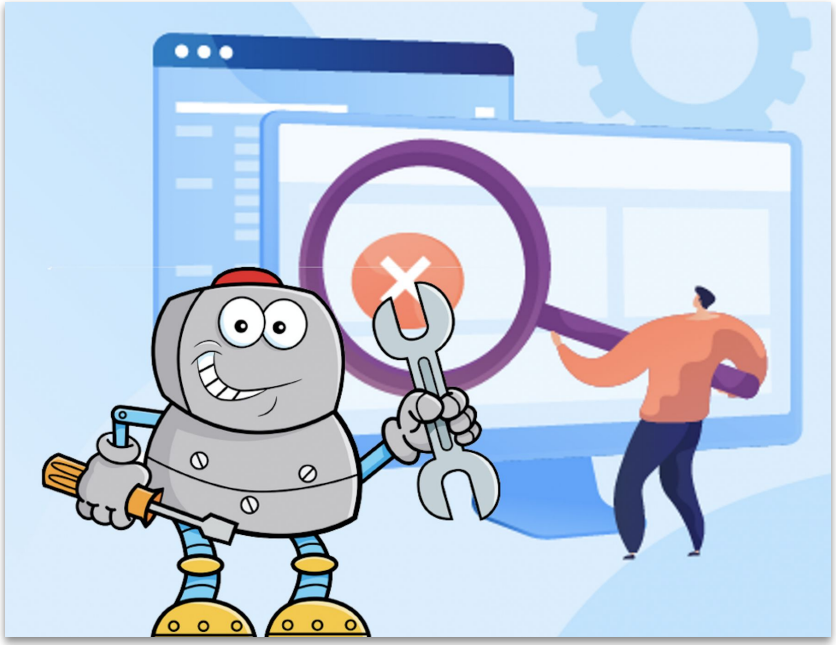
Automated Vulnerability Repair for ML Libraries



Team Saga de Codigos

J. G. H. Shehan Premathilaka (200482F)

J. G. H. Sheshan Premathilaka (200483J)

S. A. Indunil Umayanga (200671J)

Supervisors

Dr. Sandareka Wickramanayake

Dr. Nisansa de Silva

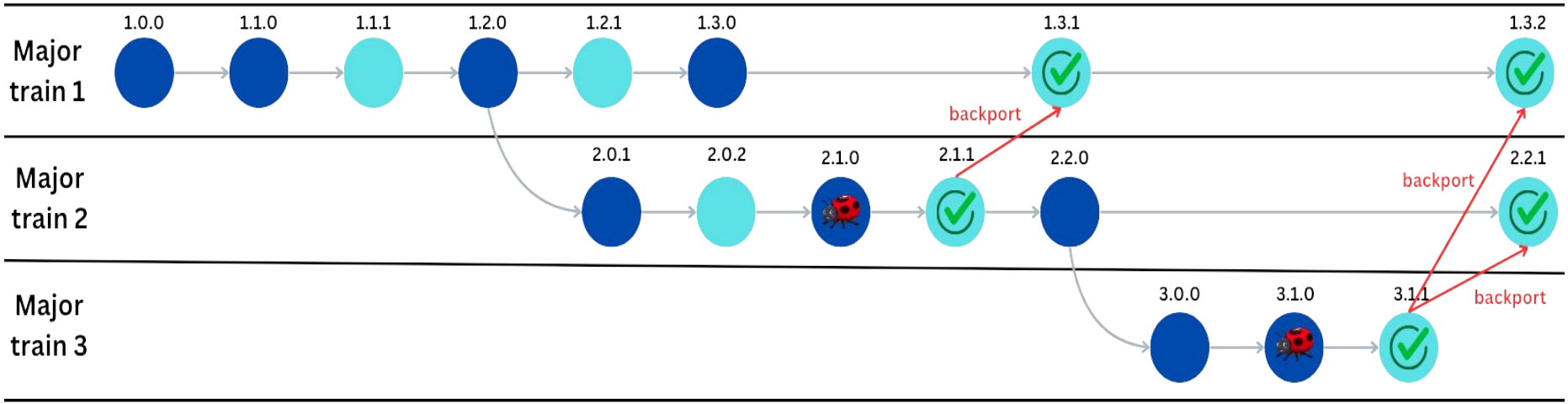
Dr. Ridwan Shariffdeen

Table of contents

Introduction

Background

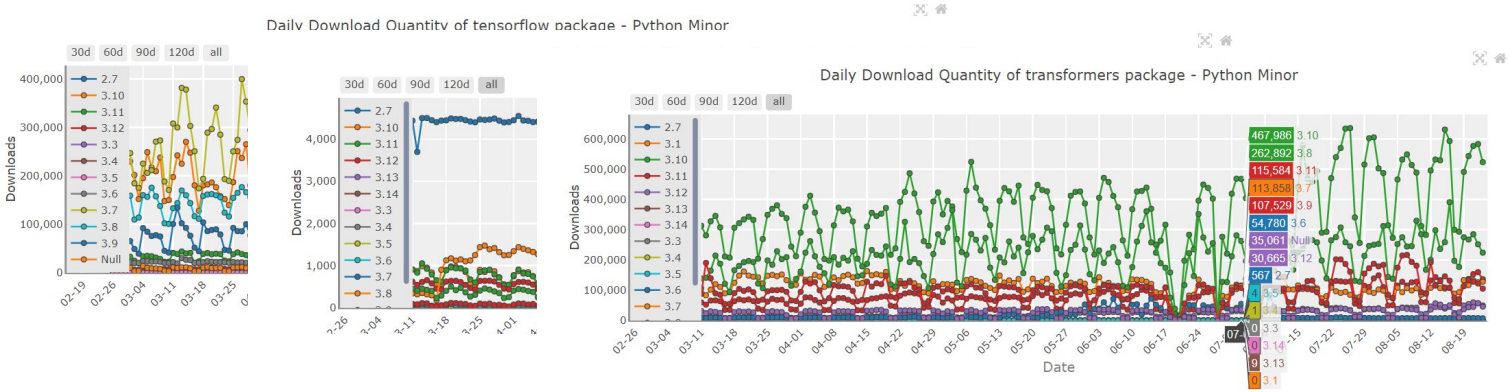
- Backporting ?



- Security backports:
  - Security backports protect older software versions with critical vulnerability fixes.

