BigIssue: A Realistic Bug Localization Benchmark

Anonymous ACL submission

Abstract

As machine learning tools progress, the inevitable question arises: How can machine learning help us write better code? With significant progress being achieved in natural language processing with models like GPT-3 and BERT, the applications of natural language processing techniques to code is starting to be explored. Most research has been focused on automatic program repair (APR), and while the results on synthetic or highly filtered datasets are promising, such models are hard to apply in real-world scenarios because of underperfoming bug localization techniques. We propose BigIssue: a realistic bug localization benchmark. The goal of the benchmark is two-fold. We provide (1) a general benchmark with a diversity of real and synthetic Java bugs and (2) a motivation to improve bug localization capabilities of models through attention to the full repository context. With the introduction of BigIssue, we hope to advance the state of the art in bug localization, in turn improving APR performance and increase its applicability to the modern development cycle.

1 Introduction

Recent advances in natural language processing (NLP) (Brown et al., 2020; Devlin et al., 2018; Liu et al., 2019b) have increased interest in applying NLP techniques to code understanding. With the development of code encoders (Feng et al., 2020a; Kanade et al., 2020), this task is becoming increasingly more accessible and appealing. As research has jumped ahead into the task of Automated Program Repair (APR), the results have been not been adequate. Although synthetic datasets have largely been solved (see Section 2.1), models have been surprisingly underperforming on real-world datasets, many not even able to repair a quarter of the bugs in the Defects4J benchmark(which benchmark?) (Lutellier et al., 2020). This is despite research suggesting that current APR benchmarks suffer from a lack of diversity (Durieux et al., 2019). As a consequence, many APR models are prone to overfitting to specific datasets (Mousavi et al., 2020). Although interesting from an academic perspective, such tools would hardly be useful in a real industrial scenario.

We posit that the three major limitations to APR methods being used today are: (1) training to fix already located bugs rather than finding bugs and fixing them, (2) the inability of models to take large contexts into account, and (3) the reliance on information besides pure code. The first limitation is straightforward: patches have limited context outside of the lines immediately before and after each patch. It has been shown that APR performance improves significantly if a good fault localization algorithm is used to detect buggy code locations (Durieux et al., 2019) (Liu et al., 2019a). The second limitation prevents models from finding bugs that depend on the context of the program. Even for human readers many real-world bugs require a lot of program-specific context to be detectable. One of the most popular code encoders today (Feng et al., 2020a) only supports encoding of sequences up to 512 tokens, not nearly enough to process most Java files in real-world programs (on average 7.5k tokens with the RoBERTa tokenizer (Liu et al., 2019b)). The third limitation follows from the fact that the most common method for fault localization used today (SBFL) (Jones and Harrold, 2005) is heavily reliant on test cases exposing potentially buggy locations (see Section 2.2).

In order to advance the state of the art of both BL (Bug Localization) and APR (Automatic Program Repair) models, we introduce BigIssue. The major contributions of BigIssue include:

- A large collection of confirmed real-world bugs with line-level annotations. Each bug has been reported by live users to the GitHub Issues bug-tracking system and fixed via a commit or pull request. The dataset contains a total of $10,905$ bugs sourced from $4,233$ Java repositories.

- A very hard, long-sequence synthetic bug dataset. Perturbations in real code collected from GitHub are generated by InCoder (Fried et al., 2022), a state of the art code generation model.

- An empirical demonstration of the hardness of the real benchmark as compared to a synthetic dataset. Even with advanced synthetic bug generation techniques, the performance on real bugs of models trained on synthetic data will not be adequate, which calls for further research into realistic bug detection.

By providing a large and diverse dataset of synthetic and real bugs from a multitude of projects without any extra information outside of code, we hope to push the direction of research towards line-level long-context bug localization for better performance on APR tasks.

2 Prior Art

2.1 Automatic Program Repair

Since bug localization is fundamentally related to automatic program repair, we provide a brief survey of existing APR benchmarks and their drawbacks.

Real-world Benchmarks The Defects4J dataset (Just et al., 2014) has been widely used in automatic program repair. It consists of 357 (835 is version 2) bugs sourced from 10 (100) top open-source Java projects. Bugs are manually reviewed and each bug has at least 1 test case that exposes the bug. APR methods, however, are not successful enough on this real dataset for the models to be useful in real-world applications. The most recent state of the art model can only fix 67 out of 357 bugs (Yuan et al., 2022), while the two previous state of the art models could only fix 44 (Lutellier et al., 2020) and 57 (Jiang et al., 2021) bugs. This is despite recent research that suggests APR methods are overperforming on Defects4J as compared to other similar benchmarks (Durieux et al., 2019). Bugs.jar (Saha et al., 2018) is a similar dataset but with an expanded scope of 8 popular projects from the Apache foundation.

iBugs (Dallmeier and Zimmermann, 2007) presents a methodology of semi-automatic bug extraction from repositories, and provides a concrete dataset of applying the methodology to the ASPECTJ project. The method involves analyzing commit logs for signs that indicate a bug fix, extracting the pre-commit and post-commit versions of the repositories, and identifying test cases that represent the bug-fix. However, this dataset is fairly small and is only sourced from one repository.

Another widely used dataset is the `ManySStubs4J` dataset (Karampatsis and Sutton, 2020). It's a collection of many "stupid" bugs mined from 100 (1,000) top open-source Java repositories. The collection includes only those changes where the change is a single line of code and falls into one of pre-determined 16 categories of bugs. While convenient, it suffers from a lack of complicated bugs and highly selective criteria.

Learning-fixes (Tufano et al., 2018) is a collection of about 58,350 short methods mined from GitHub. Each of the methods was semantically idiomized and presented in the benchmark. The main limitation of this dataset is that it's a method-level dataset: each bug should be identifiable and fixable based on the context only present in that particular method. For real bugs, this is usually not the case.

DLFix (Li et al., 2020) is another dataset aimed at APR tasks. The dataset consists of almost 5 million methods, enhanced with metadata, and the corresponding fixed versions of the method for a particular repository. While interesting for limited cases, the method-level granularity as well as the necessity of building metadata for each method limits its usefulness, especially on longer methods.

Table 1 presents a comparison of existing APR benchmarks.

Synthetic Benchmarks A natural way to deal with the lack of data diversity in current real-world benchmarks is to create synthetic benchmarks by perturbing code. The simplest way to create code perturbations is to apply rule-based perturbations to a corpus of code (Kanade et al., 2020) or via a static oracle (such as a linter) (Berabi et al., 2021). Other datasets are generated via a separate perturbation model. SPoC (Kulal et al., 2019) uses a simple LSTM to generate lines of code that might be potentially buggy. DeepDebug (Drain et al., 2021) uses a more complicated model trained on reversed git commits to generate synthetic bugs. While attractive, there is significant evidence that good performance on these benchmarks does not translate to good performance on real-life bugs (Durieux et al., 2019). We also perform experiments in Section 5 that suggests that even good performance on sophisticated perturbation datasets does not translate well to fixing real bugs.

2.2 Using Existing Benchmarks for Bug Localization

Fault localization and fault prediction on their own have been severely understudied. According to a recent survey (Zou et al., 2019) current fault localization and prediction methods can't even localize half of the bugs in the Defects4J (Just et al., 2014) dataset. The most widely used and best-performing method for fault localization is Spectrum-based fault localization (SBFL) (Jones and Harrold, 2005). While elementary and simple to implement, it relies heavily on the quality and quantity of test cases, especially for large programs (Keller et al., 2017). The lack of scalability for this method motivates further research into the problem of bug localization.

3 BigIssue Synthetic Dataset

3.1 Motivation

Evaluation of approaches towards bug localization requires the construction of a dataset with known ground-truth. One methodology to create such dataset is to consider existing code and introduce erroneous perturbations in the form of samples drawn from a generative model. In prior art (Kulal et al., 2019), synthetic perturbations have been adopted on a function-level granularity with weak generative models such as small LSTMs. The underlying distribution of such synthetic dataset may be quite dissimilar to the distribution of realistic bugs, which occur in software engineering (Durieux et al., 2019). To decrease this discrepancy, in the following, we will advance this concept to file-level data and sample perturbations from a strong generative model.

