A Thesis Proposal On
Comparative Analysis of RoBERTa and CodeBERT for Automated Misconfiguration Detection in Dockerfiles

Deepak Aryal
7925005

Overview

- Introduction
- Problem Statement
- Objectives
- Literature Review
- Methodology
- Expected Outcomes
- Work Schedule
- References

Introduction

- Software growth demands scalable infrastructure, and DevOps tools like Docker streamline testing, deployment, and delivery.
- DevOps artifacts like Dockerfiles are complex, under-studied, and often overlooked in automated analysis since they are not treated as traditional code.
- Even with popular tools like the VS Code Docker extension, Dockerfiles often contain rule violations, leading to misconfigurations and security risks.
- This proposed research evaluates NLP models (RoBERTa, CodeBERT) for detecting Dockerfile misconfigurations, comparing performance and adaptability to improve security and reliability in DevOps.

Problem Statement

- Many engineers overlook Docker best practices, misconfigurations & vulnerabilities.
- Existing tools (Hadolint, Trivy, Dockle, Chekov) are rule-based, limited to predefined checks, and cumbersome to integrate into the CI process.
- Many publicly available Docker images contain security vulnerabilities, often caused by improperly configured Docker artifacts.
- NLP/AI for Dockerfiles is still underexplored.

Objectives

The main objectives of the Thesis are:

- To design and develop automated detection framework using either roBERTA or CodeBERT, and find which one has better performance.
- To evaluate and compare the performance of RoBERTa and CodeBERT for automated misconfiguration detection in Dockerfiles, using metrics such as accuracy, precision, recall etc.

Literature Review

- The study ‘Learning from, Understanding, and Supporting DevOps Artifacts for Docker’ shows non-expert Dockerfiles violate best practices far more than expert files and proposes the Binnacle tool for rule enforcement, highlighting the need for automated, adaptive approaches[6].
- The paper ‘Shipwright: A Human-in-the-Loop System for Dockerfile Repair’ uses a modified BERT model to suggest repairs for broken Dockerfiles, improving build correctness, but focuses on build failures rather than line-level security misconfigurations[2].
- The paper ‘Dockerfile Flakiness: Characterization and Repair’ presents FlakiDock, which uses LLMs and dynamic analysis to repair flaky Dockerfiles, showing AI’s potential for automated correction, though it does not address line-level security misconfigurations[3].

Literature Review (Continue)

- The article ‘Dockerfile Linting: Every time, Everywhere’ highlights integrating linting tools like Hadolint into Docker workflows to enforce best practices and maintain consistent quality[7].
- The paper ‘Not all Dockerfile Smells are the Same’ analyzes expert handling of Hadolint-detected Dockerfile smells, showing that even experienced developers overlook issues and highlighting the importance of linting tools, though it does not explore AI/NLP-based detection[8].
- The paper ‘DRIVE: Dockerfile Rule Mining and Violation Detection’ mines implicit rules from Dockerfiles to detect violations, improving quality, but focuses on static analysis rather than AI/NLP-based detection[9].

Methodology

Dataset description:

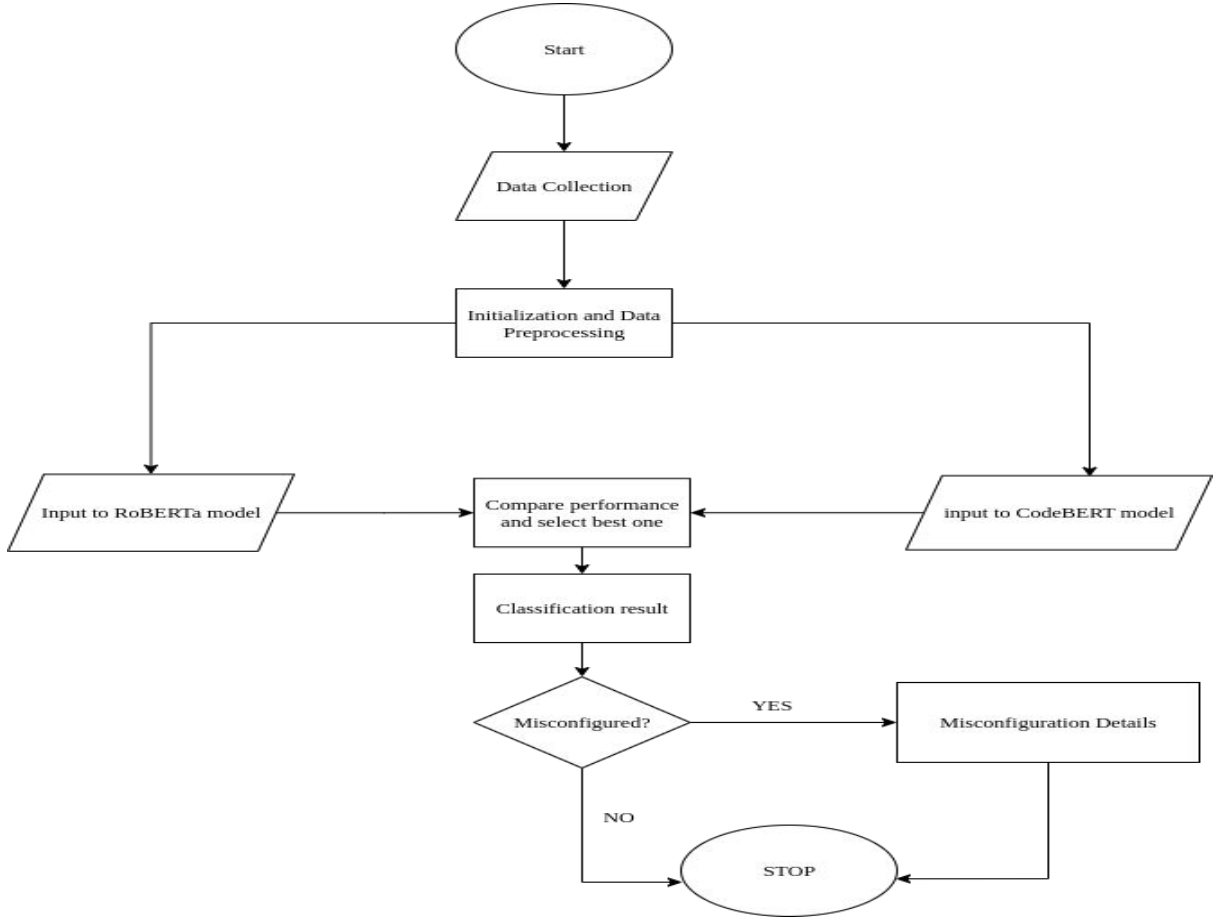
- Primary Source: Henkel et al.'s Dockerfile corpus (~178,000 Dockerfiles from GitHub, ICSE 2020) captures historical real-world practices; supplemented with Dockerfiles from 2021–2025 via GitHub Search API to reflect current practices.
- Preprocessing & Labels: Dockerfiles parsed line by line; each instruction (e.g., FROM, RUN, COPY) labeled as correct or misconfigured for training and evaluation of NLP models.