Motivation

- Why backport is needed ?
  - According to the PyPI Stats

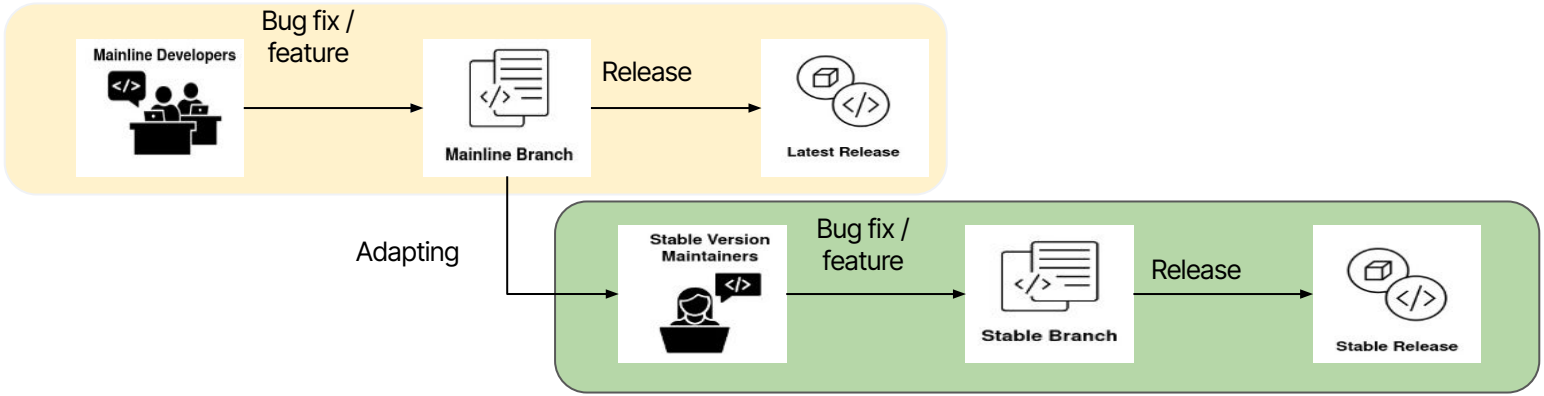


- Why machine learning libraries ?
  - Security vulnerabilities in ML libraries risk data breaches and model compromise.
  - No much research specially on ML libraries

F. Tambon, A. Nikanjam, L. An, F. Khomh, and G. Antoniol, “Silent bugs in deep learning frameworks: an empirical study of keras and tensorflow,” Empirical Software Engineering, vol. 29, no. 1, p. 10, 2024.

Motivation (cont.)

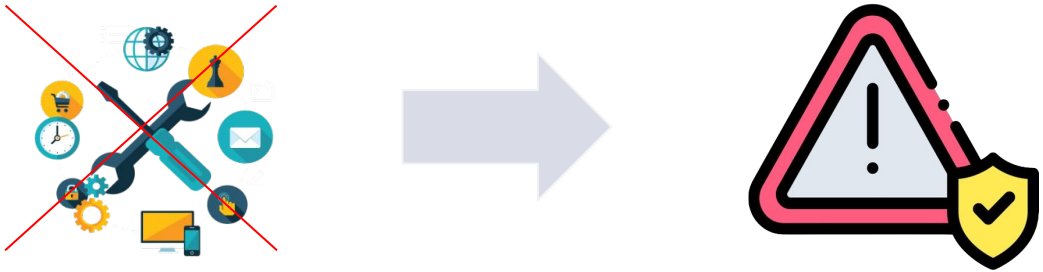
- Why automating the backporting process?
  - The delay is considerable.
    - According to the study Chakroborti et al [1], 16 days to create and 5 days to merge.
  - Automating backporting ensures consistent and efficient application of updates across software versions.



D. Chakroborti, K. A. Schneider, and C. K. Roy, "Backports: Change types, challenges and strategies," in Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, 2022, pp. 636–647.

Problem Statement

Developing an automated security vulnerability patch backporting system for machine learning systems and applications



Research Objectives

Main Objectives

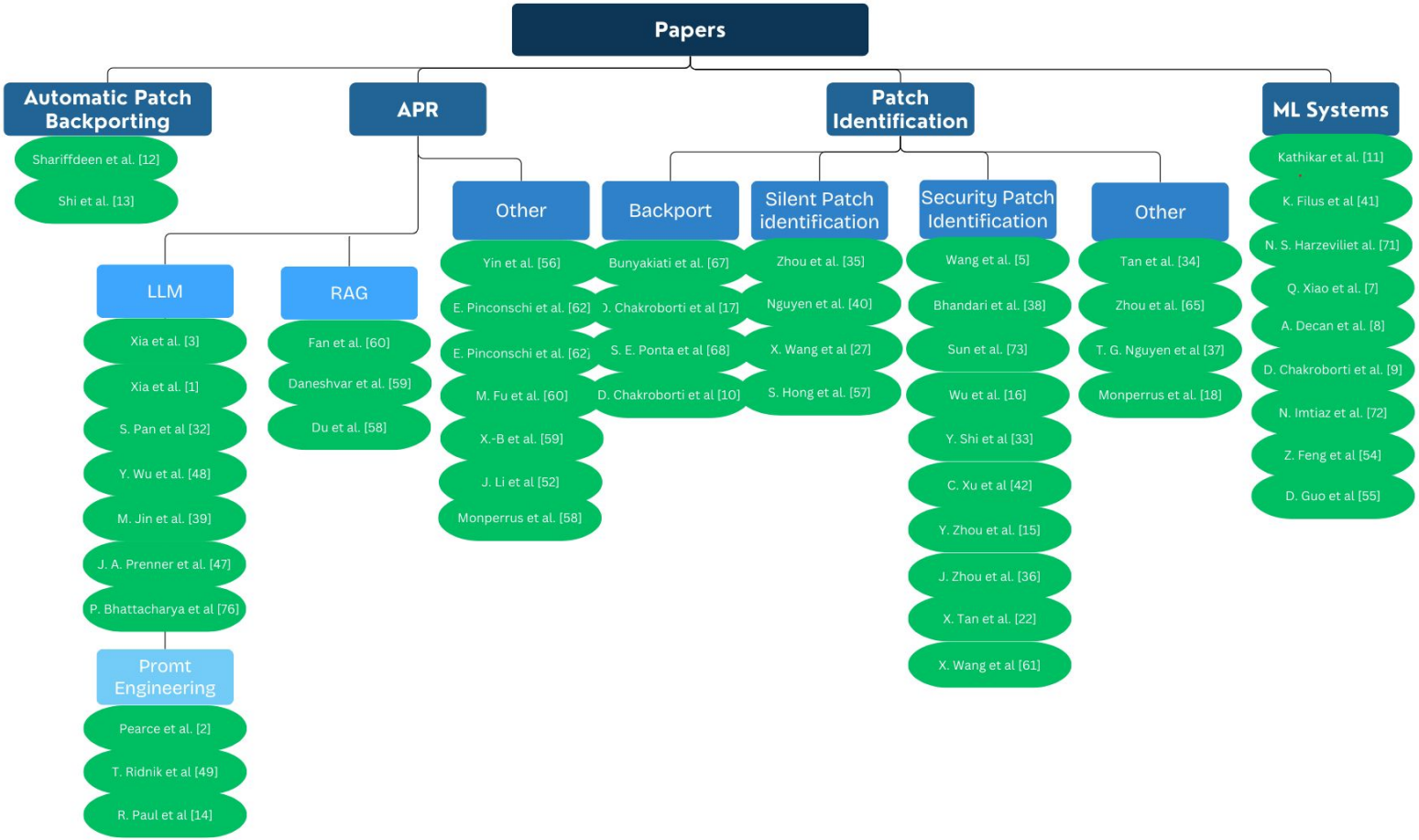
- Create a security backport dataset for ML systems that act as the benchmark for evaluating the backporting tools.

