A source window for a buggy version contains T physical source-code lines, denoted by $X = (x_1, \ldots, x_T)$, and the set of true faulty lines is $Y \subseteq \{1, \ldots, T\}$. A frozen code model f_θ produces hidden states for each token, and a downstream localizer g_ϕ outputs a logit a_t for each valid line. Sorting by a_t in descending order gives the ranking $\pi(X)$. If any line in Y occurs within the top k positions, the window is counted as a Top-k hit. A buggy version may correspond to multiple source windows. The reported metrics are aggregated at the version level according to the rule in Section 5.3, so the evaluation unit is always a buggy version rather than a window.

3.2. Information Boundary and Terminology

The localizer input contains no statement coverage matrix. The code-only condition uses frozen code representations and internal features computed from the source file; external conditions append static CFG/DFG or failure-behavior information.

Source windows are generated by our preprocessing from known buggy files. Recorded faulty lines are greedily grouped into clusters whose span relative to the first faulty line is at most 127 lines. Each cluster is then centered within a budget of at most 128 physical source lines, with the interval clamped to the file boundaries. Files shorter than this budget remain shorter at the source level; tensor padding is used only for batching and does not introduce additional source-code candidates. Fault locations therefore participate in window construction and supervision, but they are not provided to the downstream localizer as scoring features. When a buggy version contains multiple fault clusters, preprocessing can produce multiple fragments of the same source file; same-file interval overlap occurs in 19/501 BugsInPy version groups and 13/385 Defects4J version groups.

Leave-one-project-out (LOPO) holds out an entire project to test within-window generalization to unseen projects.

For clarity, Table 1 summarizes the terminology and task-boundary conventions used throughout this paper.

Table 1. Terminology and task-boundary conventions.

Term	Meaning
Source window	A candidate source-code fragment within a known buggy file. The maximum window length in this experiment is given in Table 2.
Localizer	A lightweight downstream sequence model that reads frozen line representations and outputs line-wise suspiciousness, distinguished from the frozen code model itself.
Reference	The matched Transformer+BCE downstream condition, using the representations, splits, seeds, and evaluator of this paper.
Code-Only	An input condition that uses frozen code representations and internal features without CFG, DFG, failure behavior, or other external input.

Table 1. *Cont.*

Term	Meaning
External Input	An input condition that appends static structural or failure-behavior information to Code-Only.
Representation View	A named construction of line representations from frozen states, such as Final-Layer or Layer-Mix+Scope.
Hybrid Objective	The equally weighted sum of line-wise BCE and listwise ListNet.
Equally weighted two-dataset average	The arithmetic average obtained by first averaging within BugsInPy and Defects4J and then giving the two datasets equal weight, preventing the dataset with more versions from dominating the aggregate.

Table 2. Dataset composition and evaluation units. Source windows are the input units of the localizer, buggy versions are the metric aggregation units, and positive ratios are computed over valid lines.

Dataset	Language	Source Windows	Buggy Versions	Maximum Window Lines	Positive Ratio
BugsInPy	Python	598	501	128	3.10%
Defects4J	Java	441	385	128	2.47%

4. Method

4.1. Workflow Overview

SeqRankFL receives line-level hidden states from a frozen code LLM and operates on the source windows defined in Section 3.2. Two properties of the task guide the design. The localizer produces a ranking within each source window, while evaluation aggregates window-level results at the buggy-version level; BCE, in contrast, scores each line independently. With positive lines accounting for only 3.10% of valid lines in BugsInPy and 2.47% in Defects4J, negative examples can dominate the pointwise objective. In addition, the meaning of a faulty line depends on preceding definitions, later uses, and nearby control structures. The downstream model should therefore read the window in code order. SeqRankFL combines a BiLSTM with the BCE+ListNet Hybrid objective: the BiLSTM supplies a sequential inductive bias, and the Hybrid objective constrains both line-level discrimination and relative within-window rank.

As shown in Figure 1, SeqRankFL has four stages. The frozen code model first produces a terminal-aligned representation for every physical source line. Internal representation views are then constructed from the final and intermediate-to-late states, with depth changes, neighborhood residuals, and syntax scope added when selected. The resulting line tokens, together with optional summary or external-input tokens, are processed by a lightweight sequence localizer. Finally, BCE and ListNet are optimized over valid lines within each source window to produce window-level suspiciousness rankings, which are aggregated by buggy version during evaluation as described in Section 5.3.

The stage separation also makes the comparisons reproducible. The code LLM remains frozen, and states at ten depth positions are extracted in one forward pass and cached offline. Changing the localizer, objective, or input condition therefore does not trigger another large-model inference pass. The same cache can be transformed into several representation views and reused across all downstream conditions, so the comparisons do not confound downstream design with feature-extraction randomness.

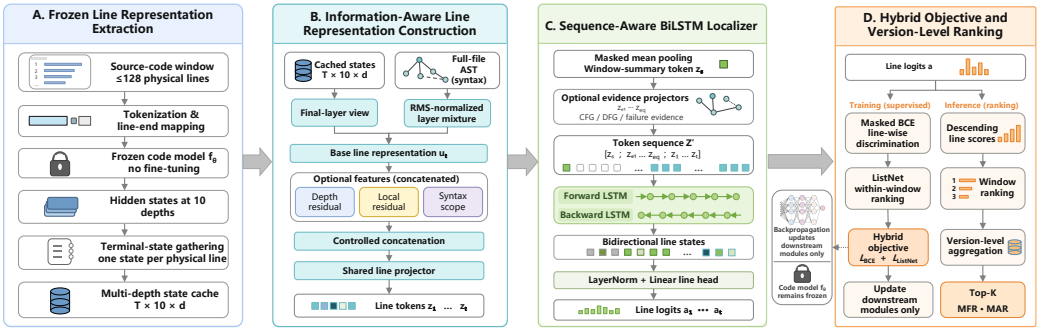

Figure 1. Overview of the SeqRankFL workflow. Dashed boxes denote one-time offline computation, and solid boxes denote reusable lightweight training and inference. External-input tokens are optional and disabled by default.

4.2. Terminal-Aligned Line Representation

For model layer ℓ and source line x_t, let $h_t^{(\ell)} \in \mathbb{R}^d$ denote the hidden state of the source token at the end of that line. If the last physical line does not end with a newline character, the extractor uses the final source token explicitly rather than assuming that a newline state exists. Line masks, labels, and physical line numbers therefore remain aligned, including for the final line. The Final-Layer view uses $h_t^{(L)}$ from the last Transformer block and applies a shared projector to obtain the line token z_t.

Terminal alignment makes every training and evaluation record traceable to one physical source line. Source windows, snapshots, 1-based window starts, line labels, and absolute faulty-line numbers for BugsInPy and Defects4J follow the same rule and can therefore be regenerated from the original source files.

4.3. Internal Representation Information

We construct four separable internal-representation factors and evaluate their prespecified combinations.

Layer-Mix. We apply RMS direction normalization to line states at the 50%, 70%, 90%, and 100% depth positions, average the normalized states with equal weights, and restore the RMS scale of the final layer:

$$h_t^{\text{mix}} = \text{RMS}(h_t^{(L)}) \cdot \frac{1}{4} \sum_{\ell \in \{0.5L, 0.7L, 0.9L, L\}} \frac{h_t^{(\ell)}}{\text{RMS}(h_t^{(\ell)})}. \tag{1}$$

The four layer states come from the same large-model forward pass. The fusion weights are fixed and are not adjusted using labels or validation results.

Depth-Residual. From the ten saved depth positions, we compute adjacent-layer cosine distances and RMS increments. The resulting 18-dimensional features describe how the representation of a line changes through the network.

