Industrial Validation of a Neural Network Model Using the Novel MixTCP Tool

Arnold Szederjesi-Dragomir[a], Radu Găceanu[b] and Andreea Vescan[c]
Computer Science Department, Faculty of Mathematics and Computer Science,
Babeş-Bolyai University, Cluj-Napoca, România

Keywords: Test Case Prioritization, Continuous Integration, Neural Network, Elixir.

Abstract: Test Case Prioritization (TCP) is crucial in the fast-paced world of software development to speed up and optimize testing procedures, particularly in Continuous Integration (CI) setups. This paper aims to first validate a state-of-the-art neural network model to TCP in CI environments, by applying it into a real-world industrial context, and second to propose *MixTCP*, a tool that integrates the neural network model and significantly enhances the regression testing experience from the software developer perspective. *MixTCP* is implemented in the Elixir programming language and employs the NEUTRON model, a state-of-the-art approach that uses neural networks to intelligently prioritize test cases, effectively improving fault detection and reducing testing time. The tool is composed of loosely coupled components (Mix TCP task, TCP Server, and NEUTRON model), thus enabling the integration of other Test Case Prioritization solutions too. The results show that MixTCP has the potential to be a valuable asset to modern software development methods, offering software engineers a more efficient, a more user-friendly, and an overall easier to integrate TCP approach.

1 INTRODUCTION

The efficiency and effectiveness of testing processes (Ammann and Offutt, 2016) are highly important in the continually evolving landscape of software development. The adoption of continuous integration (CI) and deployment processes has highlighted the crucial importance of optimizing Test Case Prioritization (TCP). For maintaining software quality it is essential to use TCP which determines the order of test case execution to maximize early fault detection. However, traditional TCP approaches, which are frequently based on heuristics, are becoming impractical when faced with the complexity and dynamic nature of current software projects, especially in Continuous Integration (CI) settings (Spieker et al., 2017; Elbaum et al., 2014; Lima and Vergilio, 2022).

Therefore, the aim of the paper is twofold:

1. to validate a neural network-based model for TCP in CI by applying it in real-world industrial settings, and
2. to integrate the neural network-based model in a test case prioritization tool in order to facilitate its use by software developers.

In order to address the first aim of this paper, we employ the NEUTRON (Vescan et al., 2023) model, which is a state-of-the-art neural network-based method to prioritize test cases. Experiments on benchmark datasets have shown its effectiveness across various budget constraints, obtaining competitive or superior results compared to other state-of-the-art approaches. Specifically designed for CI settings, NEUTRON effectively reorders test cases to detect faults earlier, thereby optimizing the testing process. With the aim of validating NEUTRON, we applied it in a private industrial project containing 59 test files with 297 test cases throughout the test files.

The second aim of this paper is to introduce the *MixTCP* tool that embeds the NEUTRON model (Vescan et al., 2023), including its functionality, practical use, and evaluation results. We show how MixTCP seamlessly interacts with existing development workflows while providing a user-friendly interface for TCP management. The architecture of *MixTCP* is composed of three main components: the Mix TCP task, the TCP Server, and the NEUTRON model. The Mix TCP task serves as the user interface, allowing developers to easily interact with the

[a] https://orcid.org/0000-0002-1106-526X
[b] https://orcid.org/0000-0002-0977-4104
[c] https://orcid.org/0000-0002-9049-5726

Szederjesi-Dragomir, A., Găceanu, R. and Vescan, A.
Industrial Validation of a Neural Network Model Using the Novel MixTCP Tool.
DOI: 10.5220/0012631100003687

In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2024)*, pages 110-119
ISBN: 978-989-758-696-5; ISSN: 2184-4895

tool using a CLI (Command Line Interface). The TCP Server is the core of the system, responsible for processing test data and orchestrating the overall workflow of the tool. It acts as a bridge between the user interface and NEUTRON. Furthermore, the paper highlights the potential of the tool to significantly improve the TCP process, ultimately contributing to more efficient and successful CI workflows.

MixTCP represents an important step forward in software engineering by bridging the gap between research and real-world software testing needs. It provides a more efficient and effective approach to TCP, particularly in CI environments, and contributes to the ongoing integration of artificial intelligence into software development practices.

The remainder of this paper is structured as follows. Section 2 motivates the need for TCP investigations, presenting the theory and definition of TCP, and finalizing with the other theoretical elements that will be used in our investigation. Section 3 highlights the current existing approaches in TCP. Section 4 describes the design and implementation of the proposed tool along with the tool architecture, while Section 5 presents experiments and results. The threats to validity are presented in Section 6 and Section 7 concludes our paper and presents possibilities for future research directions.

2 THEORETICAL BACKGROUND

This section presents relevant theoretical and technical background elements, starting with the motivation behind this research and continuing with the TCP perspectives, TCP related work, and concepts on Elixir and Mix.