- Develop a LLM-based security patch backporting tool for ML systems that can accurately backport security patches to stable versions of the system.

- Evaluate the tool using the created benchmark dataset and compare it with some of the state of art tools.

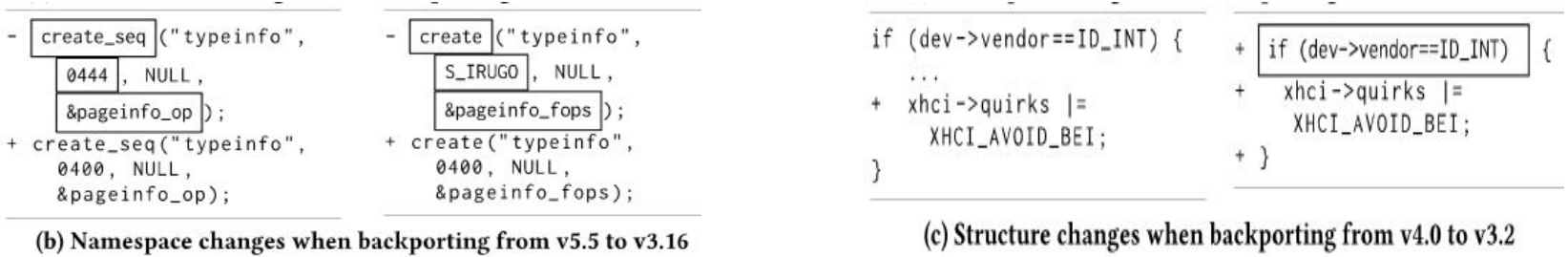
Literature Review

Overview


Papers
Automatic Patch Backporting
Shariffdeen et al. [12]
Shi et al. [13]
APR
LLM
Xia et al. [3]
Xia et al. [1]
S. Pan et al [32]
Y. Wu et al. [48]
M. Jin et al. [39]
J. A. Prenner et al. [47]
P. Bhattacharya et al [76]
Promt Engineering
Pearce et al. [2]
T. Ridnik et al [49]
R. Paul et al [14]
RAG
Fan et al. [60]
Daneshvar et al. [59]
Du et al. [58]
Other
Yin et al. [56]
E. Pinconschi et al. [62]
E. Pinconschi et al. [62]
M. Fu et al. [60]
X.-B et al. [59]
J. Li et al [52]
Monperrus et al. [58]
Patch Identification
Backport
Bunyakiati et al. [67]
D. Chakroborti et al [17]
S. E. Ponta et al [68]
D. Chakroborti et al [10]
Silent Patch identification
Zhou et al. [35]
Nguyen et al. [40]
X. Wang et al [27]
S. Hong et al. [57]
Security Patch Identification
Wang et al. [5]
Bhandari et al. [38]
Sun et al. [73]
Wu et al. [16]
Y. Shi et al [33]
C. Xu et al [42]
Y. Zhou et al. [15]
J. Zhou et al. [36]
X. Tan et al. [22]
X. Wang et al [61]
Other
Tan et al. [34]
Zhou et al. [65]
T. G. Nguyen et al [37]
Monperrus et al. [18]
ML Systems
Kathikar et al. [11]
K. Filus et al [41]
N. S. Harzeviliet al. [71]
Q. Xiao et al. [7]
A. Decan et al. [8]
D. Chakroborti et al. [9]
N. Imtiaz et al. [72]
Z. Feng et al [54]
D. Guo et al [55]

Automated Patch Backporting in Linux (Experience Paper)

- Propose and evaluate an automated tool, FixMorph, for backporting patches from the mainline version of the Linux kernel to older stable versions.
- Synthesizing a partial transformation rule from a mainline Linux patch, generalizing it based on alignment between mainline and target versions, and applying it to produce a backported patch.
- Key results
  - FixMorph, successfully backports 75.1% from 350 patches, outperforming existing techniques in both precision and recall.



(b) Namespace changes when backporting from v5.5 to v3.16

(c) Structure changes when backporting from v4.0 to v3.2

Different types of changes in backports compare to original patch

R. Shariffdeen, X. Gao, G. J. Duck, S. H. Tan, J. Lawall, and A. Roychoudhury, "Automated patch backporting in linux (experience paper)," in Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2021, pp. 633–645.

Backporting Security Patches of Web Applications: A Prototype Design and Implementation on Injection Vulnerability Patches

- Implement a framework (SKYPORT) to automate the backporting of security patches for injection vulnerabilities in legacy web applications, ensuring both security and backward compatibility.
- Propose 2 concepts:
  - Safely-Backportable Patches (SBP)
  - Safely-Back Portable Versions (SBV)
- Key results
  - SKYPORT successfully verified and backported 98 out of 155 security patches to 750 old versions of web application frameworks

Y. Shi, Y. Zhang, T. Luo, X. Mao, Y. Cao, Z. Wang, Y. Zhao, Z. Huang, and M. Yang, “Backporting security patches of web applications: A prototype design and implementation on injection vulnerability patches,” in 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 1993–2010.