Neighborhood-Residual. We compute cosine and RMS differences between the current line and its preceding and following lines, together with two-sided residuals and boundary markers. The resulting seven-dimensional features describe representation changes in the local code sequence.

Syntax-Scope. From the abstract syntax tree of the complete source file, we construct 19-dimensional structural features for each line, including method/class scope type, normalized coordinates within the scope, nesting relations, and statement family. These features do not read fault labels, patches, fault types, or test results and do not use relative window position. Syntax-Scope is computed on the experimental windows used in this study.

4.4. Sequence-Aware BiLSTM Localizer

Given projected line tokens $Z = (z_1, \ldots, z_T)$, the localizer may prepend a window-summary token and external-input tokens selected by the input condition:

$$Z' = [z_s; z_{e_1}; \ldots; z_{e_q}; z_1; \ldots; z_T]. \tag{2}$$

The BiLSTM encodes Z' in both directions, allowing line t to read preceding definitions, later uses, and nearby control structures. The two directional outputs are concatenated, normalized with LayerNorm, and passed to a linear prediction head that produces a_t. Prefix tokens participate in encoding but not in the loss or ranking; evaluation uses only the valid-line mask.

The Reference uses a bidirectional Transformer adapter to read the complete window. SeqRankFL retains the same bidirectional visibility and changes the sequence readout to a BiLSTM. Later code in the same window is available during ranking, so the readout is bidirectional. The recurrent structure supplies a stronger prior for adjacent line order and a smaller downstream parameter and search space, which is appropriate for sparse positives and a limited number of candidate lines in each window. The architecture–objective pattern is examined in Section 6.2 using the controlled 3×3 matrix.

A convolutional localizer is retained as a local-structure control: it uses neighboring lines but does not pass a bidirectional state along code order.

4.5. Hybrid Ranking Objective

Line-wise binary cross-entropy is defined as

$$\mathcal{L}_{\mathrm{BCE}} = -\frac{1}{T}\sum_{t=1}^{T}[y_t \log \sigma(a_t) + (1 - y_t)\log(1 - \sigma(a_t))]. \tag{3}$$

To include within-window ranking in the training target, positive lines are uniformly normalized into a target distribution p_t, and the listwise learning-to-rank objective ListNet is applied to the masked-logit softmax distribution q_t [31]:

$$\mathcal{L}_{\mathrm{ListNet}} = -\sum_{t=1}^{T} p_t \log q_t. \tag{4}$$

The Hybrid objective is the equally weighted sum of the two terms:

$$\mathcal{L}_{\mathrm{Hybrid}} = \mathcal{L}_{\mathrm{BCE}} + \mathcal{L}_{\mathrm{ListNet}}. \tag{5}$$

The two terms serve different purposes. BCE operates on each line's absolute probability and preserves faulty/non-faulty discrimination. ListNet operates on the score distribution within a window and requires positive lines to outrank the remaining candidates, thereby constraining relative rank rather than probability calibration. Their complementarity is evaluated by the controlled matrix in Section 6.2.

Section 7.5 reports a relative-coefficient sensitivity analysis using $\lambda\mathcal{L}_{\mathrm{BCE}} + (1 - \lambda)\mathcal{L}_{\mathrm{ListNet}}$. The $\lambda = 0.50$ setting has the same 1:1 relative weighting as the unit-coefficient main objective, differing only by an overall loss scale.

4.6. Class-Imbalance Sensitivity Analysis

The positive-example ratio among valid lines is low (Table 2). To test whether the Hybrid gain could instead be explained by class reweighting, we use the training-fold negative-to-positive ratio $r = n_-/n_+$ to set the BCE positive weight to r^α, with $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$. Class weights use training-fold statistics only. We also compare focal

loss with $\gamma \in \{1, 2, 3\}$ to test whether hard-example focusing can replace the listwise signal [34,35]. The main method remains unweighted Hybrid.

4.7. External Input Information

The Code-Only condition appends no external input. Static conditions append source-computed CFGs, DFGs, or their prespecified combinations; failure conditions append failure-behavior descriptions or a static+failure representation. Training data, labels, seeds, splits, and the evaluator remain fixed, and only the prepended input tokens change. These conditions test transferability in RQ4. SeqRankFL uses Code-Only by default.

4.8. Algorithmic Description

Algorithm 1 summarizes the control flow from frozen states to version-level ranking. Steps 1–2 are one-time offline computations; Steps 3–13 are repeated on the same cache. The frozen model f_θ receives no gradient, so changing a representation view, objective, or input condition does not alter the upstream representation assets.

Algorithm 1 SeqRankFL training and inference procedure

SeqRankFL training and inference.
Input: source-window set {X}; line-level labels {Y}; frozen code model f_θ;
  representation view $V \subseteq$ {Layer-Mix, Depth-Residual,
    Neighborhood-Residual, Syntax-Scope};
  input condition P (Code-Only / Static / Failure); objective L; seed s.
Output: within-window suspiciousness rankings and version-level metrics for each buggy version.
1: for each window X do — *Stage 1: one-time offline extraction*
2: $\{h_t^{(\ell)}\} \leftarrow$ terminal-aligned-extract(f_θ, X) — 10 depth positions
3: for each window X do — *Stage 2: construct representation view*
4: if Layer-Mix $\in V$ then $u_t \leftarrow$ RMS-fuse$(h_t^{(0.5L..L)})$ else $u_t \leftarrow h_t^{(L)}$
5: $c_t \leftarrow$ concat(optional depth, neighborhood, syntax features)
6: $z_t \leftarrow$ project$([u_t; c_t])$ — shared projector to line token
7: for each training fold under the fixed ten-fold protocol and seed s do
8: for epoch $= 1, \ldots, 5$ do
9: for each batch of size 8 do
10: $Z' \leftarrow [z_s; \text{external tokens}(P); z_1, \ldots, z_T]$
11: $a \leftarrow$ prediction-head(LayerNorm(BiLSTM(Z')))
12: update localizer parameters by $L \in$ {BCE,ListNet,Hybrid}; keep f_θ frozen
13: $\pi(X) \leftarrow$ sort$(a,$descending$)$; aggregate by buggy version $\rightarrow$ Top-k/MFR/MAR

5. Experimental Design

5.1. Datasets and Experimental Settings

We use BugsInPy and Defects4J as real-bug benchmarks for Python and Java, respectively [36,37]. Because their testing ecosystems and syntax differ, results are reported separately rather than pooling the languages into a single undifferentiated benchmark [38]. Table 2 summarizes the data composition and evaluation units.

The main localizer experiments use terminal-aligned line states from Qwen2.5-Coder-1.5B-Instruct. RQ3 and RQ4 additionally use Qwen3-4B-Instruct-2507 and Qwen2.5-Coder-7B-Instruct to test whether the information findings depend on one representation model. All code models remain frozen; multi-layer states are extracted in one forward pass and reused offline [39,40].

5.2. Splits and Training Settings

The source-window experiments follow the fixed ten-fold protocol used by the Reference. Source-window records are assigned to ten folds after a seeded shuffle; each fold is used once for validation and the remaining folds for training. Because windows from the same buggy version are not grouped in this protocol, multiple windows of one version may enter both training and validation folds.

Under seed 42 this occurs for 54/501 BugsInPy versions and 45/385 Defects4J versions. Table 3 uses the same source-window ten-fold protocol as the Reference. Section 6.1 additionally reports a version-grouped ten-fold split in which all windows of a version occupy the same fold.

Every configuration is trained for five epochs with batch size 8 and seeds 42, 43, and 44. For a given seed, all compared configurations use the same data split and random-seed schedule, so within-seed differences do not arise from fold assignment.