Our synthetic dataset adopts the methodology of gathering "real" code as observations and introducing synthetic perturbations in the observations. Here, the perturbation is a rewrite of the original sequence of code into a perturbed sequence of code. In our approach, a portion of the original code is "masked out" and a generative model is recruited to "fill in" the masked out code. The "filled in" portion of the code constitutes the synthetic perturbation. The perturbation of the original code is assumed to likely to contain "errors".

While the above approach based on perturbations may appear obvious and trivial, the construction of such datasets is challenging. This is due to, (1) existing code is not guaranteed to be free of errors, (2) the definition or ontology of an "error" or "bug" itself is non-trivial, (3) creating synthetic perturbations which are difficult to discriminate from original observations and yet reflect the distribution of "real" errors is hard.

Prior art addresses these issues by (1) reducing the scope of the code to function or line-level, effectively reducing the span of code to n lines of code (Kanade et al., 2020) (Yasunaga and Liang, 2020) (Yasunaga and Liang, 2021), (2) introducing heuristic perturbations rules or pre-defining a set of categories in which "bugs" fall (Kanade et al., 2020) (Drain et al., 2021), or (3) perturbing a single line of code in simple programs (Yasunaga and Liang, 2020) (Drain et al., 2021). While this over-simplification is a reasonable first step, the resulting dataset may be quite far from realistic errors in the wild for which localization is deemed "useful" to a practitioner.

Our work addresses (1) and (2) by doing away with the notion of an "error" and instead shifting the conceptual thinking towards the distributions of "original" and "perturbed" observations. That is, our dataset is assumed to contain errors which are not identified in the ground-truth labels. The task of error localization is relaxed as the task of localization of perturbations. This relaxation allows us to consider file-level observations without the need for a strict definition of an "error". Such relaxed definitions enable the construction of a "sanity-check" dataset to easily identify bug localization potential in prospective models. In the following, we will provide details on the creation of such data-set and in particular address (3).

3.2 Dataset Construction

The underlying methodology of the creation of this dataset is (1) gather large amounts of file-level observations (i.e., real code), (2) to introduce synthetic perturbations from a strong generative models such that discrimination of "original" and "perturbed" observation is non-trivial, (3) and relax the task of "error localization" to the task of "pertur-

Dataset	Size	Granularity	Bug Length	Context	# of Repos	Filters
BigIssue	10,905	Line	Multi-line	Repository	4233	No
Defects4J(Just et al., 2014)	357 (835)	Line	Multi-line	Repository	5 (17)	No
Bugs.jar(Saha et al., 2018)	1158	Line	Multi-line	Repository	8	No
ManySStubs4J (Karampatsis and Sutton, 2020)	10,231 (63,923)	Line	Single-line	Repository	100 (1000)	Yes
iBugs (Dallmeier and Zimmermann, 2007)	369	Line	Multi-line	Repository	1	No
Learning-Fixes (Tufano et al., 2018)	58,350	Line	Multi-line	Method	-	No
DLFix (Li et al., 2020)	4,973,000	Method	Multi-line	Repository	8	No

Table 1: Comparison of Major Java Bug Detection Datasets.

bation localization". In the following, we describe the construction of such a dataset.

Observations In order to obtain large quantities of observations for the learning and evaluation of localization models, the proposed dataset is a compilation of public, non-personal information from GitHub consisting of permissively licensed Java code in October 2021. In particular, we gathered 8 million repositories between January 2014 and October 2021 annotated with at least 1 star and considered the subset of contained files containing Java code. The files must have an average length of ≤ 100 characters and a maximum line length of $1,000$. Files where $\geq 90\%$ of the characters are decimal or hexadecimal digits are also removed. Finally, exact duplicates based on their SHA-256 hash are removed, which amounts to a substantial portion of the raw data due to forks and copies of repositories. The resulting data-set comprises 96.56 GB of raw text.

Perturbations For realistic perturbations, we resort to a method known as "inpainting" for images or "infilling" for the textual domain. That is, a portion of a giving observation is occluded (or masked out). Then, the occlusion is reconstructed or "filled in" by a sample drawn from a generative model conditional on the non-occluded context. Recently, auto-regressive causal language models (Brown et al., 2020) have demonstrated to excel at this task for which the prompt may be treated as context and the auto-regressive sample conditional on the prompt as the in-painting while preserving the statistical regularities of the training data. However, the joint distribution over tokens is usually factorized in a left-to-right order over time, for which the causal mask constraints the infill samples to only take past context into account, but not future tokens. In our case of sampling realistic perturbations at random spans within a given observation, we wish to take both the code before and after the masked out span to be taken account, so that file-level consistency remains. To address this issue, we recruit an auto-regressive sampler which re-arranges the input sequence and associated causal masking such that sampling is conditional on both past and future context (Du et al., 2022) (Fried et al., 2022). To further reduce the gap between "real" and "perturbed" sequences, we chose a large scale language model, InCoder (Fried et al., 2022) with 1 billion parameters, and lowered the temperature of auto-regressive nucleus sampling to 0.8. This temperature value was selected by manual experimentation. Equipped with such sampler, a random span in the observation is removed and infilled with a sample drawn from the InCoder model. The length of the span is drawn from a uniform distribution with minimum length of 64 tokens and maximum length of 128 tokens. The generated sample is constrained to at most the length of the span.

To further improve the quality of perturbations, we recruit rejection sampling from the InCoder model for which drawn samples not satisfying the formal grammar of the programming language are rejected. Specifically, we (1) reject any files which are not syntactically correct[1], (2) reject files containing less than $2,048$ tokens, (3) reject perturbations for which 10 attempts of infill sampling

[1]To verify syntactical correctness of Java programs, we recruit the JAVALANG library: https://github.com/c2nes/javalang.

(with a minimum span length of 64 and maximum number of tokens of 128) did not result in a syntactically correct perturbation, (4) reject samples for which the Levenshtein distance between the unperturbed and perturbed sequence is smaller than 64 tokens or larger than 192 tokens to reject samples which are highly similar to the original sequence.

Task Our proposed "perturbation localization" task can be expressed in the form of a binary classification for which each line is labelled as either "original" or "perturbed". As such, the ground-truth labels indicate whether the line is a subsequence of the observation or was (potentially partially) perturbed by the sampler. Each file contains at most one such perturbation. The length of the input sequence is limited to at most $8,192$ tokens under the RoBERTa tokenizer (Liu et al., 2019b) with at most 512 lines per file.

3.3 Dataset Examples and Artifacts

Some samples from the synthetic dataset are presented in Appendix E, and all artifact details can be found in Appendix A.

4 BigIssue Realistic Benchmark

4.1 Motivation

Based on our observations about existing benchmarks from Section 2, we concluded that a new benchmark is needed to push the state of the art forward. Therefore, we created a benchmark that prioritized quantity over perceived quality, and one that focused specifically on NL-based line-level bug localization.

For this benchmark, we defined a line as "buggy" if it has been removed or modified in the issue patch. This allows us to avoid using tests as the ground truth for bugs in code. This definition also fits well with the usage of code encoders such as CodeBERT (Feng et al., 2020a) for line-level classification, as demonstrated in Section 5.

4.2 Benchmark Construction

First, we considered Java GitHub repositories created between January 2014 and October 2021. In order to ensure that we only filter out repositories that were intended for some form of public use, we only examined repositories with at least 1 star. We further filtered down the repositories to only those repositories that had GitHub Issues enabled and had licenses permitting use of their code (full list of licences is available in Appendix C). That gave us $4,233$ repositories.