How Effective Are Neural Networks for Fixing Security Vulnerabilities

- Evaluate effectiveness of 5 LLMs, 4 fine-tuned LLMs and 4 DL based APR tools in fixing security vulnerabilities using a Java security vulnerability benchmark dataset.
- Key results
  - LLMs demonstrate the most potential in fixing security vulnerabilities.
  - Overall performance on security vulnerabilities is limited. Codex fixed most - 20.4%.
  - Most of the generated patches are uncompilable.
- Conclusions
  - There is a need for enhanced syntax handling and deeper domain knowledge within these models

Y.Wu, N. Jiang, H. V. Pham, T. Lutellier, J. Davis, L. Tan, P. Babkin, and S. Shah, “How effective are neural networks for fixing security vulnerabilities,” in Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, 2023, pp. 1282–1294

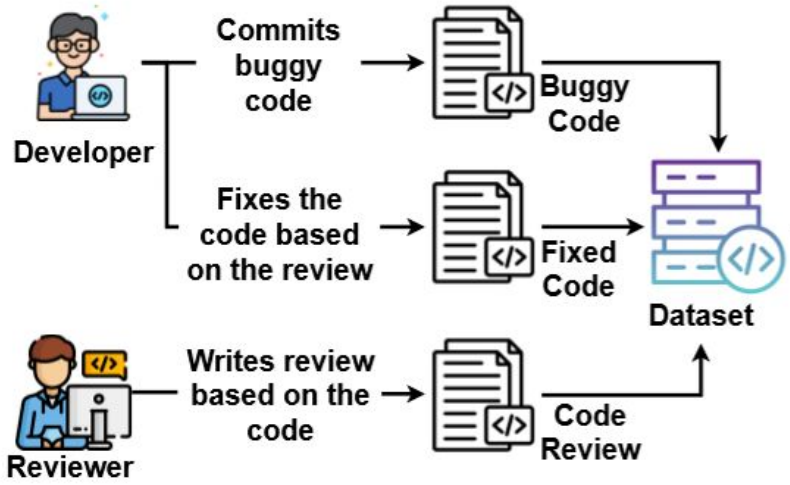
Automatic Static Bug Detection for Machine Learning Libraries: Are We There Yet?

- Evaluate 5 popular static bug detectors on 4 populer ML libraries.
- Results
  - Static bug detectors are not effective in detecting bugs in ML libraries. Collectively detect (6/410).
  - Flawfinder and RAT are the most effective. Collectively detect 4.
- Conclusions
  - Tools need more domain specific knowledge to detect ML bugs.

Tool	TPR	FNR
Flawfinder	0.04	0.95
RATS	0.03	0.97
Cppcheck	0	1
Infer	0	1
Clang static analyzer	0	1

E. Pinconschi, R. Abreu, and P. Adão, “A comparative study of automatic program repair techniques for security vulnerabilities,” in 2021 IEEE 32nd international symposium on software reliability engineering (ISSRE). IEEE, 2021, pp. 196–207.

Enhancing Automated Program Repair through Fine-tuning and Prompt Engineering



- Two datasets have been used, containing buggy codes, corresponding fixes, and code reviews.
- Results
  - Adding code reviews enhance the accuracy of code generation using zero-shot learning significantly
- Conclusions
  - Providing code explanation as a input is a good way to enhance the code generation.

- According to P. Bhattacharya et al [7] code LLM(CodeBERT) can explain the code than the Normal LLM models (Llama).

R. Paul, M. M. Hossain, M. L. Siddiq, M. Hasan, A. Iqbal, and J. Santos, "Enhancing automated program repair through fine-tuning and prompt engineering," arXiv preprint arXiv:2304.07840, 2023.
P. Bhattacharya, M. Chakraborty, K. N. Palepu, V. Pandey, I. Dindorkar, R. Rajpurohit, and R. Gupta, "Exploring large language models for code explanation," arXiv preprint arXiv:2310.16673, 2023.

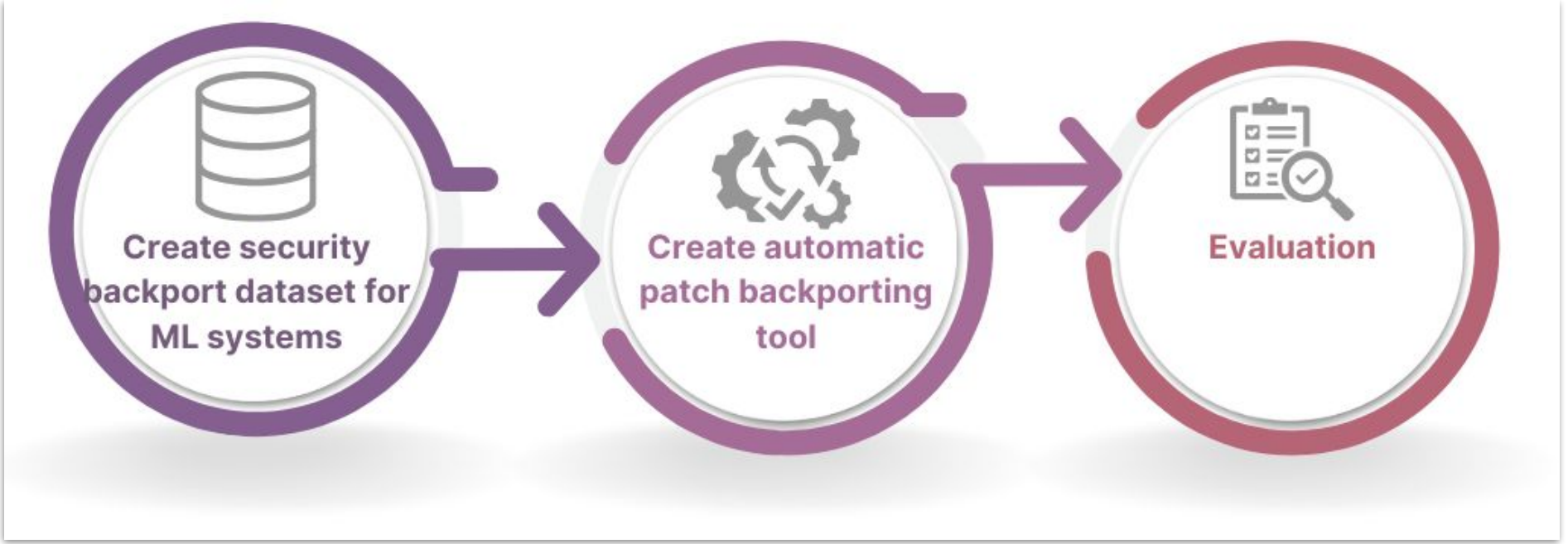
Summary

Study	Description	Importants
Shariffdeen et al.	FixMorph tool for automate patch backports in Linux kernel	Five different types of backport patches. Based on Transformation Rule Synthesis.
Shi et al.	SKYPORT tool to automate patch backporting for web application	Have used static analysis and symbolic tracking with SBP and SBV
Y. Wu et al.	Evaluate the effectiveness of LLM and APR tool in fixing security vulnerabilities.	LLM are good for fixing security vulnerabilities. But need deeper domain knowledge.
E. Pinconschi et al.	Evaluate the effectiveness of APR tools in fixing security vulnerabilities.	Fixing security vulnerabilities is challenging and need deeper domain knowledge.
N. S. Harzevili et al.	Evaluate static bug detectors on ML libraries.	Bugs in ML libraries are bit different, and tools need more domain knowledge.
R. Paul et al.	Creating a dataset with code reviews are added by experts	By adding code reviews, the accuracy of the generated code is enhanced.
P. Bhattacharya	Examine the code explanation using normal LLM and the code LLM	Can use code LLM for code explanation and automate the code explanation process.