All experiments were run on Ubuntu 22.04.5 with Python 3.12 and PyTorch 2.11.0 (CUDA 13.0).

Downstream localizers use AdamW with learning rate 10^{-3}, weight decay 0.01, hidden size 512, projection hidden size 1024, two layers, and dropout 0.1; Transformer-family models use eight heads and Pre-Norm. These settings are shared with the Reference and are held fixed across the main experiments. These values form a locked comparison protocol and are not tuned separately for each architecture or dataset; the goal is controlled comparability rather than a claim of global hyperparameter optimality. A scan of the relative coefficient between BCE and ListNet is reported in Section 7.5.

Table 3. Overall performance under Code-Only. Higher Top-*k* values are better, whereas lower MFR/MAR values are better. Values are version-level averages over three seeds. Bold values are better within each dataset.

Dataset	Configuration	Top-1 (%)	Top-3 (%)	Top-5 (%)	MFR	MAR
BugsInPy	Reference (Transformer+BCE)	50.23	70.73	78.91	6.253	13.379
BugsInPy	SeqRankFL (BiLSTM+Hybrid)	**58.88**	**74.72**	**81.44**	**5.373**	**11.663**
Defects4J	Reference (Transformer+BCE)	52.38	67.19	75.32	5.518	11.345
Defects4J	SeqRankFL (BiLSTM+Hybrid)	**58.27**	**72.12**	**78.10**	**4.389**	**9.817**
Equally weighted average	Reference (Transformer+BCE)	51.31	68.96	77.12	5.885	12.362
Equally weighted average	SeqRankFL (BiLSTM+Hybrid)	**58.58**	**73.42**	**79.77**	**4.881**	**10.740**

Leave-one-project-out (LOPO) tests within-window generalization to unseen projects. One complete project is held out at a time. No window from the held-out project participates in training or model selection, and the remaining projects are used for training.

Each experiment matrix changes only its target factor. RQ2 compares five localizer families (Transformer, gated Transformer, causal Transformer, CNN, and BiLSTM) with BCE, ListNet, and Hybrid objectives. Within the Transformer family, the structural designs are named Default, Compact, Post-Norm, Gated-Compact, Causal-Compact, Hierarchical-Pooling, Adapter-Fusion, and Cross-Attention. The class-imbalance analysis uses five positive-weight exponents and three focal parameters. RQ3 compares named representation views formed from four internal factors, and RQ4 compares Code-Only, static, and failure-input conditions. The complete configuration list and run records are released with the code rather than expanded in the main text.

5.3. Metrics and Version-Level Aggregation

We report five version-level metrics. Top-1, Top-3, and Top-5 indicate whether a true faulty line appears among the first 1, 3, or 5 ranked positions, respectively; higher values are better. MFR (mean first rank) is the average rank of the highest-ranked true faulty line in each version. MAR (mean average rank) is the mean, across versions, of the average ranks of all true faulty lines. Lower MFR and MAR values are better.

Because a buggy version may contain multiple source windows while the localizer operates on windows, metrics are first aggregated from windows to versions and only then across datasets. We use the following fixed rules.

1. **Window to version.** A version is a Top-k hit if any of its windows is a hit. MFR and MAR use the best window-level rank for that version. Each buggy version therefore contributes once, independent of its number of windows. The same version-level evaluator is used for SeqRankFL and the Reference; the evaluation unit is the buggy version. For versions with multiple windows, this any-window/best-rank aggregation can make the absolute metrics optimistic. BugsInPy contains 61/501 multi-window versions and Defects4J contains 50/385; these extra windows come from the same source file. Section 6.1 therefore reports a more conservative mean-window Top-1 contrast that averages Top-1 across windows within a version before averaging over versions.
2. **Random seeds.** The three seeds are averaged within the same method, dataset, and input condition. Seeds are not treated as independent buggy versions.
3. **Dataset.** BugsInPy and Defects4J are reported separately. The aggregate is the arithmetic mean of the two within-dataset values, giving both datasets equal weight despite their different sizes (501 and 385 buggy versions).
4. **Multiple models.** For RQ3 and RQ4, seeds are first averaged within each model–dataset–version cell, then within each model–dataset combination, and finally across the six combinations with equal weight.
5. **LOPO.** We report both version-weighted results, in which larger projects contribute more versions, and project-weighted results, in which every project contributes equally.

5.4. Research Questions

The experiments are organized around four questions concerning overall effectiveness, localizer design, internal representation information, and external input information.

- **RQ1 (overall effectiveness):** Does SeqRankFL improve version-level source-window ranking from frozen line representations and remain useful within windows under project isolation?
- **RQ2 (localizer design):** How do localizer architecture and training objective jointly affect ranking, and can class weighting replace ranking constraints?
- **RQ3 (internal representation information):** Can hierarchical, neighborhood, and syntax-scope information from frozen code models improve fault ranking?
- **RQ4 (external input information):** Can the gains from static and failure evidence transfer across models, languages, and projects?

6. Experimental Results

6.1. RQ1: SeqRankFL Improves Version-Level Source-Window Ranking

Table 3 compares the Reference with SeqRankFL under Code-Only. Both use the same Qwen2.5-Coder-1.5B-Instruct terminal-aligned line representations, ten-fold splits, seeds, and version-level evaluator. Only the downstream localizer and objective differ. The datasets are reported separately, followed by their equally weighted average.

On BugsInPy, Top-1 increases from 50.23% to 58.88%, an absolute gain of 8.65 percentage points. Top-3 and Top-5 increase by 3.99 and 2.53 percentage points, respectively. MFR decreases from 6.253 to 5.373, and MAR decreases from 13.379 to 11.663. The consistent improvements in all five metrics indicate that the effect is not limited to moving only a few versions to rank 1. Rather, true faulty lines move toward the top of the ranking overall.

On Defects4J, Top-1 increases from 52.38% to 58.27%, an absolute gain of 5.89 percentage points. Top-3 and Top-5 increase by 4.93 and 2.78 percentage points, respectively. MFR decreases from 5.518 to 4.389, and MAR decreases from 11.345 to 9.817. The absolute gain on the Java data is smaller than that on the Python data, but the directions and metric

pattern are consistent. The equally weighted two-dataset average increases from 51.31% to 58.58%, an absolute gain of 7.27 percentage points.

To quantify uncertainty on this primary comparison, we apply a paired bootstrap at the buggy-version level. For each version we first average Top-1 hits over the three seeds, then resample the 501 BugsInPy versions and the 385 Defects4J versions 10,000 times. Table 4 reports the Top-1 differences and 95% intervals. The BugsInPy gain is +8.65 percentage points, interval [+5.99, +11.31]; the Defects4J gain is +5.89 percentage points, interval [+3.38, +8.49]. Neither interval contains zero.

Table 4. Version-level intervals for the Table 3 comparison. Paired bootstrap, 10,000 resamples; the unit is the buggy version. Δ Top-1 is SeqRankFL minus Transformer+BCE.

Dataset	Versions	Δ Top-1 (pp)	95% Interval (pp)
BugsInPy	501	+8.65	[+5.99, +11.31]
Defects4J	385	+5.89	[+3.38, +8.49]

The direction can be checked at a finer granularity. Top-1 improves on both datasets, and all six paired random-seed comparisons (two datasets by three seeds) are positive. The single-seed differences range from +3.38 to +12.77 percentage points.

Under a version-grouped ten-fold split, in which all windows of a version occupy the same fold, Table 5 reports the primary comparison. The equally weighted Top-1 increases from 51.43% to 57.90%, an absolute gain of 6.46 percentage points (BugsInPy 51.70% to 58.22%; Defects4J 51.17% to 57.58%). All five metrics still improve together.