2.1 Motivation

In the current rapid software development market, the ability to produce high-quality software quickly and consistently is not only an advantage, but it is also a requirement. Continuous Integration (CI) procedures have become an essential component of modern software development, allowing teams to integrate code changes in a timely and efficient manner. However, this high speed brings with it its own set of obstacles, particularly in the field of software testing. Test Case Prioritization (TCP) becomes a necessity in this context, but it raises its own challenges. With the increasing complexity of software systems and the frequency of code changes, classic TCP approaches are no longer appropriate. These methods, which frequently rely on heuristic or manual procedures, are time-consuming, error-prone, and incapable of scaling to meet the growing size of testing needs. This inefficiency can result in delayed detection of important flaws, higher testing expenses, and, ultimately, slower deployment of the project in production.

2.2 TCP and NAPFD

According to (Graves et al., 1998), the Test Case Prioritization problem can be defined as in Definition 1.

Definition 1. ***Test Case Prioritization (Graves et al., 1998):*** *a test suite, T, the set of permutations of T, PT; a function from PT to real numbers, f. The goal is to find $T \in PT$ such that:*

$$(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]. \quad (1)$$

In order to evaluate the effectiveness of a TCP approach, several metrics are proposed. For example, APFD (Average Percentage of Faults Detected) (Pradeepa and VimalDevi, 2013) measures the effectiveness of a test suite in detecting faults as early as possible during the testing process and is defined as follows:

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \times m} + \frac{1}{2n}. \quad (2)$$

where: TF_i is the position of the first test case that detects the i-th fault, m is the total number of faults detected by the test suite, and n is the total number of test cases. This metric is especially useful for comparing different prioritization algorithms, where the goal is to reorganize the test cases so that those most likely to find flaws run first.

An extension of APFD to incorporate the fact that not all test cases are executed and failures can be undetected is the Normalized APFD (NAPFD) (Qu et al., 2007):

$$NAPFD = p - \frac{\sum_{i=1}^{m} TF_i}{n \times m} + \frac{p}{2 \times n}. \quad (3)$$

where p is the number of faults detected by the prioritized test suite divided by the number of faults detected throughout the test suite. Variations of the aforementioned metrics include, for example, the incorporation of weights associated with the faults based on their severity.

2.3 Elixir and Mix

Elixir (Elixir, nd) is a functional programming language, known for its concurrency and fault tolerance capabilities. It runs on the Erlang VM, thus giving developers complete access to Erlang's ecosystem (Erlang, nd). Besides all the features that make

Elixir stand out from other programming languages, we have also chosen it to implement the MixTCP tool because the real work project on which we validate the NEUTRON approach is also written in Elixir.

Mix (Mix, nd) is a build automation tool for working with applications written in the Elixir. Besides many other operations, it lets programmers create projects, format, build, test or run the code, and finally to package and deploy. As Mix is widely used for all kinds of tasks in a project (many times even the single tool used), we ought to implement our tool as a Mix archive. These archives can usually be installed directly from Github to a central repository (typically */.mix/archives*), thus any project can make use of them without needing any additional setup or dependency management. One thing to note is that *mix test* operates on files, so we choose to implement the test case prioritization process per test file and not per test case.

3 RELATED WORK

Extensive research has been conducted in the field of test case selection and prioritization, with numerous approaches that address various optimization objectives and employ a wide range of techniques (Pan et al., 2022; Bertolino et al., 2020; Khalid and Qamar, 2019; Kandil et al., 2017; Almaghairbe and Roper, 2017; Medhat et al., 2020). Depending on the machine learning methods applied, studies on test case selection and prioritization can be categorized into four main groups: supervised learning, unsupervised learning, reinforcement learning, and natural language processing (Pan R., 2022).

Recent studies have applied reinforcement learning (RL) to Test Case Prioritization (TCP) in Continuous Integration (CI) due to the adaptability of RL to the dynamic nature of CI without the need for full retraining. Once trained, the RL agent can evaluate a test case, assign it a score, and use that score to order or prioritize the test cases. While most RL studies focus solely on the execution history of training, one study, (Bertolino et al., 2020), incorporates code complexity metrics. In this comprehensive paper, the authors compare 10 machine learning algorithms, emphasizing the differences between supervised learning and RL for TCP. They conducted experiments on six public datasets, providing guidelines for applying machine learning in CI regression testing. In (Bertolino et al., 2020), the authors also introduce new metrics (Rank Percentile Average (RPA) and Normalized-Rank-Percentile-Average (NRPA)) to evaluate how close a prediction ranking is to the optimal one. However, (Pan R., 2022) shows that NRPA may not always be suitable. Nonetheless, (Bertolino et al., 2020) is a thorough and reproducible study and is recognized by (Pan R., 2022) as a significant contribution.