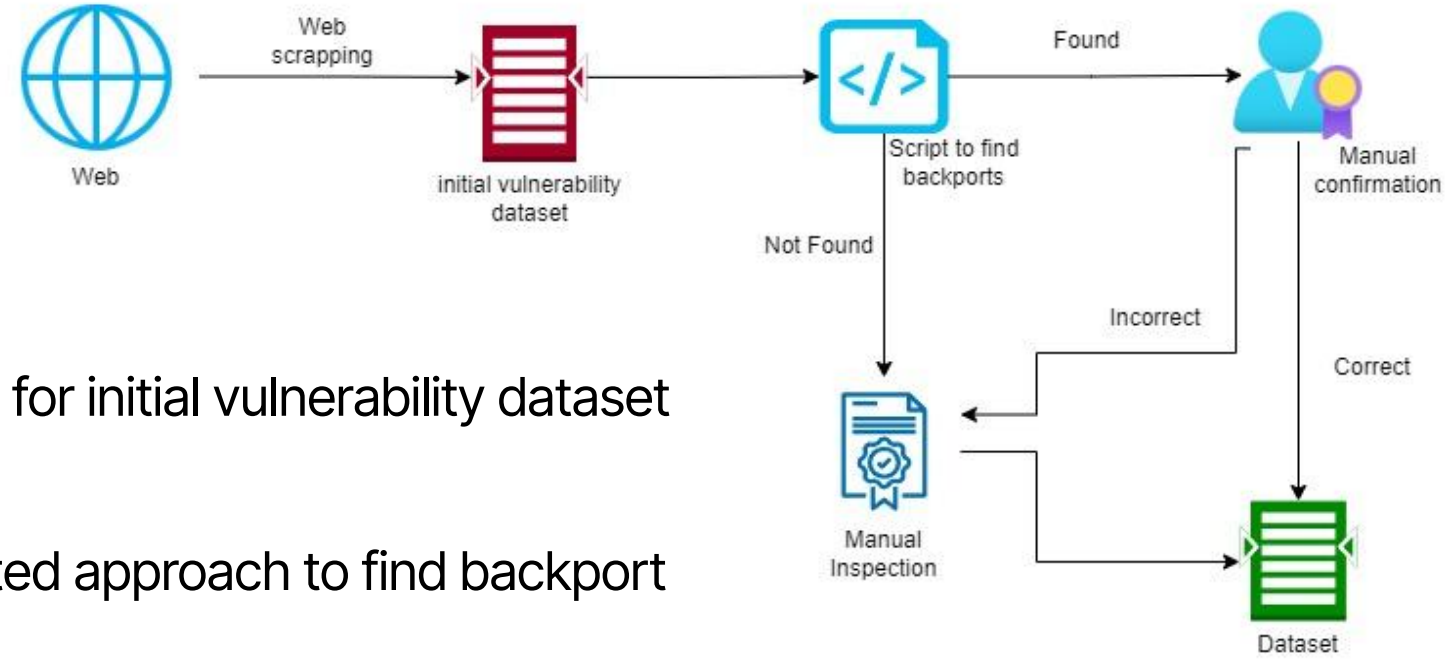
Methodology

Overview

Three main tasks

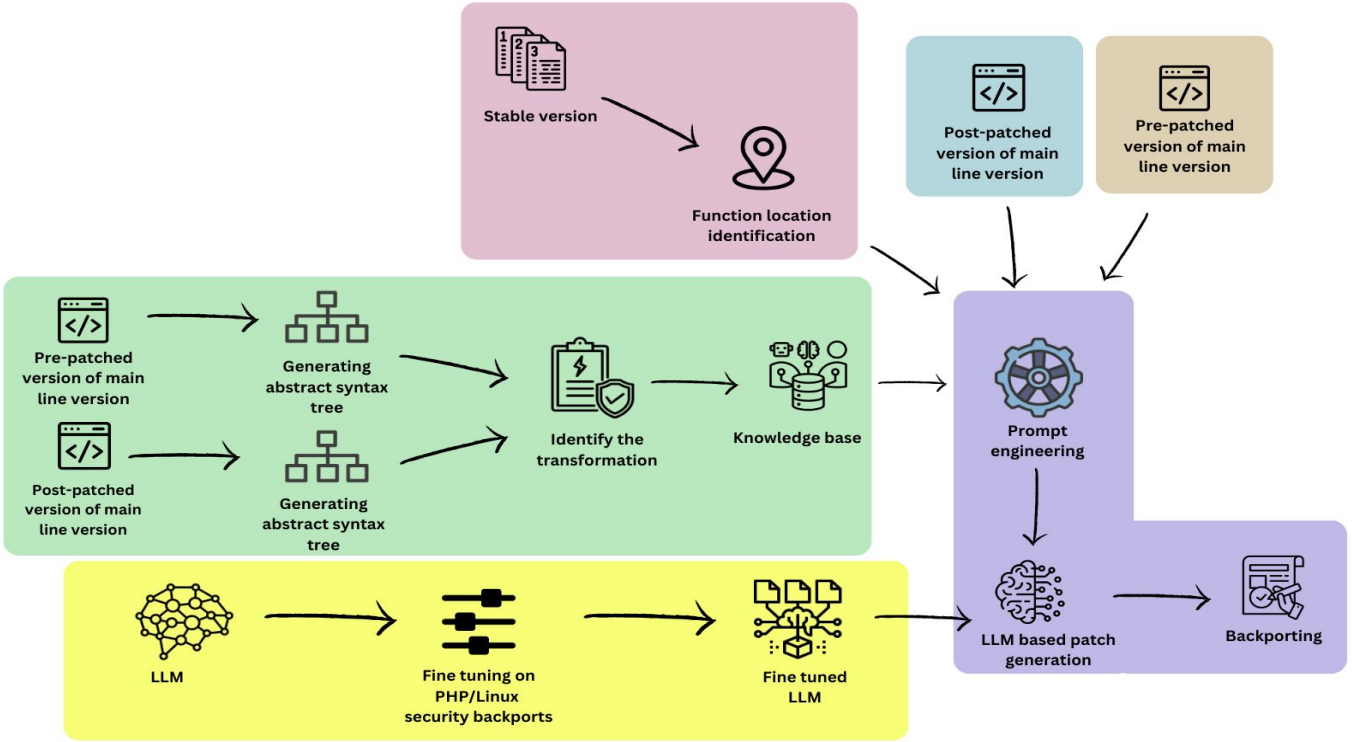


Creating the security backport dataset for ML system



- Web scraping for initial vulnerability dataset creation
- Semi automated approach to find backport
  - Script
  - Manual confirmation
  - Manual inspection