Table 5. Primary comparison under version-grouped ten-fold splits. Code-Only; values are version-level averages over three seeds. Bold values are better within each dataset.

Dataset	Configuration	Top-1 (%)	Top-3 (%)	Top-5 (%)	MFR	MAR
BugsInPy	Reference (Transformer+BCE)	51.70	73.45	79.91	5.904	13.092
BugsInPy	SeqRankFL (BiLSTM+Hybrid)	**58.22**	**75.52**	**81.97**	**5.351**	**11.879**
Defects4J	Reference (Transformer+BCE)	51.17	67.71	75.58	5.135	11.187
Defects4J	SeqRankFL (BiLSTM+Hybrid)	**57.58**	**73.16**	**80.26**	**4.223**	**9.693**
Equally weighted average	Reference (Transformer+BCE)	51.43	70.58	77.75	5.520	12.139
Equally weighted average	SeqRankFL (BiLSTM+Hybrid)	**57.90**	**74.34**	**81.11**	**4.787**	**10.786**

If Top-1 is first averaged across windows within a version and then over versions, both the Reference and SeqRankFL drop in absolute value, and the gap remains: on BugsInPy, Transformer+BCE is 46.69%, and SeqRankFL is 55.74%; on Defects4J the values are 49.42% and 55.22%.

Project-isolated results are reported in Section 6.4. Under Code-Only, LOPO Top-1 weighted by held-out versions is 50.37% for BugsInPy and 54.37% for Defects4J. The project-weighted values are 49.03% and 56.29%, respectively.

Under the same LOPO protocol, the Transformer+BCE Reference reaches version-weighted Top-1 of 46.17% on BugsInPy and 49.09% on Defects4J, and project-weighted values of 43.86% and 50.83%.

They are lower than the corresponding ten-fold results, showing that cross-project transfer incurs a loss. Nevertheless, SeqRankFL remains above the matched isolated Reference on both datasets, and Top-1 remains above 50%, indicating useful within-window ranking on unseen projects.

RQ1 answer. Under the same representations, splits, and evaluation protocol, replacing the downstream localizer with BiLSTM and using the Hybrid ranking objective improves all five version-level metrics on both datasets. The equally weighted Top-1

gain is 7.27 percentage points, and the version-level 95% intervals exclude zero on both datasets. Version-grouped ten-fold splits and the LOPO Reference comparison support the same direction.

6.2. RQ2: Joint Effects of Localizer Architecture and Training Objective

Table 6 compares localizer–objective combinations under the same Code-Only condition and aggregation rules as Table 3. All rows share representations, splits, and seeds, so the differences reflect downstream sequence modeling and objective design.

Table 6. Localizer and objective combinations. Values are equally weighted averages over the two datasets under Code-Only. Bold values are best in each metric column. Transformer, CNN, and BiLSTM form a complete 3×3 matrix with BCE, ListNet, and Hybrid.

Configuration	Top-1 (%)	Top-3 (%)	Top-5 (%)	MFR	MAR
Transformer+BCE (Reference)	51.31	68.96	77.12	5.885	12.362
Transformer+ListNet	51.09	70.70	77.57	5.409	12.529
Transformer+Hybrid	51.36	69.89	77.56	5.377	12.422
CNN+BCE	54.99	72.60	79.95	5.045	10.894
CNN+ListNet	55.99	73.27	80.37	4.963	10.901
CNN+Hybrid	56.19	**74.00**	**80.93**	4.886	10.795
BiLSTM+BCE	52.10	70.51	77.59	5.581	12.102
BiLSTM+ListNet	57.84	73.73	80.62	**4.813**	10.784
BiLSTM+Hybrid (SeqRankFL)	**58.58**	73.42	79.77	4.881	**10.740**

Changing only the objective has little effect. With the Transformer readout fixed, Transformer+Hybrid reaches 51.36%, nearly the same as the Reference at 51.31% (+0.05 percentage points), although MFR improves from 5.885 to 5.377. Changing only the readout gives BiLSTM+BCE at 52.10%, an increase of 0.79 percentage points. Both effects are much smaller than adopting BiLSTM and Hybrid together (58.58%; +7.27 percentage points).

Transformer+ListNet reaches 51.09% equally weighted Top-1 and 47.90% on BugsInPy, which is not above Transformer+BCE. In the tested matrix, ListNet mainly improves ranking for the sequential localizers and barely improves the Transformer readout.

ListNet accounts for most of the improvement. BiLSTM+ListNet reaches 57.84%, showing that replacing independent line discrimination with within-window listwise ranking is a primary driver. Adding BCE to form Hybrid raises Top-1 to 58.58% and gives the best MAR, indicating a complementary role for line-level discrimination. CNN+Hybrid is best on Top-3 and Top-5, so a listwise objective is not tied to one sequence family; its equally weighted Top-1 is, nevertheless, 2.39 percentage points below SeqRankFL.

We therefore retain BiLSTM+Hybrid, namely SeqRankFL, as the main configuration.

Table 7 reports structural and objective sensitivity. The configurations come from two localizer families, so they are grouped by family and evaluated against their matched references. Absolute values should not be compared across families.

The Transformer-family results show local gains without a comparable change in scale. Compact designs improve Top-1 by 0.17 to 2.15 percentage points relative to the matched Reference, with Causal-Compact reaching the family maximum of 53.24%. The organization designs improve Top-1 by 0.78 to 1.69 percentage points except for Cross-Attention. Every Transformer design remains below SeqRankFL at 58.58%, so rearranging the Transformer readout does not substitute for a ranking-aware objective.

At learning rate 10^{-3}, Post-Norm and Cross-Attention reach 12.20% and 25.64% Top-1. Cross-Attention uses a single summary token as context. After lowering the learning rate to the 10^{-4} scale, equally weighted Top-1 is 53.20% for Post-Norm and 53.33% for Cross-Attention, above the Transformer+BCE Reference at 51.31% and below SeqRankFL

at 58.58%. This recovery indicates optimization sensitivity under the shared fixed protocol rather than an intrinsic inability of either architecture to perform the task.

The BiLSTM-family results separate class cost from ranking alignment. The best Weighted-BCE setting, $\alpha = 0.25$, reaches 53.73%, 1.63 percentage points above unweighted BCE, but remains 4.85 percentage points below unweighted Hybrid. Increasing the weight further reduces performance, with $\alpha = 1$ falling to 49.02%. Adding weights to Hybrid does not improve Top-1: the four settings remain at or below 58.58%, although MAR reaches 10.426 at $\alpha = 0.75$. Class cost can therefore affect average rank without replacing the within-window ranking constraint. Focal loss degrades monotonically as γ increases, from 47.17% to 37.00% and then 31.29%; all settings are below unweighted BCE. In this sparse-positive task, hard-example focusing is not a substitute for listwise ranking.

Table 7. Structural and objective sensitivity. Values are equally weighted averages over the two datasets under Code-Only. Bold values are best within each localizer family. The Transformer-family reference is Transformer+BCE; the BiLSTM-family baseline is BiLSTM+BCE. Sensitivity analysis grouped by localizer family.

