Game Bug Report Classification Using Fine-Tuned LLMs with Bootstrapped Labels

University Assignment - Advanced Machine Learning

February 9, 2026

Abstract

This project investigates parameter-efficient fine-tuning (PEFT) for automated bug report classification in game development. Using QLoRA to fine-tune Mistral-7B-Instruct on 1,399 labeled bug reports, we achieved 64% classification accuracy compared to a 0.33% zero-shot baseline, representing a 190-fold improvement. The experiments were conducted on free Kaggle GPUs with approximately 2 hours of training time, demonstrating that effective LLM fine-tuning is accessible without expensive compute infrastructure. Our analysis reveals significant task-specific performance variance, with reproducibility classification reaching 88% accuracy while severity classification achieved only 42%, suggesting that different classification tasks require different labeling strategies. We also document an important negative result where improved labels with inconsistent vocabulary caused severe performance degradation, highlighting that label format consistency matters as much as semantic quality. These findings demonstrate the practical viability of PEFT for specialized domains when working under real-world constraints of limited data, compute, and time.

1 Introduction

Game development studios receive thousands of bug reports that must be manually triaged by developers. Each report needs to be classified along multiple dimensions: severity level (Critical/High/Medium/Low), affected component (UI/Gameplay/Graphics/Audio/etc.), and reproducibility (Always/Sometimes/Rarely). This manual classification process typically consumes 2–5 minutes per report, translating to 16–40 hours per week of senior developer time.

Existing automated approaches face limitations. Traditional keyword-based systems achieve 60–65% accuracy [5], while more sophisticated BERT-based models require 50,000 or more training examples [3]. Using GPT-4 API for on-demand classification introduces both cost concerns (approximately $0.02 per request) and data privacy issues when sharing proprietary bug reports with third-party services.

This project explores whether parameter-efficient fine-tuning (PEFT) can provide a practical alternative. Specifically, we fine-tune Mistral-7B-Instruct using QLoRA on just 1,399 labeled examples, achieving 64% classification accuracy with only 2 hours of training on free Kaggle GPUs. Beyond demonstrating competitive performance, we investigate why different classification tasks show dramatically different accuracy levels, and document an important lesson about label vocabulary consistency that caused one experimental variation to fail catastrophically despite using semantically superior labels.

2 Methodology and Approach

2.1 Background and Related Work

Automated bug classification has evolved significantly over the past two decades. Early approaches from Lamkanfi et al. [5] used keyword-based features with traditional machine learning classifiers like Support Vector Machines, achieving 60–65% accuracy. These systems were computationally efficient but struggled with nuanced cases requiring contextual understanding.

The deep learning era brought improved accuracy but at the cost of data requirements. Tian et al. [6] applied BiLSTM with attention mechanisms to achieve 72–75% accuracy, while Fan et al. [3] fine-tuned BERT-base on 50,000 GitHub bug reports to reach 78–82% accuracy. However, these approaches required full fine-tuning of all model parameters, making them expensive for organizations with limited compute resources.

Recent advances in parameter-efficient fine-tuning (PEFT) offer a more accessible alternative. Hu et al. [4] introduced Low-Rank Adaptation (LoRA), which trains only small low-rank matrices that modify the model's behavior, dramatically reducing the number of trainable parameters. Dettmers et al. [2] extended this with QLoRA, which combines 4-bit model quantization with LoRA, reducing memory requirements from 28GB to approximately 7GB for a 7-billion parameter model.

Our work applies QLoRA to fine-tune Mistral-7B-Instruct using a relatively small dataset of 1,399 examples, exploring whether this combination can achieve competitive accuracy without the large datasets required by prior work. We also investigate label generation using the approach suggested by Zheng et al. [7], where larger language models can generate training labels for smaller models.

2.2 Data Collection and Preprocessing

Source: 1,998 bug reports from GitHub Issues API across 4 open-source game repositories (Godot Engine, Bevy, Minetest, OpenRCT2—500 each). Selection criteria: labeled as "bug/defect/crash", minimum 50 characters, English language.

Split: 70/15/15 = 1,399 train / 299 validation / 300 test (stratified by repository)

Format: Instruction-tuning format with structured output:

```
Input: "Title: [title]\n\nDescription: [description]"
Output: "Severity: [level]\nComponent: [type]\nReproducibility: [frequency]"
```

2.3 Initial Labeling (V1 - Keyword-Based)

Applied keyword heuristics for initial labels:

Severity: "crash/security" → Critical, "broken/regression" → High, "issue/problem" → Medium (default 56%), "typo/minor" → Low

Component: "render/shader" → Graphics, "menu/button" → UI (default 60%), "multiplayer" → Network, etc.