Using the GitHub API we filtered through closed issues on these repositories. We only used public, non-personal information available through the API. In order to select issues that corresponded to bug fixes on that particular repository, we selected issues that either contained "bug", "fix", or "fixed" as separate words in the title and the body of the issue. We also included issues that contained the label "bug". We looked at issues with a corresponding "close" event, and we looked at the commit that was attached to the latest "close" event. This gave us a dataset of $23,924$ total closed issues. We further filter only those bugs that affect one Java file without test code. That yields $10,905$ bugs.

To verify the validity of our filters, we manually verified 100 sample issues. We manually verified the validity of 84 of the issues. A detailed breakdown can be found in Appendix D.

Similarly to iBugs (Dallmeier and Zimmermann, 2007), to identify buggy lines we examine the data from the hunks in the diff. If a line is (1) removed from the source file and (2) is not an import line (lines that begin with `import ...`), it is marked as buggy. In cases were hunks are exclusively adding code, we mark the two lines in the source before and after the change as buggy. Processing added code is not always straightforward: sometimes the added chunk is an outsourced piece of code from a different method. However, this simplification to the process was done to account for added code while minimizing potential impact of simply outsourced chunks.

Test-running frameworks Many of the benchmarks presented above use tests either as assistance in bug fixing or as a method of filtering bugs. We do not consider testing frameworks and tests as a criteria for whether a commit is a bug or not. Firstly, it was recently shown that unit tests on their own do not guarantee less failures inside the code (Chioteli et al., 2021) which implies that there are even more bugs inside the code that are not exposed by tests. Secondly, because we would be severely limiting the diversity and scope of our benchmark by forcing issues to include an exposing test case.

4.3 Benchmark Examples and Artifacts

Some samples from the synthetic dataset are presented in Appendix F, and all artifact details can be found in Appendix A.

5 Synthetic vs Realistic Bug Detection

In this Section, we conduct a preliminary analysis of the hardness of the BigIssues benchmark. Since the sequence length exceeds the limitations of most pre-trained language models on code, we recruit mean pooling to construct simple baselines. We hypothesize that although the realistic data is much harder than the synthetic dataset, using long-context encoders in addition to synthetic pre-training will help increase performance.

5.1 Hypothesis

The proposed BigIssue benchmark contains two variants: (1) synthetic rewrites of real code sampled from a strong generative model, (2) realistic rewrites of real code based on the commits associated with a closed issue in GitHub.

Recall, for (1) a recent large language model was recruited as sampler which, compared to prior art, not only is of significant size under scaling laws, but furthermore alters the causal masking such that future tokens can be taken into account as context. We argue that these synthetic rewrites are non-trivial to detect compared to prior art.

However, our hypothesis is that localization of real bugs is still a significantly harder task, which requires substantial research to be solved. While local, trivial bugs do not require context to be localized, harder non-local bugs can often only be resolved when taking the entire file, a set of imported files, or the entire repository into account. Pre-training on synthetic data as well as long-context encoders will increase performance on realistic data.

5.2 Model

Our architecture partitions a long input sequence of $8,192$ tokens into shorter sub-sequences, computes contextualized vectors for each chunk using a bi-directional encoder model, combines the contextualized vectors into 512 latent vectors with mean-pooling, and finally projects those vectors to logits for line-level binary classification.

Consider a sequence $x = (x_0, x_1, \dots, x_n)$ of input tokens with length $n = 8,192$. To address the issue (2) of large n, we partition x into $m = 16$ equally sized chunks $\tilde{x}_i$ with $i \in \{0, \dots, 15\}$ each containing 512 tokens. To contextualize the embedding vector of the tokens, we recruit the pre-trained bi-directional encoder f, (such as CodeBERT (Feng et al., 2020b)), and compute $f(\tilde{x}_i)$ for each partition i. Then, the contextualized partitions are concatenated $\hat{x} = (f(\tilde{x}_0), f(\tilde{x}_1), \dots, f(\tilde{x}_m))$. To restore global position information, we apply additive sinusoidal positional embeddings to $\hat{x}$. A layer of self-attention integrates the information across partitions boundaries. Mean-pooling is applied to $\hat{x}$ with a window length such that the resulting sequence of latent vectors matches the maximum number of 512 lines. A standard linear projection maps each of the line-level latent vectors to logits for binary classification. The resulting model is fine-tuned with binary cross entropy as objective function.

The appeal of the proposed model is to leverage the representations learned by a strong backbone model and the simplicity in handling variable length including line breaks in the input sequence. CodeBERT (Feng et al., 2020b) has demonstrated strong empirical performance on down-stream tasks so that the learned representations should be well suited for bug localization. To demonstrate the utility of long context for code understanding, we also use the standard Longformer (Beltagy et al., 2020) as an encoder. The mapping of contextualized vectors to latent vectors allows for variable length input sequences and avoids special treatment of new line characters. The alignment from lines of the input sequence to latent vectors for classification is implicitly learned by supervision.

5.3 Findings

To evaluate the hardness of the artifical and realistic BigIssue benchmark, the aforementioned model is trained on both datasets. Training details can be found in Appendix B.

Table 2 summarizes the binary classification performance in terms of recall, precision and F1-score for three baseline models with CodeBERT encoder: (1) A random classifier for which the line-level predictions are modeled as a Bernoulli random variable per line with probability $p = 0.5$, (2) a mean-pooling based model for which the self-attention layer between latent vectors is omitted, (3) a mean-pooling based model including self-attention between latent vectors.

Model	Recall$^{\uparrow}$		Precision$^{\uparrow}$		F1$^{\uparrow}$	
	Synthetic	Realistic	Synthetic	Realistic	Synthetic	Realistic
Random	49.58	50.99	2.68	0.96	5.08	1.88
Pooling	93.48	69.43	8.89	2.16	16.24	4.17
Pooling-Attn	95.37	64.66	26.93	1.84	**42.00**	3.58

Table 2: Comparison of the binary classification accuracy under various baselines: (1) Random Bernoulli classifier with $p = 0.5$, (2) Mean pooling model, (3) Mean pooling model with self-attention between latent vectors.

Model	Training	Recall$^{\uparrow}$		Precision$^{\uparrow}$		F1$^{\uparrow}$	
		Synthetic	Realistic	Synthetic	Realistic	Synthetic	Realistic
Longformer-4096	Synthetic	98.49	42.98	22.62	3.74	**36.79**	6.88
Longformer-512	Synthetic	97.54	46.44	18.78	3.94	31.50	7.27
Longformer-4096	Real	73.28	75.40	5.92	2.65	10.96	5.12
Longformer-512	Real	81.28	88.68	5.95	2.46	11.09	4.79

Table 3: Comparison of Longformer Models. The numbers show that synthetic training is a suitable proxy task for realistic bug detection compared to exclusively realistic training and the advantages of long-context on synthetic data.

For the synthetic dataset, the mean-pooling model including self-attention with a F1-score of 42.00 significantly improves over the random Bernoulli baseline with 5.08. Self-attention to integrate information across latent vectors improves the score by nearly 26 points, which may indicate that attention across the partitioning of 512 tokens is crucial. One may assume with further improvements in modeling, the synthetic dataset is solvable.

For the realistic benchmark, however, both of the mean-pooling baselines performed better than random. The extra layer of attention did not add any improvement. Both models tend to have high recall values, but precision is especially low for the realistic benchmark.

To test the effect of longer-context encoders, we replaced the encoder in our Mean-Pooling with Attention model with a Longformer (Beltagy et al., 2020) that is capable of handling sequences up to 4096 tokens. Instead of chunking the sequence into 16 chunks of 512, we chunked it into 2 chunks of 4096. We trained a Longformer-4096 token Mean-Pooling with Attention model. We also trained the same model solely on realistic data. The results after $50,000$ steps of training are presented in Table 3. The results on synthetic data suggest that longer-context encoders significantly improve performance. Furthermore, a model trained on synthetic data gives better results on realistic data than a model trained exclusively on realistic data. This suggests that synthetic pre-training on simpler bugs helps the model detect more complicated bugs in real bugs. While the 512-chunked model performed better than the 4096-chunked model on realistic data, the difference cannot be described as significant due to low precision values.