Localizer Family	Design	Top-1 (%)	Top-3 (%)	Top-5 (%)	MFR	MAR
Transformer	Default (Reference)	51.31	68.96	77.12	5.885	12.362
Transformer	Compact	52.90	70.55	**77.99**	5.730	**11.944**
Transformer	Post-Norm	12.20	27.38	38.40	20.050	33.832
Transformer	Gated-Compact	53.03	69.89	76.39	5.867	12.244
Transformer	Causal-Compact	**53.24**	**71.34**	77.00	**5.607**	11.903
Transformer	Hierarchical-Pooling	53.00	70.24	77.43	5.691	12.271
Transformer	Adapter-Fusion	52.09	69.48	76.76	5.773	12.305
Transformer	Cross-Attention	25.64	46.63	57.87	10.005	18.882
BiLSTM	BCE (BiLSTM Baseline)	52.10	70.51	77.59	5.581	12.102
BiLSTM	Weighted-BCE ($\alpha = 0.25$)	53.73	70.66	77.81	5.408	11.770
BiLSTM	Weighted-BCE ($\alpha = 0.50$)	53.48	71.07	77.37	5.416	11.538
BiLSTM	Weighted-BCE ($\alpha = 1$)	49.02	67.68	74.26	6.247	12.516
BiLSTM	ListNet	57.84	73.73	80.62	4.813	10.784
BiLSTM	Hybrid (SeqRankFL)	**58.58**	73.42	79.77	4.881	10.740
BiLSTM	Weighted-Hybrid ($\alpha = 0.50$)	58.37	**74.20**	**80.91**	**4.737**	10.611
BiLSTM	Weighted-Hybrid ($\alpha = 0.75$)	58.21	74.18	80.42	4.803	**10.426**
BiLSTM	Focal ($\gamma = 1$)	47.17	65.68	73.67	6.339	13.586
BiLSTM	Focal ($\gamma = 2$)	37.00	56.74	66.72	7.933	16.118
BiLSTM	Focal ($\gamma = 3$)	31.29	51.49	61.25	8.917	17.521

RQ2 answer. Ranking quality is determined jointly by localizer architecture and training objective. Replacing only the loss or only the localizer gives gains of +0.05 and +0.79 percentage points, respectively, whereas adopting BiLSTM and Hybrid together gives +7.27 percentage points. The listwise objective is the primary driver, with line-wise discrimination providing complementary information. This decomposition is descriptive and is not a formal statistical architecture-by-objective interaction test. Class weighting and hard-example focusing affect class cost but cannot replace ranking constraints. The best Transformer-family Top-1 is 53.24%, below SeqRankFL at 58.58%.

6.3. RQ3: Effects of Internal Representation Information

RQ3 fixes SeqRankFL and Code-Only and changes only how line representations are constructed from frozen states. The experiment covers three frozen models and two datasets. The three seeds are first averaged within each model–dataset–version cell, then

within each model–dataset combination, and finally across the six combinations with equal weight. Table 8 averages three models; Table 3 uses only Qwen2.5-Coder-1.5B-Instruct.

Each view that adds internal information is at least competitive with Final-Layer on the five metrics. The final layer is therefore not the only readable source of fault signals. The best view depends on the metric: Layer-Mix+Scope gives the highest Top-1, 0.856 percentage points above Final-Layer, whereas Layer-Mix+All gives the lowest MAR, 0.748 below Final-Layer. Later paragraphs therefore report both Top-1 and MAR.

The two metrics also differ in directional stability across model–dataset combinations. Relative to Final-Layer, Layer-Mix+Scope improves MAR in all six combinations, with gains from 0.235 to 0.972. Top-1 is positive in four combinations and negative in two, namely -0.532 percentage points for Qwen1.5B/BugsInPy and -0.519 percentage points for Qwen7B/Defects4J. Thus, average rank improves consistently in this experiment, whereas the first-hit rate does not.

Table 8. Comparison of internal representation views. SeqRankFL and Code-Only are fixed. In the table, "Neighbor" denotes Neighborhood-Residual and "All" denotes Depth-Residual+Neighborhood-Residual+Syntax-Scope. Values are equally weighted averages over six model–dataset cells. Bold values are best in each metric column.

Representation View	Top-1 (%)	Top-3 (%)	Top-5 (%)	MFR	MAR
Final-Layer	58.173	74.736	81.215	4.771	10.732
Layer-Mix+Scope	**59.029**	75.332	81.956	4.644	10.154
Layer-Mix+Neighbor	58.429	**75.495**	82.092	**4.497**	10.059
Layer-Mix+All	58.516	75.443	**82.108**	4.511	**9.984**

The metric definitions explain why these outcomes can diverge. MAR responds to the positions of all true faulty lines, so several lines can move upward without changing the line at rank 1. Top-1 changes only when the first-ranked line changes and is therefore more discrete. The observed pattern is consistent with multi-depth and structural coordinates moving faulty lines upward overall without always replacing the top-ranked line.

The factor decomposition does not support a "more features is better" rule. The average Top-1 marginal effects of Layer-Mix, Neighborhood-Residual, and Syntax-Scope are +0.296, +0.108, and +0.084 percentage points, respectively, whereas Depth-Residual has an effect of -0.168 percentage points. The Layer-Mix/Syntax-Scope interaction is +0.473 percentage points, while the Neighborhood-Residual/Syntax-Scope interaction is -0.693 percentage points. The opposite interaction signs indicate conditional dependence, so directly concatenating every computable feature is not guaranteed to improve Top-1. The best view also differs across the six model–dataset combinations.

RQ3 answer. Intermediate-to-late states and source-file structural coordinates contain information that is not fully retained in the final layer. Adding them can improve version-level ranking, with MAR improving in the same direction across all six model-dataset combinations and Top-1 improving in four combinations. The factors are conditionally dependent, and the best construction varies with the frozen model and dataset. We therefore report the availability and conditionality of internal information.

6.4. RQ4: Conditionality of External Input Information

RQ4 tests whether prepending external information to the line sequence produces a stable ranking gain. The static-input experiments fix the Transformer+BCE Reference, seeds, splits, and evaluation script, and change only the added input. They therefore isolate input effects. Table 9 reports Code-Only and the highest-Top-1 input design for each model–dataset combination.

Only four of the six combinations select an external input as the best design. For Qwen3-4B/BugsInPy and Qwen7B/BugsInPy, Code-Only is best, so no static input improves Top-1. The organization is also condition-dependent: CFG+DFG (Joint) is best for both Qwen1.5B combinations, whereas CFG+DFG (Paired) is best for the larger-model Defects4J combinations. Top-1 changes relative to Code-Only range from 0.260 to 1.530 percentage points, smaller than the 5.89 to 8.65 percentage-point gains from changing the localizer and objective in RQ1. The five metrics need not move together; for example, Qwen1.5B/Defects4J improves on Top-1, Top-3, MFR, and MAR, while Top-5 decreases from 75.32% to 74.11%.

Table 9. Static-input performance by frozen-model and dataset combination. The Transformer+BCE Reference, seeds, splits, and evaluator are fixed. Each combination lists Code-Only and the input design with the highest Top-1; Code-Only is listed alone when it is already the highest.

Frozen Model	Dataset	Input Design	Top-1 (%)	Top-3 (%)	Top-5 (%)	MFR	MAR
Qwen1.5B	BugsInPy	Code-Only	50.23	70.73	78.91	6.253	13.379
Qwen1.5B	BugsInPy	CFG+DFG (Joint)	51.76	71.46	78.51	6.148	13.105
Qwen1.5B	Defects4J	Code-Only	52.38	67.19	75.32	5.518	11.345
Qwen1.5B	Defects4J	CFG+DFG (Joint)	52.64	67.36	74.11	5.309	10.984
Qwen3-4B	BugsInPy	Code-Only	51.10	71.19	78.91	6.130	13.149
Qwen3-4B	Defects4J	Code-Only	53.59	67.71	75.84	5.176	10.919
Qwen3-4B	Defects4J	CFG+DFG (Paired)	54.11	69.70	75.84	5.154	10.669
Qwen7B	BugsInPy	Code-Only	51.70	72.12	78.91	5.807	13.197
Qwen7B	Defects4J	Code-Only	53.68	69.35	75.93	5.016	10.908
Qwen7B	Defects4J	CFG+DFG (Paired)	54.55	69.00	78.01	4.932	10.763

The DFG and failure-behavior analysis fixes the most complete representation view. Results are shown in Table 10.