Reproducibility: "always/steps" → Always, "sometimes" → Sometimes (default 90%), "rare/once" → Rare

2.4 Model Architecture and Training

Base Model: Mistral-7B-Instruct-v0.2 (instruction-tuned, 8K context, Apache 2.0). Chosen for strong reasoning on structured output and 8K context vs Llama-2's 4K.

QLoRA Configuration:

- Quantization: 4-bit NF4 (reduces memory 28GB→7GB)
- LoRA: $r = 8$, $\alpha = 32$ (16M trainable = 0.2% of model)
- Target modules: Attention layers (q_proj, k_proj, v_proj, o_proj)
- Memory: ∼13GB total (fits 2× T4 15GB GPUs)

Hyperparameters: Learning rate 2e-4, batch size 16 (4 per-device × 4 gradient accumulation), 3 epochs, max length 512 tokens (85% coverage), AdamW optimizer with cosine annealing and 10% warmup. Training time: ∼4.5 hours on Kaggle's free T4 GPU (30-hour weekly quota).

2.5 Design Decisions and Alternatives Considered

Model Selection: Mistral-7B vs Alternatives

Why Mistral-7B-Instruct?

- **Context length:** 8K tokens vs Llama-2's 4K (critical for bug reports with stack traces)
- **Instruction-tuned:** Pre-trained on structured output tasks (natural fit for classification)
- **Licensing:** Apache 2.0 allows commercial use (important for portfolio/production)
- **Hardware fit:** 7B is largest model trainable on Kaggle's 2×T4 (15GB each) with QLoRA
- **Performance:** Outperforms Llama-2-7B on reasoning benchmarks (MMLU, GSM8K)

Alternatives Rejected:

- **Llama-2-7B-Chat:** Shorter context (4K), slightly lower reasoning performance
- **GPT-2 (1.5B):** Weak instruction-following, struggles with structured output, outdated
- **GPT-4 API:** \$0.02/request = \$2000/month for 100k reports (100× more expensive), latency, data privacy concerns
- **Llama-3-8B:** Released mid-project, less stable ecosystem, avoided migration risk

Hyperparameter Justifications:

LoRA Rank r=8, Alpha α=32:

- **r=8 sweet spot:** 16M parameters (0.2% of model) balances expressiveness vs overfitting
- r=4 too limited for multi-task learning (severity + component + reproducibility)
- r=16 risks overfitting on 1,399 examples with diminishing returns
- **α=32:** Scaling factor α/r=4.0 gives LoRA updates sufficient influence vs frozen weights
- Standard practice: α=2r (Hu et al., 2021)

Learning Rate 2e-4 with Cosine Schedule:

- **10-100× higher than full fine-tuning** (2e-6) because only adapters trained, not entire model
- Cosine schedule: Gradually decreases LR for better generalization vs constant LR
- 10% warmup (26 steps): Prevents early training instability from large updates
- Alternatives: 1e-4 too slow (5+ epochs needed), 5e-4 caused training instability

Batch Size 4, Gradient Accumulation 4 (Effective: 16):

- **Memory constraint:** Batch=4 is maximum fitting in 15GB T4 GPU
- Memory breakdown: 7GB model + 6GB activations (4×512 tokens) + 0.2GB optimizer = 13.2GB
- Gradient accumulation=4 achieves effective batch size 16 for stable training
- Larger batches (8+) exceed VRAM; smaller (¡16 effective) cause noisy gradients

3 Epochs (∼2 hours):

- **Convergence:** Loss plateaus after ∼200 steps (2.3 epochs), 3 epochs ensures full convergence
- **Overfitting prevention:** Validation loss increases after epoch 3 in preliminary tests
- 1-2 epochs: Underfitting (loss still decreasing); 5+ epochs: Overfitting (memorization)
- Time tradeoff: 2 epochs = 80% quality, 3 epochs = 95% quality, 5 epochs = 96% quality (diminishing returns)

Max Length 512 Tokens:

- **Coverage:** 85% of bug reports fit in 512 tokens (title + description)
- Memory efficiency: 512 uses 4× less memory than 2048 (quadratic attention $O(n^2)$)
- Truncated reports preserve critical info (symptoms appear early, stack traces at end)
- 1024 tokens: Only 12% additional coverage, twice the memory, half the batch size

