Appendix A. Supplementary materials for experimental settings

Hardware and random seed settings. All experiments in this study were conducted on a server equipped with an NVIDIA Tesla P40 24GB GPU, an Intel(R) Xeon(R) Gold 5118 CPU, 100GB of RAM, and running CentOS 7.6. For experiments requiring multiple runs, we used five random seeds: 42, 88, 1234, 2024, and 2048. These seeds include the default seeds commonly adopted in the official implementations of the evaluated models, ensuring that our results are consistent with widely used baselines while also covering additional seeds to assess robustness under diverse random initialization.

For the three CCPTMs (CCBERT, CCRep, and CCT5) and the five JIT-DP models (DeepJIT, SimCom, JIT-Fine, JIT-Smart, and CCT5+EF). At least two authors independently reproduced the relevant JIT-DP experiments for each model based on their official open-source implementations. After confirming that the reproduced results were generally consistent with those reported in the original papers, we extended the implementations by adding any missing evaluation metric calculation (consistent with the current open-source CodeT5+RF/CodeT5+EF implementation) and reorganized our preprocessed Unified *LApredict* dataset into the input formats expected by each model. All experiments were then repeated across all projects using the selected random seeds.

Specifically, the different JIT-DP models originally adopt diverse dataset organizations and input formats, which makes directly applying a unified dataset non-trivial. To ensure fair and reliable reproduction, we strictly referred to the original dataset structures used by each model and carefully preprocessed the Unified *LApredict* dataset to match the exact input format and file organization expected by each baseline. This strategy minimizes the need to modify the official data loading and processing code of the original implementations, thereby reducing the risk of introducing human bias or unintentional errors.

Besides, it is worth noting that CCRep leverages CodeBERT to generate embedding representations for both pre-change and post-change code snippets, utilizing a specialized "query back" mechanism to construct feature vectors for code changes. This mechanism supports three granularity modes (token-, line-, and hybrid-level); in our study, we adopted the line-level mode, which has been empirically demonstrated to achieve optimal performance for JIT-DP tasks.

Regarding JIT-Smart, we disabled its Defect Localization Network (DLN) for two reasons: (1) Dataset Limitation: The *LApredict* dataset lacks line-level defect labels required by DLN. (2) Minimal Impact: Both the original JIT-Smart paper and our replication experiments confirmed that DLN's influence on JIT-DP performance is marginal-the difference in key metrics with/without DLN is consistently below 1%, suggesting its exclusion does not compromise validity.

For the two representative CodePTMs (CodeBERT and CodeT5). We adopt the same fine-tuning framework as CCT5, with necessary enhancements to support more comprehensive metric calculation and adjustments to ensure correct extraction of input embeddings from each model.

Specifically, the classification module for both CodeT5 and CCT5 is implemented by inheriting from the official `T5ForConditionalGeneration` class, while CodeBERT employs the `RobertaModel` class. For all these PTMs, we use only their encoder parts and take the hidden vector corresponding to the position of the special classification token (i.e., [CLS]) in the input sequence as the representation vector for prediction.

All tokenizers, model checkpoints, and configuration files are obtained from the official Hugging Face repository. When multiple model sizes are available (e.g., small, base, large), we consistently select the base version, which is the most widely used in prior related work.

In addition, the runtime environment, including versions of Python, PyTorch, CUDA, and relevant dependencies, as well as all hyperparameters such as training steps, batch size, learning rate, and dropout, follow the default settings used in the original CCT5 implementation to ensure fair comparison and reproducibility.