Data preparation:

- Training Set: 70% of the data
- Test Set: 30% of the data

Flowchart

System flowchart



Evaluation Metrics

Accuracy:

Accuracy measures the overall correctness of the model's prediction, defined as the ratio of correctly identified instances to the total number of instances:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

where TP = true positives, TN = true negatives, FP = false positives and FN = false negatives.

Precision:

Precision measures the percentage of positive predictions that are actually positive, also known as the positive predictive value.

$$\text{Precision} = TP / (TP + FP)$$

Evaluation Metrics (Continue)

Recall:

Recall measures how well a model identifies true positives and it is given by:

$$\text{Recall} = TP / TP + FN$$

High recall ensures that most misconfigurations are correctly detected.

F1 score:

F1 score measures weighted average of precision and recall, and is calculated using the harmonic mean of these two matrices.

$$\text{F1 score} = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$

Expected Outcome

- Determine which NLP model (RoBERTa or CodeBERT) performs better in detecting Dockerfile misconfigurations.
- Implement the best-performing model to classify configurations as correct/incorrect and explain the type of misconfiguration

Time Schedule

The project started on 3rd August 2025 and is targeted to be completed within 12 weeks, by the end of October 2025.

S. N	Activity/Month	1^{st}-2^{nd} week	3^{rd}- 4^{th} Week	5^{th}-6^{th} week	7^{th}-8^{th} week	9^{th}-10^{th} week	11^{th}-12^{th} week
1	Preliminary Study						
2	Literature Review						
3	Data Collection						
4	Model Testing and Analysis						
5	Documentation						

Figure: Gantt Chart

References

[1] S. Phillips, T. Zimmermann, and C. Bird, “Understanding and Improving Software Build Teams,” Proc. - Int. Conf. Softw. Eng., May 2014, doi:10.1145/2568225.2568274.

[2] J. Henkel, D. Silva, L. Teixeira, M. d’Amorim, and T. Reps, “Shipwright: A Human-in-the-Loop System for Dockerfile Repair,” 2021 IEEE/ACM 43rd Int. Conf. Softw. Eng., pp. 1148–1160, 2021, [Online]. Available: https://api.semanticscholar.org/CorpusID:232045443

[3] T. Shabani, N. Nashid, P. Alian, and A. Mesbah, “Dockerfile Flakiness: Characterization and Repair,” 2025 IEEE/ACM 47th Int. Conf. Softw. Eng., pp. 1793–1805, 2024, [Online]. Available: https://api.semanticscholar.org/CorpusID:271855816

References (Continue)

[4] J. Henkel, C. Bird, S. K. Lahiri, and T. Reps, “ICSE 2020 Artifact for: Learning from, Understanding, and Supporting DevOps Artifacts for Docker.” Zenodo, 2020. doi:10.5281/zenodo.3628771.

[5] A. Vaswani et al., “Attention is All you Need,” in Neural Information Processing Systems, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:13756489

[6] J. Henkel, C. Bird, S. K. Lahiri, and T. Reps, “Learning from, Understanding, and Supporting DevOps Artifacts for Docker,” 2020 IEEE/ACM 42nd Int. Conf. Softw. Eng., pp. 38–49, 2020, [Online]. Available: https://api.semanticscholar.org/CorpusID:211069127

References (Continue)

[7] D. W. Elliott, “Dockerfile Linting: Every time, Everywhere,” 2020, [Online]. Available: https://medium.com/@david.w.elliott/dockerfile-linting-every-time-everywhere-d8d271a1e650

[8] G. Rosa, S. Scalabrino, G. Robles, and R. Oliveto, Not all Dockerfile Smells are the Same: An Empirical Evaluation of Hadolint Writing Practices by Experts. 2024. doi:10.1145/3643991.3644905.

[9] Y. Zhou, W. Zhan, Z. Li, T. Han, T. Chen, and H. C. Gall, “DRIVE: Dockerfile Rule Mining and Violation Detection,” ACM Trans. Softw. Eng. Methodol., vol. 33, pp. 1–23, 2022, [Online]. Available: https://api.semanticscholar.org/CorpusID:254564701

Thank you!!!