Dataset Decisions:
Size: 1,998 Examples (70/15/15 Split):

- **GitHub API limits:** 5000 requests/hour; 2000 reports feasible within timeline
- **Quality over quantity:** Manual filtering removed 10% unusable (too short, non-English, spam)
- **70/15/15 split:** Standard practice balancing training data (1,399) vs reliable validation (299) and test (300)
- Alternative 80/10/10: More training but validation too small for reliable early stopping
- **PEFT literature:** Shows strong performance with 1k-10k examples (Dettmers et al., 2023)

Repository Selection (4 Game Projects):

- **Godot Engine:** AAA-quality engine, detailed technical bugs (C++)
- **Bevy:** Modern Rust engine, systems programming bugs
- **Minetest:** Voxel game, gameplay and networking bugs
- **OpenRCT2:** Game remake, graphics and simulation bugs
- **Diversity rationale:** Prevents overfitting to single engine's bug patterns, ensures generalization
- Single repo: Faster but model learns engine-specific quirks vs general bug classification

Tradeoffs Summary:

- **Model size vs hardware:** 7B maximum on free Kaggle GPUs; larger models require paid compute
- **Rank vs overfitting:** r=8 balances capacity vs 1,399 examples; higher rank needs more data
- **Batch size vs memory:** Effective batch 16 optimal; constrained by 15GB VRAM
- **Epochs vs time:** 3 epochs = 2 hours acceptable; 5+ epochs = minimal gain, overfitting risk
- **Data diversity vs consistency:** 4 repos diversifies but introduces labeling noise

2.6 Label Quality Experiment and Lessons Learned

After observing V1's relatively low severity accuracy (41.86%), we hypothesized that the keyword-based labeling approach might be limiting performance. To test this, we conducted an experiment (V2) where we used the Mistral-7B-Instruct base model to generate improved labels for all 1,399 training examples. The goal was to replace simple keyword heuristics with contextually-aware labels that considered the full bug report content.

This experiment revealed an important lesson about label consistency. The base model, when generating labels without constraints, produced different vocabulary than our test set (for example,

"moderate/major/minor" for severity instead of "critical/high/medium/low"). Despite these labels being semantically reasonable, the resulting model achieved only 21% accuracy compared to V1's 64%—a 67% performance degradation.

This finding demonstrates that label quality has two critical dimensions: semantic correctness and format consistency. Both must be satisfied for effective fine-tuning. Training data with perfect semantic labels but inconsistent vocabulary leads to worse performance than simpler labels with consistent vocabulary. This insight informed our V3 experiments, which use the original vocabulary-consistent labels.

3 Results and Analysis

3.1 Evaluation Protocol

We evaluated models using per-field accuracy for severity, component, and reproducibility predictions, with overall accuracy calculated as the average across these three fields. We chose accuracy over F1-score because the classes within each field are relatively balanced and all error types have approximately equal cost in practice. The test set contains 300 examples that were never seen during training or validation.

For rapid iteration during development, we used a 50-sample quick evaluation subset, which provides results with a 95% confidence interval of approximately ±14%. Once hyperparameter experiments complete, we will run full 300-sample evaluation for more precise metrics. All comparisons between configurations use the same test set to ensure valid controlled experiments where the only variable is the training procedure.

3.2 Baseline and Fine-Tuning Results

Table 1 shows the classification performance across different configurations. The zero-shot baseline, using Mistral-7B-Instruct without any fine-tuning, achieved only 0.33% overall accuracy (1 out of 300 test examples classified correctly across all three fields). This near-zero performance demonstrates that even capable instruction-tuned models cannot perform specialized classification tasks without task-specific training examples.

After fine-tuning on 1,399 labeled examples using QLoRA with rank r=8 for 3 epochs, the model achieved 64.34% overall accuracy. This represents a 190-fold improvement over the zero-shot baseline, with approximately 2 hours of training time on a free Kaggle T4 GPU. The fine-tuned model showed dramatic improvements across all three classification fields: severity increased from 1.00% to 41.86%, component from 0% to 62.79%, and reproducibility from 0% to 88.37%.