Table 10. Effects of DFG and failure-behavior inputs. The representation is fixed to Layer-Mix+All (Depth-Residual+Neighborhood-Residual+Syntax-Scope), and values are equally weighted over six model–dataset combinations. Bold values are best in each column.

Input Design	Top-1 (%)	Top-3 (%)	Top-5 (%)	MFR	MAR
Code-Only	58.516	75.443	82.108	**4.511**	**9.984**
DFG	58.152	75.233	82.033	4.625	10.204
Failure	58.412	75.123	82.094	4.582	10.092
DFG+Failure	**58.675**	**75.991**	**82.243**	4.566	10.081

Relative to Code-Only under the same representation view, the Top-1 changes for DFG, Failure, and DFG+Failure are −0.364, −0.104, and +0.159 percentage points. MAR deteriorates by 0.221, 0.108, and 0.097, respectively. All changes are within 0.4 percentage points, and no input is best on all five metrics. DFG+Failure is highest on the three Top-k metrics but worse than Code-Only on both MFR and MAR.

The dataset-specific directions are clearer than the global averages. All three input types produce negative average Top-1 changes on BugsInPy: −0.843 percentage points for DFG and Failure and −0.577 percentage points for DFG+Failure. All three are positive on Defects4J: +0.115, +0.635, and +0.895 percentage points, respectively. The same disagreement appears across frozen models. For DFG, Qwen7B is positive on both datasets (+1.464 and +1.212 percentage points), whereas Qwen3-4B is negative on both (−2.794 and −1.385 percentage points). The same external information can therefore act in opposite directions across datasets and frozen models; a single global average cannot resolve this difference.

LOPO results further test whether external information transfers across projects, as shown in Table 11.

Under the same LOPO protocol, the same terminal-aligned representation, and Code-Only, Table 12 compares Transformer+BCE with SeqRankFL. SeqRankFL is 4.20 percentage points higher on BugsInPy and 5.28 percentage points higher on Defects4J in version-weighted Top-1.

CFG+DFG (Joint) increases version-weighted Top-1 by 0.47 percentage points in the BugsInPy LOPO setting but decreases it by 2.16 percentage points on Defects4J. Top-3, Top-5, MFR, and MAR change in the same directions, and the project-weighted results show the same pattern. This design is the best input for both Qwen1.5B combinations in Table 9 but becomes negative on Defects4J under project isolation. A ten-fold winner therefore cannot be extrapolated directly to unseen projects.

Table 11. LOPO results for SeqRankFL. The target project is excluded from training and model selection. Values aggregate three random seeds. The third column weights all held-out versions, and the fourth weights projects equally. Bold values indicate the better of the two input designs within each dataset for each metric column; higher Top-*k* values and lower MFR/MAR values are better.

Dataset	Input Design	Version-Weighted Top-1 (%)	Project-Weighted Top-1 (%)	Top-3 (%)	Top-5 (%)	MFR	MAR
BugsInPy	Code-Only	50.37	49.03	70.53	77.84	6.209	13.601
BugsInPy	CFG+DFG (Joint)	**50.83**	**49.27**	**71.19**	**78.51**	**6.124**	**13.292**
Defects4J	Code-Only	**54.37**	**56.29**	**67.79**	**75.84**	**5.751**	**11.418**
Defects4J	CFG+DFG (Joint)	52.21	54.86	65.89	73.94	5.864	11.742

Table 12. Transformer+BCE Reference under the same LOPO protocol. Code-Only; values aggregate three random seeds.

Dataset	Configuration	Version-Weighted Top-1 (%)	Project-Weighted Top-1 (%)	Top-3 (%)	Top-5 (%)	MFR	MAR
BugsInPy	Transformer+BCE	46.17	43.86	70.19	77.11	6.420	14.523
BugsInPy	SeqRankFL (Table 11)	50.37	49.03	70.53	77.84	6.209	13.601
Defects4J	Transformer+BCE	49.09	50.83	64.24	72.03	6.400	12.869
Defects4J	SeqRankFL (Table 11)	54.37	56.29	67.79	75.84	5.751	11.418

RQ4 answer. External inputs change how frozen representations are used, but their direction depends jointly on the frozen model, programming language, and project partition. Two of six model–dataset combinations select Code-Only as their best input. DFG and Failure are negative overall on BugsInPy and positive overall on Defects4J, and the ten-fold static-input winner reverses direction under project isolation. Within the tested scope, no input design provides a stable cross-condition gain. Code-Only is therefore the more robust starting point, and external inputs should be enabled only after target-domain validation.

7. Discussion

7.1. Matching Sequence Structure to the Ranking Objective

In the evaluated task, the localizer produces relative priorities within each source window, and the resulting metrics are aggregated by buggy version rather than treating every line as an independent decision. With only 2–3% positive lines among valid candidates, line-wise BCE is dominated by negative examples and leaves a mismatch between training

and evaluation. ListNet directly constrains the relative position of faulty lines within a window, which explains why the RQ2 decomposition shows little change when only the loss or only the localizer is replaced (+0.05 and +0.79 percentage points) but a 7.27-point gain when BiLSTM and Hybrid are adopted together. In the tested matrix, Transformer+ListNet is not above Transformer+BCE, so the listwise objective mainly benefits the sequential localizers evaluated here.

BiLSTM contributes a code-order inductive bias. A faulty line may depend on preceding definitions, later uses, and adjacent control structures, all of which are arranged in source order. The recurrent readout propagates information along this order, while the Transformer alternatives must learn the relevant positional pattern from the downstream data. CNN+Hybrid performs better on Top-3 and Top-5, so local convolution also works with a listwise objective; Top-1 remains the selection rule, and the main configuration is BiLSTM+Hybrid.

7.2. Class Cost Is Not Ranking Alignment

Sparse positives can suggest a class-imbalance remedy, but the RQ2 sensitivity analysis separates class cost from ranking alignment. Positive weighting changes the penalty for a line-level error; it does not require a faulty line to precede other candidates in the same source window. The best Weighted-BCE setting raises Top-1 only from 52.10% to 53.73%, remains 4.85 points below unweighted Hybrid, and falls to 49.02% at $\alpha = 1$. Focal loss similarly declines as γ increases.

Adding weights to Hybrid does not improve Top-1, although MAR reaches 10.426 at $\alpha = 0.75$. Class cost can therefore move several faulty lines upward on average without reliably changing the first-ranked line. We retain unweighted Hybrid as the main objective and treat reweighting as a sensitivity analysis.

7.3. Internal Representation Information Is Complementary but Conditional

The final layer of a frozen code model is optimized for the model's pretraining objective and need not retain every cue related to local syntax. Layer-Mix combines multiple intermediate-to-late states, while Syntax-Scope adds method- and class-internal coordinates and statement-family information. Their combination improves MAR in all six model–dataset combinations, consistent with complementary semantic-depth and structural cues.

MAR and Top-1 respond differently to the same ranking change. Several faulty lines can move upward and improve MAR without replacing the first-ranked line, so MAR improves in all six combinations while Top-1 improves in four. The factor decomposition shows a negative marginal effect for Depth-Residual and a negative Neighborhood-Residual/Syntax-Scope interaction. Directly concatenating every computable feature is therefore not a general rule, and the preferred representation view remains condition-dependent.