As hypothesized, real bug detection is a much harder challenge than synthetic data. However, using longer contexts and pre-training on synthetic data improves the results on realistic benchmarks. It is our hope that this finding spurs research towards the modeling of long contexts to approach the task of real bug detection.

6 Conclusion

We propose a new benchmark to be used in assessing line-level bug localization models. The diversity and size of the dataset aim to provide a measure with realistic difficulty, encouraging larger context BL modeling that doesn't rely on project test suites. We also provide a synthetically generated benchmark, and show that although the perturbations can be sophisticated and borderline realistic, success on synthetically generated datasets does not transfer to realistic benchmarks.

We hope that our contributions inspire and push future research into realistic, long-context, NLP-based bug localization techniques. Advances in this area would bring automatic program repair to a state that would be useful and transformative to the modern software development process.

References

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Berkay Berabi, Jingxuan He, Veselin Raychev, and Martin Vechev. 2021. Tfix: Learning to fix coding errors with a text-to-text transformer. In *International Conference on Machine Learning*, pages 780–791. PMLR.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Efstathia Chioteli, Ioannis Batas, and Diomidis Spinellis. 2021. Does unit-tested code crash? a case study of eclipse. In *25th Pan-Hellenic Conference on Informatics*, pages 260–264.

Valentin Dallmeier and Thomas Zimmermann. 2007. Extraction of bug localization benchmarks from history. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 433–436.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dawn Drain, Colin B Clement, Guillermo Serrato, and Neel Sundaresan. 2021. Deepdebug: Fixing python bugs using stack traces, backtranslation, and code skeletons. *arXiv preprint arXiv:2105.09352*.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.

Thomas Durieux, Fernanda Madeiral, Matias Martinez, and Rui Abreu. 2019. Empirical review of java program repair tools: A large-scale experiment on 2,141 bugs and 23,551 repair attempts. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 302–313.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020a. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020b. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wentau Yih, Luke Zettlemoyer, and Mike Lewis. 2022. Incoder: A generative model for code infilling and synthesis. *arXiv preprint arXiv:2204.05999*.

Nan Jiang, Thibaud Lutellier, and Lin Tan. 2021. Cure: Code-aware neural machine translation for automatic program repair. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1161–1173. IEEE.

James A Jones and Mary Jean Harrold. 2005. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 273–282.

René Just, Darioush Jalali, and Michael D Ernst. 2014. 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*, pages 437–440.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020. Learning and evaluating contextual embedding of source code. In *International Conference on Machine Learning*, pages 5110–5121. PMLR.

Rafael-Michael Karampatsis and Charles Sutton. 2020. How often do single-statement bugs occur? the manysstubs4j dataset. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 573–577.

Fabian Keller, Lars Grunske, Simon Heiden, Antonio Filieri, Andre van Hoorn, and David Lo. 2017. A critical evaluation of spectrum-based fault localization techniques on a large-scale software system. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 114–125. IEEE.

Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. 2019. Spoc: Search-based pseudocode to code. *Advances in Neural Information Processing Systems*, 32.

Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. Dlfix: Context-based code transformation learning for automated program repair. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 602–614.

Kui Liu, Anil Koyuncu, Tegawendé F Bissyandé, Dongsun Kim, Jacques Klein, and Yves Le Traon. 2019a. You cannot fix what you cannot find! an investigation of fault localization bias in benchmarking automated program repair systems. In *2019 12th IEEE conference on software testing, validation and verification (ICST)*, pages 102–113. IEEE.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Thibaud Lutellier, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. 2020. Coconut: combining context-aware neural translation models using ensemble for program repair. In *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*, pages 101–114.

S Amirhossein Mousavi, Donya Azizi Babani, and Francesco Flammini. 2020. Obstacles in fully automatic program repair: A survey. *arXiv preprint arXiv:2011.02714*.

Ripon K Saha, Yingjun Lyu, Wing Lam, Hiroaki Yoshida, and Mukul R Prasad. 2018. Bugs. jar: a large-scale, diverse dataset of real-world java bugs. In *Proceedings of the 15th international conference on mining software repositories*, pages 10–13.

Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2018. An empirical investigation into learning bug-fixing patches in the wild via neural machine translation. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 832–837.

Michihiro Yasunaga and Percy Liang. 2020. Graph-based, self-supervised program repair from diagnostic feedback. In *International Conference on Machine Learning*, pages 10799–10808. PMLR.

Michihiro Yasunaga and Percy Liang. 2021. Break-it-fix-it: Unsupervised learning for program repair. In *International Conference on Machine Learning*, pages 11941–11952. PMLR.

Wei Yuan, Quanjun Zhang, Tieke He, Chunrong Fang, Nguyen Quoc Viet Hung, Xiaodong Hao, and Hongzhi Yin. 2022. Circle: Continual repair across programming languages. *arXiv preprint arXiv:2205.10956*.

Daming Zou, Jingjing Liang, Yingfei Xiong, Michael D Ernst, and Lu Zhang. 2019. An empirical study of fault localization families and their combinations. *IEEE Transactions on Software Engineering*, 47(2):332–347.

A Data Description, Hosting Details, and Data Access

We publish the training, evaluation, and validation sets for the synthetic data. We also publish the realistic benchmark. These items can be accessed in a Google Cloud Storage bucket at `https://console.cloud.google.com/storage/browser/bigissue-research`. All materials are released under the MIT License.

Realistic Pre-training data For the realistic Pooling and Pooling-Attention models, we created a pre-training dataset similar to other projects. We select Java GitHub repositories with 5 stars or more, we clone the main branch of the repository, while only downloading files under 2 megabytes. We then filter the commits that include the words "error", "bug", "fix", "issue", "mistake", "incorrect", "fault", "defect", "flaw", or "type", using standard practice in ManySStubs4J project (Karampatsis and Sutton, 2020) . Since our models are designed only for single-file bug localization, we take each modified file and apply the labeling procedure described in the paper to generate the examples and labels. We truncate files at 8192 tokens in the same manner as in (Feng et al., 2020a). In total, we get about 195 GB of data to use for pre-training.

B Training Details

We train all of our models on a single pod with 16 A100 GPUs. For models with the CodeBERT encoder, we optimized the model with a linear schedule AdamW (Loshchilov and Hutter, 2017) optimizer, with a starting learning rate of 5e-5, and $10,000$ warmup steps. We train over 50,000 steps with a batch size of 8. For models with the Longformer encoder, we optimized with a linear schedule AdamW optimizer, starting learning rate of 3e-5, and 1000 warmup steps.

We provide the full training code at `https://anonymous.4open.science/r/BigIssue-8EE2/`.

Model Checkpoints We provide the model checkpoints for the Pooling and Pooling-Attention models trained on realistic data in the GitHub repository `https://anonymous.4open.science/r/BigIssue-8EE2/`.

C Data Collection Ethical Statement

We did not collect any personal information from the GitHub API. We only collect commit information and data inside the commits, without taking into the account the origin or the user profile of the user making the changes.

We also present here the list of licences that we use in our paper: `https://anonymous.4open.science/r/BigIssue-8EE2/licences.txt`

D Realistic Bug Analysis

We've analyzed the quality of our selection process by sampling 100 random issues. This will allow other researchers to use this method with greater flexibility in licensing. We analyzed them based on three criteria: (a) whether the issue represents a valid bug (b) whether the fix represents a valid fix for the bug and (c) whether the fix depends on code conventions or APIs, and is therefore identifiable by a human with adequate knowledge of the above. The last criteria is to ensure possible identifiability by a model: if a bug is only marked as such because of outside software criteria, there is no reasonable way a model can learn that bug pattern.