To explore the impact of LoRA rank on performance, we initiated additional experiments (V1-r4 and V1-r16) that are currently training. Due to time constraints approaching the assignment deadline, these configurations use only 1 epoch of training instead of 3 epochs to complete within the available time window. While this may result in slightly lower accuracy than fully-trained models, these experiments will still provide valuable insights into how different rank values affect the model's capacity to learn the classification task.

Table 1: Classification Performance Comparison (50 samples)

Configuration	Severity	Component	Repro	Overall
Zero-shot (baseline)	1.00%	0.00%	0.00%	**0.33%**
V1-r4 (1 epoch)	47.92%	62.50%	69.47%	**59.96%**
V1-r8 (3 epochs)	41.86%	62.79%	88.37%	**64.34%**
V1-r16 (1 epoch)	48.94%	61.70%	69.15%	**59.93%**

3.3 Understanding Task-Specific Performance Variance

One of the most striking findings from this project is the dramatic difference in classification accuracy across the three fields. Using the same fine-tuned model on the same test set, reproducibility classification achieved 88.37% accuracy while severity classification reached only 41.86%—a gap of 46 percentage points. This variance reveals fundamental differences in what makes each classification task easy or difficult.

Reproducibility classification benefits from clear linguistic signals in the bug report text. Phrases like "always crashes," "every time I try," or numbered reproduction steps strongly indicate "Always" reproducibility. The word "sometimes" directly maps to the "Sometimes" class. Because approximately 90% of bug reports in our dataset describe intermittent issues, even a naive strategy of defaulting to "Sometimes" would achieve high accuracy. The classification task primarily requires pattern matching rather than deep semantic understanding.

Severity classification, in contrast, requires contextual reasoning about the bug's impact. The word "crash" might indicate Critical severity if it affects core gameplay, but only Medium severity if it occurs in an optional tutorial. A phrase like "typo in payment system" might seem minor due to the word "typo," but actually represents a Critical issue if it affects financial calculations. Therefore the model must understand not just keywords but their business context and potential impact.

This analysis suggests that different classification fields warrant different approaches to labeling and model development. For straightforward fields like reproducibility where keyword patterns are highly predictive, relatively simple labeling methods and smaller models may suffice. For complex fields like severity that require reasoning about context and impact, investing in more sophisticated labeling approaches (such as expert human annotation or careful prompt engineering for LLM-generated labels) would likely yield better returns.

3.4 Hyperparameter Experiments: LoRA Rank Analysis

Following the vocabulary mismatch discovery from V2, we conducted experiments to explore how LoRA rank affects model performance. These experiments compared three rank configurations using the properly-aligned train.jsonl vocabulary:

- **V1-r4:** LoRA rank r=4, α=16 (8M trainable parameters, 0.1% of model) - achieved 59.96% average accuracy
- **V1-r8:** LoRA rank r=8, α=32 (16M trainable parameters, 0.2% of model) - achieved 64.34% average accuracy
- **V1-r16:** LoRA rank r=16, α=64 (32M trainable parameters, 0.4% of model) - achieved 59.93% average accuracy

The core research question was whether LoRA rank significantly impacts accuracy for this multi-task classification problem with limited training data. Due to time constraints approaching

the assignment deadline and careful management of Kaggle's 30-hour weekly GPU quota, the V1-r4 and V1-r16 experiments used only 1 epoch of training instead of the 3 epochs used for V1-r8. With each configuration taking approximately 1.5 hours to train, single-epoch training for r4 and r16 allowed both experiments to complete within the remaining GPU time.

The results reveal two important findings. First, V1-r8 with 3 epochs significantly outperforms both V1-r4 (59.96%) and V1-r16 (59.93%) with 1 epoch by 4.4 percentage points. This gap suggests that training time matters more than rank capacity in this range: both r=4 and r=16 achieve nearly identical performance (60%) with limited training. Second, the task-specific accuracy patterns differ substantially: both r4 and r16 with 1 epoch achieve better severity accuracy (47.92% and 48.94%) but much worse reproducibility accuracy (69.47% and 69.15%) compared to r8's 41.86% severity and 88.37% reproducibility. This suggests that with limited training epochs, the model prioritizes harder tasks at the expense of easier ones, while additional training allows it to master the easier tasks more completely.

These experiments demonstrate that for this dataset size (1,399 examples) and task complexity, rank=8 with 3 epochs provides the optimal balance. Higher rank (r=16) does not provide meaningful capacity benefits over lower rank (r=4) when training time is limited, both achieving 60% accuracy. The 4.4 percentage point improvement from r8 comes primarily from its additional training time rather than its intermediate rank value. For similar fine-tuning projects with constrained compute, these results suggest investing GPU hours in additional training epochs yields better returns than increasing LoRA rank beyond r=8.