7.4. Relevant Information Is Not Automatically Usable

CFGs and DFGs describe dependencies related to fault propagation, and failure behavior is closer to the observed failure than static structure. Yet, RQ4 shows that relevance does not guarantee a stable gain. Frozen representations may already encode part of the same dependency information; version-level failure descriptions may not align with line-level ranking; and static-analysis quality can differ across language ecosystems. These factors are consistent with the negative direction on BugsInPy and positive direction on Defects4J.

The practical rule is therefore simple: start from Code-Only and enable external inputs only after validation on the target domain. Because external inputs do not change the upstream representation cache, this decision requires retraining only the lightweight localizer.

7.5. Relative Coefficient Between BCE and ListNet

The main method uses the unit-coefficient sum $\mathcal{L}_{\text{BCE}} + \mathcal{L}_{\text{ListNet}}$, corresponding to equal relative weighting of the two terms. Section 7.5 varies the relative coefficient using $\lambda\mathcal{L}_{\text{BCE}} + (1-\lambda)\mathcal{L}_{\text{ListNet}}$. Thus, $\lambda = 0.50$ has the same 1:1 relative weighting as the main objective, while other λ values shift the balance toward BCE or ListNet.

Table 13 compares $\lambda \in \{0.10, 0.25, 0.50, 0.75, 0.90\}$ on the BiLSTM localizer, together with ListNet-only, BCE-only, and the unit-coefficient main run. No non-equal λ improves both datasets simultaneously. Although $\lambda = 0.10$ gives the highest equally weighted Top-1 (59.11%), the gain is concentrated on BugsInPy (60.48%), while Defects4J decreases from 58.27% in the main run to 57.75%. We therefore retain equal relative weighting as the main cross-dataset configuration and treat the scan as sensitivity evidence rather than evidence for a universally optimal coefficient.

Table 13. Relative coefficient in the Hybrid objective. Code-Only, BiLSTM localizer; Top-1 (%). The $\lambda = 0.50$ row and the unit-coefficient main run have the same 1:1 relative weighting; they differ in overall loss scale.

Setting	Equally Weighted Top-1	BugsInPy	Defects4J
ListNet only ($\lambda = 0$)	57.84	57.75	57.92
$\lambda = 0.10$	59.11	60.48	57.75
$\lambda = 0.25$	57.76	57.95	57.58
$\lambda = 0.50$	58.37	58.82	57.92
$\lambda = 0.75$	58.69	58.42	58.96
$\lambda = 0.90$	58.69	59.28	58.10
BCE only ($\lambda = 1$)	52.10	50.70	53.51
Main method (unit-weight sum)	58.58	58.88	58.27

8. Threats to Validity and Limitations

Construct validity. Top-1, Top-3, Top-5, MFR, and MAR evaluate source-window rankings after the fixed window-to-version aggregation in Section 5.3, and LOPO adds project-isolation evidence.

Internal validity. A missing newline state at the end of a line could misalign representations and labels. Terminal-aligned extraction and line-level audits reduce this risk. Internal features do not read fault labels, patches, fault types, or test results, and class weights use only training-fold statistics. The main ten-fold protocol assigns source-window records to folds; because records are not grouped by buggy version, multiple windows of the same version may enter training and validation folds. Table 5 reports version-grouped ten-fold results, and LOPO provides project isolation. For versions with multiple windows, the any-window Top-*k* and best-window MFR/MAR aggregation can make absolute metrics optimistic; Section 6.1 therefore reports a more conservative mean-window Top-1 contrast. Post-Norm and Cross-Attention are strongly optimization-sensitive under the shared 10^{-3} protocol and recover to 53.20% and 53.33% after lowering the learning rate to the 10^{-4} scale; the extreme fixed-protocol rows should therefore not be interpreted as intrinsic architectural failure.

Statistical and comparative validity. Table 4 reports paired-bootstrap 95% intervals at the buggy-version level. Seeds are averaged within a configuration and are not independent buggy versions. The main localizer results use one frozen model and two datasets, whereas the representation and input audits use three models and two datasets; absolute values across these tables are therefore not directly comparable.

External validity. The datasets are limited to BugsInPy (Python) and Defects4J (Java), and the frozen models are limited to three publicly available Qwen code models. The results

do not represent other programming languages, closed-source models, industrial code repositories, or agent-based fault-localization workflows. Differences among the three models are also insufficient to attribute effects to instruction following, reasoning ability, or parameter scale because these factors were not independently controlled.

9. Conclusions

SeqRankFL is a sequence-aware ranker for statement-level fault localization from frozen code-LLM representations. It combines a BiLSTM readout with the BCE+ListNet Hybrid objective to model code-order context and within-window ranking, with metrics aggregated at the buggy-version level.

Under matched representations and evaluation protocols, SeqRankFL raises the equally weighted Top-1 from 51.31% to 58.58%, an absolute gain of 7.27 percentage points. BugsInPy and Defects4J improve by 8.65 and 5.89 percentage points, respectively, and all five version-level metrics improve on both datasets. The controlled results identify the listwise objective as the primary driver, with BiLSTM further improving Top-1 under that objective; class reweighting and hard-example focusing cannot replace the listwise constraint.

The information audit shows that Layer-Mix and Syntax-Scope can improve average ranks, while external-input effects depend on the frozen model, language, and project partition. No external input is a stable default in the tested conditions. SeqRankFL is a source-window ranker for known buggy files; additional representation views and external inputs should be validated separately for each target domain.

Author Contributions: Conceptualization, D.A., S.W. and B.L.; Methodology, D.A. and B.L.; Software, D.A.; Validation, D.A. and L.Z.; Formal analysis, D.A. and L.Z.; Investigation, D.A.; Resources, S.W.; Data curation, D.A. and L.Z.; Writing—original draft, D.A.; Writing—review & editing, D.A., S.W. and L.Z.; Visualization, D.A. and L.Z.; Supervision, S.W. and B.L.; Project administration, S.W. and B.L.; Funding acquisition, S.W. and B.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are openly available in SeqRankFL-62BD at https://anonymous.4open.science/status/SeqRankFL-62BD (accessed on 4 September 2026).

Acknowledgments: The authors gratefully acknowledge the editor and the anonymous reviewers for their comments that improved this manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wong, W.E.; Gao, R.; Li, Y.; Abreu, R.; Wotawa, F. A Survey on Software Fault Localization. *IEEE Trans. Softw. Eng.* **2016**, *42*, 707–740. [CrossRef]
2. Jones, J.A.; Harrold, M.J. Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique. In Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE), Long Beach, CA, USA, 7–11 November 2005; pp. 273–282. [CrossRef]
3. Abreu, R.; Zoeteweij, P.; van Gemund, A.J.C. An Evaluation of Similarity Coefficients for Software Fault Localization. In Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing (PRDC), Riverside, CA, USA, 18–20 December 2006; pp. 39–46. [CrossRef]
4. Wong, W.E.; Debroy, V.; Gao, R.; Li, Y. The DStar Method for Effective Software Fault Localization. *IEEE Trans. Reliab.* **2014**, *63*, 290–308. [CrossRef]
5. Li, X.; Li, W.; Zhang, Y.; Zhang, L. DeepFL: Integrating Multiple Fault Diagnosis Dimensions for Deep Fault Localization. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Beijing, China, 15–19 July 2019; pp. 169–180. [CrossRef]