In total, we have collected:

- 84 valid issues.
- 3 issues that did not represent valid bugs.
- 3 issues were the fixes did not fix the bugs.
- 2 issues that were not subject to analysis.
- 8 issues that are not identifiable without outside context.

Each item in the following list represents an issue considered for our dataset and is identified by the corresponding link to the github issue. We precede invalid issues with "*" for clarity.

1. * - `https://github.com/gsantner/markor/issues/314` - Valid bug, but the fix doesnt address the issue. In fact the issue persists despite the issue being closed.
2. `https://github.com/ReplayMod/ReplayMod/issues/422` - Valid bug, valid fix for the bug, bug is identifiable through similar usage of code around the buggy area.

3. https://github.com/lingochamp/FileDownloader/issues/855 - Valid bug, valid one-line fix for the file, bug is identifiable because ‘model.status‘ is being set in almost all action methods of this class.

4. https://github.com/danielCantwell/Fit-Friend/issues/3 - Valid bug, valid fix. Bug is identifiable by the fact that the variables used in that particular area of code are not used at all.

5. https://github.com/OpenJEVis/JEVis/issues/1699 - Valid bug, valid fix. Bug is an edge case if the JEVis samples are not sampled at a regular interval. Bug is identifiable because of similarities in time-series interactions.

6. https://github.com/TechReborn/TechReborn/issues/549 - Valid bug, valid fix. Bug is identifiable because (a) a lot of values are hard-coded and (b) some of the variables did not follow the human logic of what a minecraft operator is supposed to do (e.g. not checking if there is any space in the output container for more items to go to).

7. https://github.com/milaboratory/mixcr/issues/509 - Valid bug, valid fix, although there are extraneous style fixes. The style fix is that whenever the percentage is being used, the text for the log must be written as percent used . This follows a pattern from other places in this repository. The bug is that the percentages can sometimes be over 100

8. * - https://github.com/Angry-Pixel/The-Betweenlands/issues/895 - Valid bug, and valid fix, but the bug is not generally identifiable by humans without external context. The minecraft bug suggests that a certain item has to be not repairable, but from the code alone there seems to be no suggestion that this item should be unrepairable.

9. https://github.com/VazkiiMods/Quark/issues/2920 - Valid bug, valid fix. Identifiable by humans if they have knowledge and context about the Create library that cannot accept `FakePlayers` as players when performing operations on the world.

10. https://github.com/15knots/cmake4eclipse/issues/26 - Valid bug, valid fix. The essence of the bug is that certain build directories might be deleted outside of Eclipse, and that needs to be handled by the code, which is a common safeguard that code needs to implement.

11. https://github.com/TheThirdOne/JSoftFloat/issues/1 - Valid bug, valid fix. This bug should be identifiable with knowledge of floating-point arithmetic calculations.

12. https://github.com/ververica/flink-cdc-connectors/issues/326 - Valid bug, valid fix. To identify this bug, one must need to know that a certain parameter in the config can be null. In particular, if there is no pre-defined database history ID, one must be set.

13. https://github.com/spring-cloud/spring-cloud-sleuth/issues/333 - Valid bug, valid fix. Bug is identfiable based on existing usage from other Callable services, where the class is frequently used as a wrapper for calls.

14. https://github.com/tango-controls/rest-server/issues/192 - Valid bug, valid fix. Identifiable with knowledge of the TANGO API specification.

15. https://github.com/guillaume-alvarez/ShapeOfThingsThatWere/issues/7 - Valid bug, valid fix. The issue is that the camera movement logic is tied to frame-processing, and if that pattern is known then the bug is identifiable.

16. https://github.com/spring-cloud/spring-cloud-config/issues/128 - Valid bug, valid fix. Bug identifiable by common environment loading patterns.

17. https://github.com/BentoBoxWorld/

TwerkingForTrees/issues/9 - Valid bug, valid fix. Identifiable by minecraft logic of not allowing players to modify blocks outside of world border.

18. https://github.com/requery/requery/issues/63 - Valid bug, valid fix: known issue in SQLite https://stackoverflow.com/questions/28385069/sqliteopenhelper-setwriteaheadloggingenabled-causes-an-error-log-line.

19. * - https://github.com/mpcjanssen/ubiquitous-capture/issues/4 - Valid bug, valid fix. Not idenfitiable because this is fundamentally a UX bug.

20. https://github.com/assertj/assertj-vavr/issues/141 - Valid bug, valid fix. Identifiable based on other patterns in the same repository.

21. https://github.com/smartdevicelink/sdl_java_suite/issues/53 - Valid issue, valid fix. Identifiable through other similar patterns in similar code in the repository.

22. https://github.com/darcy-framework/darcy-webdriver/issues/30 - Valid bug, valid fix. Identifiable through other similar patterns in similar code in the repository.

23. * - https://github.com/AlexFalappa/nb-springboot/issues/167 - Valid bug, valid fix. Not identifiable.

24. https://github.com/GabrielOlvH/Carrier/issues/2 - Valid issue, fix not permanent, but does indeed correctly point to the location of the problem. The problem, broadly speaking, is caused by the fact that the Wolf entity is different from all of the other entities, and therefore calling `updateHolding` method on it will cause issues.

25. https://github.com/AgriCraft/AgriCraft/issues/82 - Valid issue, valid fix. Its logic that is identifiable by humans by looking at the variable names and intended usage.

26. https://github.com/rasmus-saks/aken-ajalukku/issues/65 - Valid issue, valid fix. Identifiable based on the context of the application, the fact that this is actually a walking tour, therefore the mode on google maps should be for walking rather than driving.

27. https://github.com/CJMinecraft01/DoubleSlabs/issues/81 - Valid issue, valid fix. Fix is identifiable in principle, but a lot of context about how minecraft slabs interact is needed.

28. https://github.com/hzi-braunschweig/SORMAS-Project/issues/6832 - Valid issue, valid fix. Bug is identifiable, the start date is replaced with Enddate in some cases on records.

29. https://github.com/MachinePublishers/jBrowserDriver/issues/67 - Valid issue, valid fix. The bug is identifiable in the long context and with knowledge of the general cookie-creation pattern. The problem is that the domain for the cookie is not set, so its not being used by the web-driver on repeat visits to a website.

30. https://github.com/release-engineering/pom-manipulation-ext/issues/240 - Valid issue, valid fix. This one just fixes an NPE, but it does contain a lot of style/whitespace changes.

31. https://github.com/Angry-Pixel/The-Betweenlands/issues/948 - Valid bug, valid fix. This one is in principle identifiable with knowledge of the pattern in minecraft servers to have different types of blocks that constitute a single entity (a door in this case).

32. https://github.com/ehcache/ehcache3/issues/2638 - Valid bug, valid fix. Bug is straightforward and identifiable.

33. https://github.com/thingsboard/thingsboard/issues/3992 - Valid bug, valid fix. The essence is that the method `getDeviceTypes` should call the /devices/types api endpoint rather than /devices. Should be identifiable based on semantic context.

34. https://github.com/vert-x3/vertx-config/issues/20 - Valid bug, valid fix. The bug is easily identifiable because (a) Vertx is often used in code, and (b) the host variable is left unused despite being declared, and there is only one logical place where it can be potentially used.

35. https://github.com/metarhia/jstp-java/issues/24 - Valid bug, valid fix. The bug is identifiable given context about the `ExecutionHandler` class.

36. * - https://github.com/Cassiobsk8/Industrial_Renewal/issues/126 - Valid bug, valid fix. This bug is difficult to identify because (a) the fix is to override a particular method of the class and (b) its not obvious that there can be an obstruction with bunk beds..

37. https://github.com/Tamaized/AoV/issues/13 - Valid bug, valid fix. Contains some whitespace additions in addition to bug fix. Bug related to a certain config option not being used, its identifiable through semantic understanding of the code.

38. * - https://github.com/PyvesB/advanced-achievements/issues/172 - Valid bug, valid fix. The bug is hard to identify, because it requires context about brewing stand operations.