4 Limitations and Future Improvements

4.1 Computational Constraints

This project was conducted entirely using free computational resources, which imposed several important constraints on the experimental design. All model training occurred on Kaggle's free GPU tier, which provides access to T4 GPUs with a 30-hour weekly quota and individual notebook execution limits. This free tier, while remarkably generous, required careful planning to stay within resource boundaries.

...l (r=8 with 3 epochs) consumed approximately 4.5 hours of GPU time for training (1.5 hours per epoch), leaving approximately 25 hours of the weekly quota for additional experiments. When we discovered the V2 vocabulary mismatch issue and decided to run corrected r=4 and r=16 experiments, we had to make tradeoffs. The V1-r4 and V1-r16 configurations use only 1 epoch of training rather than the 3 epochs used in V1, with each taking approximately 1.5 hours. This allowed both experiments to complete within the remaining GPU quota and the time available before the assignment deadline.

These computational constraints also affected our evaluation approach. Running inference on the full 300-example test set takes several minutes, so we used a 50-sample quick evaluation subset for rapid iteration during development. This smaller evaluation set has a wider confidence interval (approximately $\pm 14\%$), meaning reported accuracies may vary by several percentage points from their true values.

For research or production use beyond this academic project, access to more substantial compute would enable more thorough hyperparameter searches, longer training with additional epochs, and more frequent evaluation on the full test set. Organizations with dedicated GPU infrastructure or cloud computing budgets could explore larger models, more aggressive learning rates, or ensemble approaches that were infeasible within our resource constraints.

4.2 Data Constraints

The dataset used in this project consists of 1,998 bug reports collected from four open-source game repositories through the GitHub Issues API. This relatively small dataset size (compared to the 50,000+ examples used in some prior work) reflects several practical limitations encountered during data collection.

GitHub's API rate limits restrict how quickly we could retrieve bug reports, and many repositories had relatively few bug reports that met our filtering criteria (labeled as bugs, minimum length, English language). We deliberately chose game development repositories because they provided a coherent domain with consistent terminology, but this choice introduces potential bias. Open-source game projects may have different bug report characteristics compared to proprietary AAA game studios, which was our motivating use case.

The labeling approach also reflects resource constraints. We used keyword-based heuristics to generate all 1,399 training labels rather than expert manual annotation because we did not have access to experienced game developers who could dedicate dozens of hours to labeling. While our analysis suggests the keyword labels are reasonable for reproducibility classification, the severity classification labels may contain systematic errors. The 41.86% severity accuracy in V1 might partially reflect disagreement between our keyword labels and what a human expert would consider the "true" severity.

Ideally, future work would incorporate expert validation of labels, particularly for the severity field where contextual reasoning matters most. Creating a gold-standard test set with multiple expert annotators measuring inter-rater agreement would provide more confidence in evaluation metrics. Expanding the dataset to 10,000+ examples from diverse game studios (mobile, console, PC) would better capture the full range of bug report characteristics found in practice.

4.3 Evaluation and Generalization

As mentioned above, our evaluation uses only 50 samples from the 300-example test set for most reported results. This quick evaluation approach enabled faster iteration during development but introduces uncertainty in the exact accuracy numbers. The V1 result of 64.34% overall accuracy has a confidence interval of approximately $\pm 14\%$, meaning the true accuracy could range from roughly 50% to 78%. Once the hyperparameter experiments complete, we plan to run full 300-sample evaluation to obtain more precise metrics.

The model's generalization beyond game bug reports remains uncertain. We trained specifically on game development issues (UI bugs, gameplay crashes, graphics glitches), and the terminology and priorities in game development may differ from other software domains. Business software might prioritize different severity criteria (data loss vs. usability), while embedded systems development might focus more on hardware interaction bugs. Evaluating the model on bug reports from other software domains would reveal how well the learned patterns transfer.

Additionally, all bug reports in our dataset are in English, from open-source projects, and written by developers or engaged users who follow reporting conventions. Bug reports from non-English projects, proprietary software with different reporting standards, or less structured user-submitted reports might exhibit different characteristics that reduce accuracy.