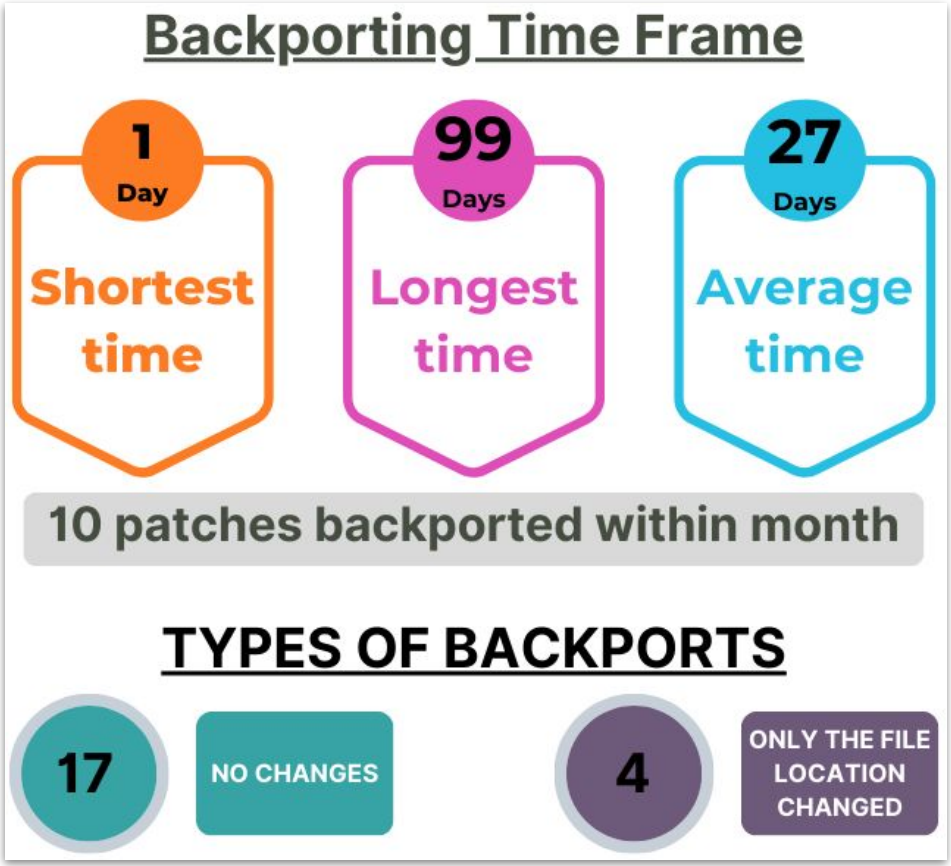
Creating automatic patch backporting tool



- RAG model based solution
- Fine tuned LLM based code generation
  - Prompt engineering

Experimental study –> Dataset

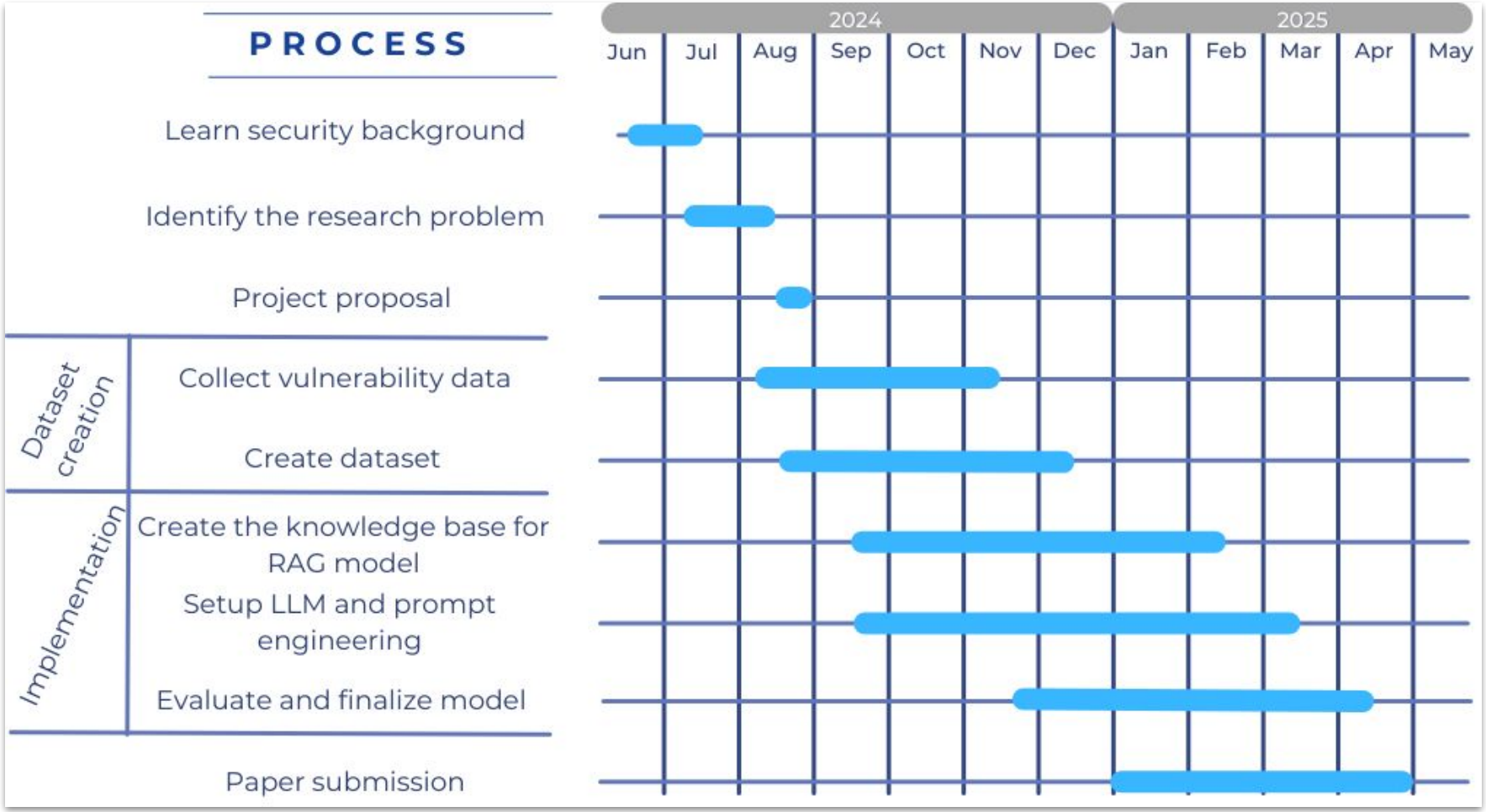
So far collected **21** security backports relate to **Tensorflow** library