6. Li, Y.; Wang, S.; Nguyen, T.N. Fault Localization with Code Coverage Representation Learning. In Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, Spain, 25–28 May 2021; pp. 661–673. [CrossRef]
7. Lou, Y.; Zhu, Q.; Dong, J.; Li, X.; Sun, Z.; Hao, D.; Zhang, L.; Zhang, L. Boosting Coverage-Based Fault Localization via Graph-Based Representation Learning. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Athens, Greece, 23–28 August 2021; pp. 664–676. [CrossRef]
8. Sohn, J.; Yoo, S. FLUCCS: Using Code and Change Metrics to Improve Fault Localization. In Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Santa Barbara, CA, USA, 10–14 July 2017; pp. 273–283. [CrossRef]
9. Meng, X.; Wang, X.; Zhang, H.; Sun, H.; Liu, X. Improving Fault Localization and Program Repair with Deep Semantic Features and Transferred Knowledge. In Proceedings of the IEEE/ACM 44th International Conference on Software Engineering (ICSE), Pittsburgh, PA, USA, 22–27 May 2022; pp. 1169–1180. [CrossRef]
10. Qian, J.; Ju, X.; Chen, X. GNet4FL: Effective Fault Localization via Graph Convolutional Neural Network. *Autom. Softw. Eng.* **2023**, *30*, 16. [CrossRef]
11. Yang, A.Z.H.; Goues, C.L.; Martins, R.; Hellendoorn, V.J. Large Language Models for Test-Free Fault Localization. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE), Lisbon, Portugal, 14–20 April 2024. [CrossRef]
12. Wu, Y.; Li, Z.; Zhang, J.M.; Papadakis, M.; Harman, M.; Liu, Y. Large Language Models in Fault Localisation. *arXiv* **2023**, arXiv:2308.15276. [CrossRef]
13. Kang, S.; An, G.; Yoo, S. A Quantitative and Qualitative Evaluation of LLM-Based Explainable Fault Localization. *Proc. ACM Softw. Eng.* **2024**, *1*, 1424–1446. [CrossRef]
14. Xu, C.; Liu, Z.; Ren, X.; Zhang, G.; Liang, M.; Lo, D. FlexFL: Flexible and Effective Fault Localization with Open-Source Large Language Models. *IEEE Trans. Softw. Eng.* **2025**, *51*, 1455–1471. [CrossRef]
15. Papadakis, M.; Traon, Y.L. Metallaxis-FL: Mutation-Based Fault Localization. *Softw. Test. Verif. Reliab.* **2015**, *25*, 605–628. [CrossRef]
16. Sarhan, Q.I.; Beszédes, Á. A Survey of Challenges in Spectrum-Based Software Fault Localization. *IEEE Access* **2022**, *10*, 10618–10639. [CrossRef]
17. Qin, Y.; Wang, S.; Lou, Y.; Dong, J.; Wang, K.; Li, X.; Mao, X. AgentFL: Scaling LLM-Based Fault Localization to Project-Level Context. *arXiv* **2024**, arXiv:2403.16362. [CrossRef]
18. Qin, Y.; Wang, S.; Lou, Y.; Dong, J.; Wang, K.; Li, X.; Mao, X. SoapFL: A Standard Operating Procedure for LLM-Based Method-Level Fault Localization. *IEEE Trans. Softw. Eng.* **2025**, *51*, 1173–1187. [CrossRef]
19. Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2020, Online, 16–20 November 2020; pp. 1536–1547. [CrossRef]
20. Guo, D.; Ren, S.; Lu, S.; Feng, Z.; Tang, D.; Liu, S.; Zhou, L.; Duan, N.; Svyatkovskiy, A.; Fu, S.; et al. GraphCodeBERT: Pre-Training Code Representations with Data Flow. In Proceedings of the 9th International Conference on Learning Representations (ICLR), Online, 3–7 May 2021.
21. Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H.P.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. Evaluating Large Language Models Trained on Code. *arXiv* **2021**, arXiv:2107.03374. [CrossRef]
22. Rozière, B.; Gehring, J.; Gloeckle, F.; Sootla, S.; Gat, I.; Tan, X.E.; Adi, Y.; Liu, J.; Sauvestre, R.; Remez, T.; et al. Code Llama: Open Foundation Models for Code. *arXiv* **2023**, arXiv:2308.12950. [CrossRef]
23. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *arXiv* **2017**, arXiv:1706.03762.
24. Peters, M.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep Contextualized Word Representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), New Orleans, LA, USA, 1–6 June 2018; pp. 2227–2237. [CrossRef]
25. Tenney, I.; Das, D.; Pavlick, E. BERT Rediscovers the Classical NLP Pipeline. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), Florence, Italy, 28 July–2 August 2019; pp. 4593–4601. [CrossRef]
26. Jawahar, G.; Sagot, B.; Seddah, D. What Does BERT Learn about the Structure of Language? In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), Florence, Italy, 28 July–2 August 2019; pp. 3651–3657. [CrossRef]
27. Hewitt, J.; Manning, C.D. A Structural Probe for Finding Syntax in Word Representations. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, MN, USA, 2–7 June 2019; pp. 4129–4138. [CrossRef]
28. Ferrante, J.; Ottenstein, K.J.; Warren, J.D. The Program Dependence Graph and Its Use in Optimization. *ACM Trans. Program. Lang. Syst.* **1987**, *9*, 319–349. [CrossRef]
29. Allamanis, M.; Brockschmidt, M.; Khademi, M. Learning to Represent Programs with Graphs. *arXiv* **2017**, arXiv:1711.00740.

30. Burges, C.; Shaked, T.; Renshaw, E.; Lazier, A.; Deeds, M.; Hamilton, N.; Hullender, G. Learning to Rank Using Gradient Descent. In Proceedings of the 22nd International Conference on Machine Learning (ICML), Bonn, Germany, 7–11 August 2005; pp. 89–96. [CrossRef]
31. Cao, Z.; Qin, T.; Liu, T.Y.; Tsai, M.F.; Li, H. Learning to Rank: From Pairwise Approach to Listwise Approach. In Proceedings of the 24th International Conference on Machine Learning (ICML), Corvallis, OR, USA, 20–24 June 2007; pp. 129–136. [CrossRef]
32. Sahoo, P.; Singh, A.K.; Saha, S.; Jain, V.; Mondal, S.; Chadha, A. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. *arXiv* **2024**, arXiv:2402.07927. [CrossRef]
33. Liu, N.F.; Lin, K.; Hewitt, J.; Paranjape, A.; Bevilacqua, M.; Petroni, F.; Liang, P. Lost in the Middle: How Language Models Use Long Contexts. *Trans. Assoc. Comput. Linguist.* **2024**, *12*, 157–173. [CrossRef]
34. He, H.; Garcia, E.A. Learning from Imbalanced Data. *IEEE Trans. Knowl. Data Eng.* **2009**, *21*, 1263–1284. [CrossRef]
35. Lin, T.Y.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. Focal Loss for Dense Object Detection. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 2999–3007. [CrossRef]
36. Widyasari, R.; Sim, S.Q.; Lok, C.; Qi, H.; Phan, J.; Tay, Q.; Tan, C.; Wee, F.; Tan, J.E.; Yieh, Y.; et al. BugsInPy: A Database of Existing Bugs in Python Programs to Enable Controlled Testing and Debugging Studies. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Virtual Event, 8–13 November 2020; pp. 1556–1560. [CrossRef]
37. Just, R.; Jalali, D.; Ernst, M.D. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA), San Jose, CA, USA, 21–25 July 2014; pp. 437–440. [CrossRef]
38. Rezaalipour, M.; Furia, C.A. An Empirical Study of Fault Localization in Python Programs. *Empir. Softw. Eng.* **2024**, *29*, 92. [CrossRef]
39. Hui, B.; Yang, J.; Cui, Z.; Yang, J.; Liu, D.; Zhang, L.; Liu, T.; Zhang, J.; Yu, B.; Lu, K.; et al. Qwen2.5-Coder Technical Report. *arXiv* **2024**, arXiv:2409.12186. [CrossRef]
40. Qwen Team. Qwen3 Technical Report. *arXiv* **2025**, arXiv:2505.09388. [CrossRef]