39. https://github.com/eclipse/vorto/issues/442 - Valid bug, valid fix. The issue is identifiable by the fact that resource id is hardcoded to 0 rather than the resourceId variable provided. Also semantically identifiable. Contains whitespace changes as well.

40. https://github.com/hsyyid/AdminShop/issues/5 - Valid bug, valid fix. Identifiable bug.

41. * - https://github.com/labhackercd/edm/issues/5 - Valid bug, unsure if valid fix. The bug has little information, and the crash that the bug reports does not seem to be identifiable from the code alone (maybe its device-dependent, but the pattern used that is considered buggy is widely recommended, see https://stackoverflow.com/questions/2422562/how-to-change-theme-for-alertdialog).

42. https://github.com/Haptic-Apps/Slide/issues/655 - Valid bug, valid fix. A run-time exception from a method should be catched.

43. * - https://github.com/PortuguesDoSeculoXXI/PortuguesDoSeculoXXI/issues/48 - Unsure, hard to ascertain, there is a lot of text in portuguese

44. https://github.com/OpenJEVis/JEVis/issues/840 - Valid bug, valid fix. Although the text is in German, the bug is about processing a list of items into a menu rather than just one. This is not a new feature, since the items to be processed were always packaged in a variable-length list.

45. https://github.com/commons-app/apps-android-commons/issues/587 - Valid bug, valid fix. Links need to be sanitized.

46. https://github.com/twizmwazin/CardinalPGM/issues/86 - Valid bug, valid fix. A map cycle schedule was only set when time was under 5 seconds, so the scheduling needed to be moved out of that particular if statement.

47. * - https://github.com/assemblits/eru/issues/100 - Not a valid bug. The change is just a change to the title of an alert to set it to a class name rather than a generic connection failure message.

48. https://github.com/MachinePublishers/jBrowserDriver/issues/21 - Valid

bug, valid fix. The issue is that cookies come in a lot of formats, and not all of them were supported by jBrowser.

49. https://github.com/lucas-tulio/server-simulator/issues/6 - The bug is valid, fix invalid.

50. https://github.com/spring-cloud/spring-cloud-netflix/issues/1724 - Valid bug, valid fix. The property for preferIPAddress was not taken into account. The config is needed to be able to identify this issue.

51. https://github.com/Col-E/Recaf/issues/344 - Valid bug, valid fix. The previous version replaced all $ in a class name with ., but only the last one needs to be replaced by convention.

52. https://github.com/plan-player-analytics/Plan/issues/1313 - Valid bug, valid fix. The front-end called the wrong endpoint, and the backend was adjusted so that the front-end was calling the correct endpoint to get a list of players for a server.

53. * - https://github.com/cabaletta/baritone/issues/330 - Not really a bug, more like an extra feature. Also has the enhancement tag.

54. https://github.com/sosy-lab/java-common-lib/issues/19 - Valid bug, valid fix. The problem was that one of the iterators in a method that returned sorted list of two collections wasnt fully exhausted.

55. https://github.com/Cactiw/Timetable/issues/4 - Valid bug, valid fix. The issue was that an update task wasnt put into an async call, and therefore caused issues downstream. Putting it in async fixes the issue.

56. https://github.com/jooby-project/jooby/issues/1489 - Valid bug, valid fix. Pretty easily identifiable that the factory is closed rather than the session that was just checked in the surrounding if statement.

57. https://github.com/Gaming32/ArrayV-v4.0/issues/43 - Valid bug, valid fix. Method that obviously should have been synchronized based on surrounding code wasn't synchronized.

58. https://github.com/Zedd7/ZHorse/issues/25 - Valid bug, valid fix. The duplicate horse is not assigned a name, so when deleting it the message should display the original horsess name.

59. https://github.com/neo4j-contrib/neo4j-apoc-procedures/issues/303 - Valid bug, valid fix. The issue is that two nodes cannot have the same main key, and when merging two nodes the previous node was not deleted therefore causing an exception. The solution is that the properties of the source node should be stored, the source node deleted, and then the properties of the source node have to loaded into the target node from the stored variable.

60. https://github.com/mikepenz/CrossfadeDrawerLayout/issues/15 - Valid bug, valid fix. The bug is that when opening/closing the drawer, the state is not necessarily updated. The fix is to override the appropriate methods to update state. Identifiable by the common pattern of state updates when calling certain parent class methods.

61. https://github.com/wultra/powerauth-webflow/issues/345 - Valid bug, valid fix. The issue is that the message of the logger wasnt aligned with the exception being caught. There are also some whitespace changes added in the commit.

62. https://github.com/TeamLapen/Vampirism/issues/333 - Valid bug, valid fix. Off-by-one error, identifiable by knowing about minecraft item stacks as well as generally looking around the code.

63. https://github.com/home-climate-control/dz/issues/144 - Valid bug, valid fix. Authentication parameters passed to the bean were not actually set when creating the bean. Bug is identifiable.

64. https://github.com/kontalk/androidclient/issues/1264 - Valid bug, valid fix. The problem is that the push notification service would not be started as a foreground service and the system would kill it after 15 seconds. The fix is to start it in the foreground before. Identifiable because this is a common pattern in messaging applications.

65. https://github.com/BetonQuest/BetonQuest/issues/734 - Valid bug, valid fix. The problem is that the method used previously to detect entity deaths in Minecraft was sub-optimal, and it was replaced by a better method that was seen in a different minecraft plugin. Identifiable with knowledge of the spigot library usage patterns.

66. https://github.com/jmockit/jmockit1/issues/98 - Valid bug, valid fix. The problem is that sometimes types that should be `null` are not mocked as `null` objects. The fix addresses these cases with cascading types. Bug is identifiable with knowledge mocking patterns.

67. * - https://github.com/Freeyourgadget/Gadgetbridge/issues/529 - Invalid issue that has been deleted.

68. https://github.com/scenerygraphics/sciview/issues/181 - Valid bug, valid fix. One needs to call `setSize` on the panel before displaying it, and the fix addresses that.

69. https://github.com/ramack/ActivityDiary/issues/153 - Valid bug, valid fix. Identifiable through app context.

70. https://github.com/decarbonization/android-fonz/issues/26 - Valid bug, valid fix. Type mismatch in settings crashed the app. Identifiable from android property conventions.

71. https://github.com/ICIJ/datashare/issues/41 - Valid bug, valid fix. Fix the ISO code representations of languages by a) adding more enums and b) using both iso1 and iso2 codes to identify a language. Should be easily identifiable since both iso1 and iso2 parameters are passed in.

72. https://github.com/twizmwazin/CardinalPGM/issues/645 - Valid bug, valid fix. The fix is to use a more high-level API provided by the library in question, and simplify the existing code dramatically and add support for detecting TNT damage and account for points. Should be identifiable given the whole library context.

73. * - https://github.com/manoelcampos/cloudsimplus/issues/368 - Valid bug, valid fix. This is an issue that would be hard to identify since its configuration/resource usage based, and depends on system parameters a lot.

74. * - https://github.com/almosr/android-svg-code-render/issues/67 - Valid bug, valid fix. Not identifiable due to the bug being in templating.

75. https://github.com/dcm4che/dcm4chee-arc-light/issues/523 - Valid bug, valid fix. When a study is deleted, the number of studies that patients of this study participated in has to be decreased. Identifiable with understanding of relationship between patients and studies.

76. * - https://github.com/BCA-Team/Buildcraft-Additions/issues/356 - Valid bug, valid fix. Minecraft lasers fired underwater get stuck. The fix implements the logic of dissipating the laser once it hits lava or water. Not identifiable without knowing in-app logic about lasers and expected behavior upon hitting water or lava.

77. https://github.com/TotalHamman/BetterBlockExchanger/issues/7 - Valid bug, valid fix. The app was reading state from the previous state during a swap rather than the current state, causing an NPE.