Experimental study –> Evaluation matrix

	Description	Formula
Syntactic Correctness Rate (SCR)	Proportion of generated patches that are syntactically correct(do not introduce syntax errors)	$SCR = \frac{N_{syntacticallyCorrect}}{N_{totalGenerated}} \times 100\%$
Functionally Correctness Rate (FCR)	Proportion of generated patches that are functionally correct(do not introduce functional or logical errors)	$FCR = \frac{N_{functionallyCorrect}}{N_{totalGenerated}} \times 100\%$
Overall Correct Backport Rate (OCBR)	Proportion of all generated patches that are correctly backported.	$OCBR = \frac{N_{correctBackports}}{N_{totalGenerated}} \times 100\%$

Proposed timeline



Conclusion

Important ?

- ML security backport dataset
  - Can be used as a benchmark dataset to evaluate future backporting tools and APR tools
- LLM based automatic backporting tool.
  - Will reduce the manual effort and improves the ML relate software security.
  - Paves path to future AI driven security enhancement.

Limitations

- Only focus about the ML security. Can not be guaranteed about the accuracy of general use.

References

1. C. S. Xia and L. Zhang, “Less training, more repairing please: revisiting automated program repair via zero-shot learning,” in Proceedings of the 30th ACM. Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022, pp. 959–971.
2. H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, “Examining zeroshot vulnerability repair with large language models,” in 2023 IEEE Symposium on Security and Privacy (SP). IEEE, 2023, pp. 2339–2356.
3. C. S. Xia, Y. Wei, and L. Zhang, “Automated program repair in the era of large pre-trained language models,” in 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023, pp. 1482–1494.
4. X. Wang, S. Wang, P. Feng, K. Sun, and S. Jajodia, “Patchdb: A large-scale security patch dataset,” in 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2021, pp. 149–160.
5. Q. Xiao, K. Li, D. Zhang, and W. Xu, “Security risks in deep learning implementations,” in 2018 IEEE Security and privacy workshops (SPW). IEEE, 2018, pp. 123–128.
6. A. Decan, T. Mens, A. Zerouali, and C. De Roover, “Back to the past–analysing backporting practices in package dependency networks,” IEEE Transactions on Software Engineering, vol. 48, no. 10, pp. 4087–4099, 2021
7. D. Chakroborti, K. A. Schneider, and C. K. Roy, “Reback: Recommending back-ports in social coding environments,” Automated Software Engineering, vol. 31, no. 1, p. 18, 2024
8. D. Chakroborti, C. Roy, and K. Schneider, “A study of backporting code in open-source software for characterizing changesets,” in Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, 2024, pp. 296–297.
9. A. Kathikar, A. Nair, B. Lazarine, A. Sachdeva, and S. Samtani, “Assessing the vulnerabilities of the open-source artificial intelligence (ai) landscape: A large-scale analysis of the hugging face platform,” in 2023 IEEE International Conference on Intelligence and Security Informatics (ISI). IEEE, 2023, pp. 1–6.
10. R. Shariffdeen, X. Gao, G. J. Duck, S. H. Tan, J. Lawall, and A. Roychoudhury, “Automated patch backporting in linux (experience paper),” in Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2021, pp. 633–645.

References

11. Y. Shi, Y. Zhang, T. Luo, X. Mao, Y. Cao, Z. Wang, Y. Zhao, Z. Huang, and M. Yang, “Backporting security patches of web applications: A prototype design and implementation on injection vulnerability patches,” in 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 1993–2010.
12. R. Paul, M. M. Hossain, M. L. Siddiq, M. Hasan, A. Iqbal, and J. Santos, “Enhancing automated program repair through fine-tuning and prompt engineering,” arXiv preprint arXiv:2304.07840, 2023.
13. Y. Zhou, J. K. Siow, C. Wang, S. Liu, and Y. Liu, “Spi: Automated identification of security patches via commits,” ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 31, no. 1, pp. 1–27, 2021.
14. B. Wu, S. Liu, R. Feng, X. Xie, J. Siow, and S.-W. Lin, “Enhancing security patch identification by capturing structures in commits,” IEEE Transactions on Dependable and Secure Computing, 2022.
15. D. Chakroborti, K. A. Schneider, and C. K. Roy, “Backports: Change types, challenges and strategies,” in Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, 2022, pp. 636–647
16. X. Tan, Y. Zhang, C. Mi, J. Cao, K. Sun, Y. Lin, and M. Yang, “Locating the security patches for disclosed oss vulnerabilities with vulnerability-commit correlation ranking,” in Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, 2021, pp. 3282–3299.
17. X. Wang, K. Sun, A. Batcheller, and S. Jajodia, “Detecting" 0–day" vulnerability: An empirical study of secret security patch in oss,” in 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2019, pp. 485–492.
18. S. Pan, Y. Wang, Z. Liu, X. Hu, X. Xia, and S. Li, “Automating zero-shot patch porting for hard forks,” arXiv preprint arXiv:2404.17964, 2024.
19. Y. Shi, Y. Zhang, T. Luo, X. Mao, and M. Yang, “Precise (un) affected version analysis for web vulnerabilities,” in Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022, pp. 1–13.
20. X. Tan, Y. Zhang, J. Cao, K. Sun, M. Zhang, and M. Yang, “Understanding the practice of security patch management across multiple branches in oss projects,” in Proceedings of the ACM Web Conference 2022, 2022, pp. 767–777.