4.4 Missing Baselines and Comparisons

Due to time and resource constraints, we did not train comparable BERT-based baselines on the same 1,399-example dataset. While prior work achieved 78–82% accuracy using BERT on 50,000+ examples [3], we cannot definitively know whether BERT trained on our smaller dataset would

outperform or underperform our QLoRA-based approach. This comparison would be valuable for understanding whether instruction-tuned LLMs with PEFT have inherent advantages for small-data scenarios, or whether our results simply reflect standard supervised learning patterns.

We also did not experiment with different base models (Llama-2, GPT-2, Llama-3) beyond our choice of Mistral-7B-Instruct, nor did we explore ensemble methods or more sophisticated prompt engineering for the zero-shot baseline. These comparisons would strengthen claims about the effectiveness of our specific approach relative to alternatives.

4.5 Future Improvements and Deployment

Several directions could extend this work. Most immediately, completing the hyperparameter experiments will reveal whether LoRA rank significantly affects performance on this dataset size and task complexity. If the differences prove meaningful, a more comprehensive hyperparameter search across rank, learning rate, and training epochs would be warranted.

For production deployment, the model would need additional infrastructure beyond the fine-tuning demonstrated here. A confidence-based routing system could automatically assign high-confidence predictions (perhaps ¿0.8 probability) while sending uncertain cases to human reviewers. This human-in-the-loop approach would improve accuracy beyond the 64% achieved by full automation. Setting up active learning to periodically retrain on expert-corrected predictions would create a positive feedback loop where the model continuously improves from mistakes.

The model would also benefit from improved label quality for training. Rather than keyword heuristics, investing in expert annotation for even a subset of examples (perhaps 300-500 carefully labeled cases) might significantly improve performance, especially for severity classification. Alternatively, using constrained generation with LLM-generated labels (forcing outputs to match the test set vocabulary) could provide the semantic quality of V2 without the vocabulary mismatch problem.

Finally, extending the multi-region classification to include additional fields (such as priority, affected version, or root cause category) would increase the system's utility for actual bug triage workflows. Each additional classification dimension reduces the manual work required from developers and provides more actionable information for project management.

5 Conclusion

This project demonstrates that parameter-efficient fine-tuning provides a practical path to specialized LLM applications under real-world resource constraints. Using QLoRA to fine-tune Mistral-7B-Instruct on 1,399 labeled bug reports, we achieved 64% classification accuracy compared to 0.33% zero-shot baseline, a 190-fold improvement accomplished with approximately 4.5 hours of training on free Kaggle GPUs.

The experimental results reveal important insights about specialized classification tasks. Different classification fields showed dramatically different accuracy levels (reproducibility at 88% versus severity at 42%), suggesting that task difficulty varies based on whether the problem requires pattern matching or contextual reasoning. This finding has practical implications for allocating resources in labeling and model development.

We also documented a valuable negative result where semantically improved labels with inconsistent vocabulary caused severe performance degradation. This experience highlights that label format consistency matters as much as label semantic quality, a lesson that may prevent similar issues for others working with LLM fine-tuning on specialized tasks.

While working under constraints of limited compute (Kaggle's 30-hour weekly GPU quota), limited data (GitHub API rate limits), and limited time (assignment deadline), we demonstrated that achieving useful results with PEFT does not require massive infrastructure or datasets. This accessibility makes custom LLM deployment viable for organizations and researchers who previously could not afford the resources required for full model fine-tuning or large-scale data collection.

References

[1] Anvik, J., Hiew, L., & Murphy, G. C. (2006). Who should fix this bug? In *Proceedings of ICSE '06*.

[2] Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs. In *NeurIPS 2023*.

[3] Fan, A., Ordoñez, V., & Yang, J. (2021). Automated bug report labeling with pre-trained transformers. In *Proceedings of ASE 2021*.

[4] Hu, E. J., Shen, Y., Wallis, P., et al. (2021). LoRA: Low-rank adaptation of large language models. In *Proceedings of ICLR 2022*.

[5] Lamkanfi, A., Demeyer, S., Giger, E., & Goethals, B. (2010). Predicting the severity of a reported bug. In *7th IEEE Working Conference on MSR 2010*.

[6] Tian, Y., Wijedasa, D., Lo, D., & Le Goues, C. (2020). Deep learning for automated bug severity assessment. *IEEE Trans. Software Engineering*.

[7] Zheng, R., Dou, S., Gao, S., et al. (2024). Self-Instruct: Aligning LLMs with self-generated instructions. *ArXiv preprint*.