78. https://github.com/BasicAirData/GPSLogger/issues/132 - Valid bug, valid fix. The contents of a

variable on which a switch was conditioned could be potentially null, and that caused an NPE.

79. https://github.com/OfficeDev/ews-java-api/issues/8 - Valid bug, valid fix. The bug is identifiable by the fact that there is an used variable, and the only place where it can be reasonably used is in a method override of a method inherited from the parent class.

80. https://github.com/dcm4che/dcm4chee-arc-light/issues/1180 - Valid bug, valid fix. The problem is that the program assigns the type of a SOP instance not according to the DICOM specification. Knowledge of the DICOM specification is necessary to identify the bug.

81. https://github.com/LMBishop/Quests/issues/281 - Valid bug, valid fix. The problem is that player quests are not restored to the player object once the player joined the server. Identifiable with common quest/server patterns.

82. * - https://github.com/danielricci/solitaire/issues/90 - Valid bug, invalid fixes.

83. https://github.com/hv0905/SchoolStoryCollection/issues/2 - Valid bug, valid fix, identifiable by human.

84. https://github.com/davidcorbin/mygcc-api/issues/21 - Valid bug, valid fix. The issue is that when processing an image the code searches for the last occurrence of .jpg, but this cannot be found and throws an error if the image URL is uppercase.

85. https://github.com/Tamaized/AoV/issues/104 - Valid bug, valid fix. The issue is that a player can just hop in and out of bed to recharge certain abilities, but in reality they need to fully sleep in the bed for that. Identifiable bug because this is a common pattern (you need to actually *sleep*) in many minecraft games.

86. https://github.com/Electroblob77/Wizardry/issues/513 - Valid bug, valid fix. The fix consists of using the stream and filter apis to avoid removing items from a list that can be concurrently modified. Identifiable with knowledge of concurrency mechanisms.

87. https://github.com/tsandmann/ct-sim/issues/62 - Valid bug (because the code does not correspond to the documentation comments), valid fix. Interestingly the documentation is in German.

88. https://github.com/voxelwind/voxelwind/issues/33 - Valid bug, valid fix. The problem is that the project implements a server for Minecraft: Pocket Edition, and it doesnt fully comply with the API contract, in particular with respect to the yaw parameters that clients pass in. Identifiable with knowledge of the API.

89. https://github.com/phrack/ShootOFF/issues/651 - Valid bug, valid fix. Wrong class was used to test if a shot color matched certain constants. Identifiable.

90. * - https://github.com/rmichela/GiantTrees/issues/31 - Not really a bug, this just implements mechanisms that silence warnings if certain resource files do not exist.

91. https://github.com/sriharshachilakapati/SilenceEngine/issues/38 - Valid bug, valid fix. If the vertices for a certain polygon are cleared, some parameters are set to maximum and minimum infinity, and some downstream methods throw errors. Fix is modification of these methods to account for cases when vertices are 0.

92. https://github.com/lsfusion/platform/issues/164 - Valid bug, valid fix. The method would not account for ftp files that did not exist, and the modification allows the method to handle non-existent ftp files as well.

93. https://github.com/spring-cloud/spring-cloud-sleuth/issues/1816 - Valid bug, valid fix. The sleuth

library was interfering with openfeigns circuitbreaker capabilities, the fix was to conditionally create the feign bean only if circuitbreaker was disabled. Identifiable by humans.

94. `https://github.com/vinaygaba/CreditCardView/issues/13` - Valid bug, valid fix. The issue is that the setter methods did not modify the actual state of the class. Easily identifiable by humans.

95. `https://github.com/AludraTest/aludratest/issues/36` - Valid bug, valid fix. Identifiable with context of the Selenium library.

96. `https://github.com/VazkiiMods/Quark/issues/3374` - Valid bug, valid fix. Simple fix to fix the formatting of chat events in certain cases of item links.

97. `https://github.com/Tamaized/AoV/issues/93` - Valid bug, valid fix. The problem is that casting the furious howl spell should only apply to a selected target. Identifiable through other examples of spells cast on a specific target in the library.

98. `https://github.com/spring-projects/spring-boot-data-geode/issues/55` - Valid bug, valid fix. Identifiable with knowledge of Spring beans.

99. `https://github.com/twizmwazin/CardinalPGM/issues/657` - Valid bug, valid fix. Identifiable through Bukkit/game conventions: the game sets player metadata, but doesnt remove it on match end.

100. `https://github.com/rundeck/rundeck-cli/issues/43` - Valid bug, valid fix. Annotation text doesnt align with documentation about command line option usage in the rest of the code.

E Synthetic Dataset Samples

E.1 Example of InCoder perturbation

We present an example of a sample perturbation generated by the InCoder model in Figure 1. The original observation is on the left-hand side, and the perturbed observation is on the right-hand side.

Remarkably, both sequences appear to be syntactically correct code. The auto-repressive sampler took future tokens into account. For example, the type resolution of the object `map` may be resolved by the return signature of the function `public static ChainMap<...>` which was not masked out and the invocation of `map.put(...)`. While the original code iterates over the list of objects `obj`, the perturbed code only considers the first element of the list, if the list contains a single element. Whether the rewrites constitute a "bug" depends on the definition of the term, as earlier discussed. However, given the context one can argue that the rewritten implementation seems less probable to follow the underlying intent.

E.2 Synthetic Samples with Explanations

In this section we provide a few samples of bugs generated in the second revision of the synthetic dataset as well as brief explanations of the perturbations in Figures 2, 3, and 4. The red lines mark lines affected by the perturbations.

In Sample 1 (2), the bug is obviously introduced because the password string is being printed out into STDOUT. This is valid Java code, bug a significant deviation from secure coding practices. The code in Sample 2 (3) endless recursion loop call. The code in Sample 3 (4) is from a Spring app that controls a Planet API. Every time a call to `addPlanet` API is made, the API returns `HttpStatus.BAD_REQUEST` every time regardless of whether the operation succeeded or not.

F Realistic Benchmark Samples

In order to show the necessity for long-context language models for bug localization, we demonstrate an example of a bug that is highly dependent on external context outside of the scope of the file where the bug is located. The issue [2] in question is related to a bug in a software project Catnip that provides a Discord[3] API wrapper in Java. The bug is that the Java plugin uses `java.awt.Color`[4] to store Color parameter for embeds. This class stores not only the RGB bits of the color, but also 8 extra alpha bits, constituting a 64 bit representation of each color. However, the Discord API only

[2]https://github.com/mewna/catnip/issues/105
[3]https://discord.com
[4]https://docs.oracle.com/javase/7/docs/api/java/awt/Color.html

```java
package com.github.yt.mybatis.utils;

import com.github.yt.base.exception.BaseErrorException;
import org.apache.commons.lang3.StringUtils;
import java.lang.reflect.Field;

public class BeanUtils {

    public static ChainMap<String, Object>
      getValueMap(Object... objs) {
        try {
            ChainMap<String, Object> map =
      new ChainMap<>();
            for (Object obj : objs) {
                if (null == obj) {
                    continue;
                }
                for (Class<?> c = obj.getClass(); Object.
     class!= c; c = c.getSuperclass()) {
                    for (Field field : c.getDeclaredFields())
      {
                        field.setAccessible(true);
                        Object value = field.get(obj);
                        if (null == value) {
                            continue;
                        }
                        if (field.getType().isAssignableFrom(
     String.class) && StringUtils.isEmpty((String) value)) {
                            continue;
                        }
                        map.put(field.getName(), value);
                    }
                }
            }
            return map;
        } catch (Exception e) {
            throw new BaseErrorException("Object to Map
     convert Error", e);
        }
    }
}
```

Original observation.

```java
package com.github.yt.mybatis.utils;

import com.github.yt.base.exception.BaseErrorException;
import org.apache.commons.lang3.StringUtils;
import java.lang.reflect.Field;

public class BeanUtils {

    public static ChainMap<String, Object>
      getFieldMap(Object... objs) {

        ChainMap<String, Object> map =
      new ChainMap<String, Object>();
        try {
            if (objs != null && objs.length == 1) {
                Object obj = objs[0];
                for(Class<?> c = obj.getClass(); Object.class
     != c; c = c.getSuperclass()) {
                    for (Field field : c.getDeclaredFields())
      {
                        field.setAccessible(true);
                        Object value = field.get(obj);
                        if (null == value) {
                            continue;
                        }
                        if (field.getType().isAssignableFrom(
     String.class) && StringUtils.isEmpty((String) value)) {
                            continue;
                        }
                        map.put(field.getName(), value);
                    }
                }
            }
            return map;
        } catch (Exception e) {
            throw new BaseErrorException("Object to Map
     convert Error", e);
        }
    }
}
```

Perturbed observation.