References

21. J. Zhou, M. Pacheco, J. Chen, X. Hu, X. Xia, D. Lo, and A. E. Hassan, “Colefunda: Explainable silent vulnerability fix identification,” in 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023, pp. 2565–2577.
22. J. Zhou, M. Pacheco, Z. Wan, X. Xia, D. Lo, Y. Wang, and A. E. Hassan, “Finding a needle in a haystack: Automated mining of silent vulnerability fixes,” in 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2021, pp. 705–716.
23. T. G. Nguyen, T. Le-Cong, H. J. Kang, X.-B. D. Le, and D. Lo, “Vulcurator:a vulnerability-fixing commit detector,” in Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022, pp. 1726–1730.
24. G. Bhandari, A. Naseer, and L. Moonen, “Cvefixes: automated collection of vulnerabilities and their fixes from open-source software,” in Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering, 2021, pp. 30–39.
25. M. Jin, S. Shahriar, M. Tufano, X. Shi, S. Lu, N. Sundaresan, and A. Svyatkovskiy, “Inferfix: End-to-end program repair with llms,” in Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2023, pp. 1646–1656.
26. S. Nguyen, T. T. Vu, and H. D. Vo, “Vffinder: A graph-based approach for automated silent vulnerability-fix identification,” in 2023 15th International Conference on Knowledge and Systems Engineering (KSE). IEEE, 2023, pp. 1–6.
27. K. Filus and J. Doma´nska, “Software vulnerabilities in tensorflow-based deep learning applications,” Computers & Security, vol. 124, p. 102948, 2023.
28. C. Xu, B. Chen, C. Lu, K. Huang, X. Peng, and Y. Liu, “Tracking patches for open source software vulnerabilities,” in Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022, pp. 860–871

References

29. T. Lutellier, H. V. Pham, L. Pang, Y. Li, M. Wei, and L. Tan, “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, 2020, pp. 101–114.
30. D. Drain, C. Wu, A. Svyatkovskiy, and N. Sundaresan, “Generating bug-fixes using pretrained transformers,” in Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming, 2021, pp. 1–8.
31. J. A. Prenner, H. Babii, and R. Robbes, “Can openai’s codex fix bugs? an evaluation on quixbugs,” in Proceedings of the Third International Workshop on Automated Program Repair, 2022, pp. 69–75.
32. Y. Wu, N. Jiang, H. V. Pham, T. Lutellier, J. Davis, L. Tan, P. Babkin, and S. Shah, “How effective are neural networks for fixing security vulnerabilities,” in Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, 2023, pp. 1282–1294.
33. T. Ridnik, D. Kredo, and I. Friedman, “Code generation with alphacodium: From prompt engineering to flow engineering,” arXiv preprint arXiv:2401.08500, 2024.
34. J. Li, G. Li, Y. Li, and Z. Jin, “Structured chain-of-thought prompting for code generation,” arXiv preprint arXiv:2305.06599, 2023.
35. Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang et al., “Codebert: A pre-trained model for programming and natural languages,” arXiv preprint arXiv:2002.08155, 2020.
36. D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li et al., “Deepseek-coder: When the large language model meets programming–the rise of code intelligence,” arXiv preprint arXiv:2401.14196, 2024.
37. X. Yin, C. Ni, S. Wang, Z. Li, L. Zeng, and X. Yang, “Thinkrepair: Self-directed automated program repair,” arXiv preprint arXiv:2407.20898, 2024.
38. S. Hong, H. Sun, X. Gao, and S. H. Tan, “Investigating and detecting silent bugs in pytorch programs,” in 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2024, pp. 272–283.
39. M. Monperrus, “Automatic software repair: A bibliography,” ACM Computing Surveys (CSUR), vol. 51, no. 1, pp. 1–24, 2018.

References

40. X.-B. D. Le, “Towards efficient and effective automatic program repair,” in Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, 2016, pp. 876–879.
41. M. Fu, C. Tantithamthavorn, T. Le, V. Nguyen, and D. Phung, “Vulrepair: a t5based automated software vulnerability repair,” in Proceedings of the 30th ACM joint european software engineering conference and symposium on the foundations of software engineering, 2022, pp. 935–947.
42. X. Wang, S. Wang, P. Feng, K. Sun, S. Jajodia, S. Benchaaboun, and F. Geck, “Patchrnn: A deep learning-based system for security patch identification,” in MILCOM 2021–2021 IEEE Military Communications Conference (MILCOM). IEEE, 2021, pp. 595–600.
43. E. Pinconschi, R. Abreu, and P. Adão, “A comparative study of automatic program repair techniques for security vulnerabilities,” in 2021 IEEE 32nd international symposium on software reliability engineering (ISSRE). IEEE, 2021, pp. 196–207
44. Y. Zhou and A. Sharma, “Automated identification of security issues from commit messages and bug reports,” in Proceedings of the 2017 11th joint meeting on foundations of software engineering, 2017, pp. 914–919.
45. F. Tambon, A. Nikanjam, L. An, F. Khomh, and G. Antoniol, “Silent bugs in deep learning frameworks: an empirical study of keras and tensorflow,” Empirical Software Engineering, vol. 29, no. 1, p. 10, 2024.
46. P. Bunyakiati and C. Phipathananunth, “Cherry-picking of code commits in long-running, multi-release software,” in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 994–998.
47. S. E. Ponta, H. Plate, A. Sabetta, M. Bezzi, and C. Dangremont, “A manually-curated dataset of fixes to vulnerabilities of open-source software,” in 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). IEEE, 2019, pp. 383–387.
48. P. M. Bittner, A. Schultheiß, S. Greiner, B. Moosherr, S. Krieter, C. Tinnes, T. Kehrer, and T. Thüm, “Views on edits to variational software,” in Proceedings of the 27th ACM International Systems and Software Product Line Conference Volume A, 2023, pp. 141–152.

References

49. N. S. Harzevili, J. Shin, J. Wang, S. Wang, and N. Nagappan, “Automatic static vulnerability detection for machine learning libraries: Are we there yet?” in 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2023, pp. 795–806.
50. N. Imtiaz, A. Khanom, and L. Williams, “Open or sneaky? fast or slow? light or heavy?: Investigating security releases of open source packages,” IEEE Transactions on Software Engineering, vol. 49, no. 4, pp. 1540–1560, 2022.
51. S. Sun, S. Wang, X. Wang, Y. Xing, E. Zhang, and K. Sun, “Exploring security commits in python,” in 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2023, pp. 171–181.
52. P. Bhattacharya, M. Chakraborty, K. N. Palepu, V. Pandey, I. Dindorkar, R. Rajpurohit, and R. Gupta, “Exploring large language models for code explanation,” arXiv preprint arXiv:2310.16673, 2023.

Thank You

Any Question ?