Figure 1: Sampled perturbation introduces a non-trivial rewrite, which may be considered as a "bug". *Left*: Original Java code iterates over a given list of objects (green highlight). *Right*: Perturbed Java code only considers the first object in the list, if the list contains precisely one element (red highlight).

accepts 48 bit representations of colors. The fix, therefore, consists of masking out the first 8 bits of the color representation of `java.awt.Color` and passing that into the API. The fixing hunk is shown in Figure 5.

For a human to be able to understand and identify this issue, they need to know about the Discord API, the common patterns of calling the Discord API, the peculiarities of color representations in the `java.awt.Color` class, and how to perform bit operations. We present an piece of code [5] available online that demonstrates calling the discord API. This code passes in a color for the embed frame as a 48-bit digit (Figure 6). Examples like this in training or in the context are necessary for the model to have a chance at locating this bug. Without the context, even a human observer cannot reliably mark this as buggy code.

[5]https://discordjs.guide/popular-topics/embeds.html#embed-preview

```
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(
   AuthenticationToken token) throws
   AuthenticationException {
       System.out.println("====");
       //token
       String realname = (String)token.getPrincipal();

       //
       User user = null;
       try {
          user = userService.findObjectByName(realname);
       } catch (Exception e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        if(user == null) {
            return  new SimpleAuthenticationInfo("!", false
   , getName());
        }
        //
        String password = getPasswordEncoder().
   encodeToString(user.getPassword());
        System.out.println(""+password);

        //AuthenticationInfo

        //activeUser
        ActiveUser activeUser = new ActiveUser();
        activeUser.setRealname(user.getRealname());
        activeUser.setPhone(user.getPhone());
        activeUser.setValid(user.getValid());


        //System.out.println(""+activeUser);
        //   - :
        //  AuthenticatingRealm  credentialsMatcher !
        SimpleAuthenticationInfo info =
                new SimpleAuthenticationInfo(activeUser,
   password,credentialsSalt,getName());
        SecurityUtils.getSubject().getSession().
   setAttribute("currentUser",user);
        return info;
    }
}
```

Figure 2: Revised Synthetic Dataset Sample 1

```
   public class LazyFragment extends Fragment {

       protected LayoutInflater inflater;
       private View contentView;
       private Context context;
       private ViewGroup container;

       @Override
       public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           context = getActivity().getApplicationContext();
       }

       @Override
       public  View onCreateView(LayoutInflater inflater,
   ViewGroup container, Bundle savedInstanceState) {
           this.inflater = inflater;
           this.container = container;
           onCreateiew(inflater, container, savedInstanceState
   );
           return contentView;
       }

       @Override
       public void onDestroyView() {
           super.onDestroyView();
           contentView = null;
           container = null;
           inflater = null;
       }
```

Figure 3: Revised Synthetic Dataset Sample 2

```
@Controller
@RequestMapping(value = "/api")
@CrossOrigin(origins = "*")
//@CrossOrigin(origins = "http://localhost:4200")
public class PlanetController {

// private PlanetService planetService; I DONT HAVE A
    SERVICE AT THE MOMENT

 private PlanetRepo planetRepo;

 public PlanetController(PlanetRepo planetRepo) {
   super();
   this.planetRepo = planetRepo;
 }


//  @RequestMapping(value = "/getPlanets", method =
     RequestMethod.GET)
 @GetMapping(value = "/getPlanets") //These are identical
     in configuration. http://localhost:8080/SpringMVC/mvc/
     api/getPlanets
 public @ResponseBody List<Planet> getAllPlanets(){
   return planetRepo.selectAll();
 }

 @GetMapping(value = "/getPlanetsWithParam", produces="
     application/json", params = {"id"}) //http://localhost
     :8080/SpringMVC/mvc/api/getPlanetsWithParam?id=2
 public ResponseEntity<Planet> getPlanetById(int id) {
   return new ResponseEntity<Planet>(planetRepo.
     selectPlanetById(id), HttpStatus.INSUFFICIENT_STORAGE);
 }


 @ResponseStatus(value = HttpStatus.ALREADY_REPORTED)
 @GetMapping(value = "/getPlanetWithUrl/{num}")
 public @ResponseBody Planet getPlanetByIdTwo(@PathVariable
     ("num") int id ) {
   return planetRepo.selectPlanetById(id);
 }


 @ResponseStatus(value = HttpStatus.BAD_REQUEST)
 @PostMapping(value = "/addPlanet")
 public @ResponseBody String addPlanet(@RequestBody Planet
     incomingPlanet) {

   /*
    * This method is executed when the user requests to add
      a new planet to
    * the database.
    *
    * The default behavior for MVC is to ignore the
    incoming JSON
    * and treat it like if it were a GET request.
    *
    * In our example, we would expect this request to  If
     the incoming JSON does NOT HAVE all the fields, it will
      provide just default values.
    */
   planetRepo.insert(incomingPlanet);

   return "Success";
 }

 @GetMapping(value = "/allTheHeaders")
 public ResponseEntity<String> allHeaders(@RequestHeader
    Map<String,String> allHeaders){

   //THIS IS NOTHING TO DO WITH MVC
   //This is from Collections (Week 1)
   for(Entry<String, String> entry: allHeaders.entrySet())
    {
     System.out.println(entry.getKey() + "\t" + entry.
    getValue());
   }

   HttpHeaders responseHeader = new HttpHeaders();

   responseHeader.set("Name", "Bobby");
   responseHeader.set("superSecrets", "********");

   return new ResponseEntity<String>("Success",
    responseHeader, HttpStatus.FORBIDDEN);
 }

}
```

Figure 4: Revised Synthetic Dataset Sample 3

```
@CheckReturnValue
public EmbedBuilder color(@Nullable final Color color) {
    if (color != null) {
        this.color = color.getRGB();
        // Mask off the alpha bits
        this.color = color.getRGB() & 0x00FFFFFF;
    }
    return this;
}
```

Figure 5: Hunk from sample issue from Catnip. This bug demonstrates the need for more context than file-level information.

```
const exampleEmbed = new EmbedBuilder()
  .setColor(0x0099FF)
  .setTitle('Some title')
  .setURL('https://discord.js.org/')
  .setAuthor({ name: 'Some name',
      iconURL: 'https://i.imgur.com/AfFp7pu.png',
      url: 'https://discord.js.org' })
  .setDescription('Some description here')
  .setThumbnail('https://i.imgur.com/AfFp7pu.png')
  .addFields(
    { name: 'Regular field title', value: 'Some value
 here' },
    { name: '\u200B', value: '\u200B' },
    { name: 'Inline field title', value: 'Some value
 here', inline: true },
    { name: 'Inline field title', value: 'Some value
 here', inline: true },
  )
  .addFields({ name: 'Inline field title', value: 'Some
 value here', inline: true })
  .setImage('https://i.imgur.com/AfFp7pu.png')
  .setTimestamp()
  .setFooter({ text: 'Some footer text here', iconURL: '
 https://i.imgur.com/AfFp7pu.png' });
```

Figure 6: Example of calling the Discord API with a RGB (24-bit) color representation while RGBA (32-bit) is expected