Clustering in the context of Test Case Prioritization (TCP) operates on the assumption that test cases with similar attributes, such as coverage, exhibit similar fault detection capabilities. Numerous studies, including (Khalid and Qamar, 2019), employ the K-means algorithm or its variations for this purpose. Although the Euclidean distance is the commonly used similarity measure in clustering, some explore alternatives such as the Hamming distance, as seen in (Kandil et al., 2017). An interesting approach can be found in (Almaghairbe and Roper, 2017), where clustering is used for anomaly detection of passing and failing executions.

Supervised learning is a widely used ML technique for addressing TCP as a ranking problem, typically employing one of three ranking models: pointwise, pairwise, or listwise. In (Bertolino et al., 2020), the authors evaluated various models such as Random Forest (RF), Multiple Additive Regression Tree (MART), L-MART, RankBoost, RankNet, and Coordinate ASCENT (CA) for TCP using a state-of-the-art ranking library (Dang and Zarozinski, 2020). MART emerged as the most accurate model. However, a key drawback of supervised learning is the requirement of a complete dataset before training.

The use of NLP in TCP is relatively limited, but it aims to extract and use information from textual software development artifacts (e.g., bug descriptions) or treat source code as textual data. In general, this involves converting test cases into vectors and calculating the distances between them, followed by different prioritization strategies. An interesting approach, as presented in (Medhat et al., 2020), utilizes NLP to preprocess the specification of the components of the system. Recurrent neural networks are then used to classify specifications into components such as user devices, protocols, gateways, sensors, actuators, and data processing. On the basis of this classification, test cases related to these components are selected, and search-based techniques, such as genetic algorithms and simulated annealing, are used for prioritization.

In the following, some studies are presented that specifically target the CI context.

In the approach from (Elbaum et al., 2014), the authors use the sliding time window to choose the test suits to be applied in a pre-submit phase of testing by tracking their history. In a subsequent post-submit phase, a similar approach is employed to prioritize tests. Experiments with the Google Shared

Dataset of Test Suite Results (GSDTSR) indicate that the testing load is reduced and delays in fault detection are reduced. In (Spieker et al., 2017), an innovative method is introduced to prioritize and select test cases in Continuous Integration (CI) environments. This approach uses reinforcement learning to prioritize test cases based on their duration, previous execution, and failure history. The study evaluates the proposed method using industrial datasets and compares it with deterministic test case prioritization methods. The results indicate that this approach can effectively learn to prioritize test cases in 60 cycles, even without prior information on the test cases. This suggests its promise for CI test case prioritization. In (Lima and Vergilio, 2022), a Multi-Armed Bandit (MAB) approach is introduced for the prioritization of test cases in CI environments. MAB problems are a simplified form of Reinforcement Learning (RL), as they do not require context information, do not change environment states, and do not involve state spaces or function approximators. Extensive experiments were conducted, showing that, with certain parameter configurations, the MAB approach outperformed the method proposed in (Spieker et al., 2017). In (Omri and Sinz, 2022), an approach based on the Dueling Bandit Gradient Descent (DBGD) algorithm is introduced. Evaluation of several industrial datasets indicates that after 150 cycles, the proposed method outperforms other state-of-the-art approaches.

Most approaches from related work in CI use online models for prioritization and advocate that this choice is an advantage for the continuous integration context because the training phase is eliminated. Nevertheless, to obtain comparable results with deterministic methods, rather significant volumes of data are still needed (e.g. at least 60 cycles, that is, about two months of data, for one of the state-of-the-art methods (Spieker et al., 2017)). The approach of (Omri and Sinz, 2022), which is to our knowledge the best in the literature, needs 150 cycles (that is, about 5 months of data) to achieve this result. Considering the vast volumes of data (and time span) needed by these methods to become efficient, we argue that eliminating the training phase is not necessarily a great advantage as it may seem. Our approach, which uses a neural network, needs, of course, a training phase, but our experiments show that it outperforms related approaches in some of the studied use cases. In our experiments, we investigate several scenarios, some of which involve transformations of the original datasets, enriching them with additional features, to improve the performance of our model.

The NEUTRON approach from (Vescan et al., 2023) uses a neural network to prioritize test cases in continuous integration environments. Experiments are carried out on benchmark datasets (Spieker et al., 2017) obtaining better results in terms of NAPFD for some budgets compared to the following state-of-the-art methods: Elbaum (Elbaum et al., 2014), RETECTS (Spieker et al., 2017), COLEMAN (Lima and Vergilio, 2022), LearnTec (Omri and Sinz, 2022). This is why we have chosen our NEUTRON approach for validation in a real-world industrial context.

TCP Tools. Despite decades of research on test case prioritization techniques, there is a noticeable absence of practical and user-friendly tools in this domain. The MOTCP tool by (Islam et al., 2012) uses a multi-objective test prioritization technique that uses information related to the code and requirements coverage, along with test case cost execution. Two case studies were performed with MOTCP using 4 Java programs with various numbers of requirements and available test cases. The tool proposed by (Sampath et al., 2011) allows testers to prioritize and reduce user-session-based test cases. The investigation using tertiary studies by (Singhal et al., 2021) provided a comprehensive list of tools used in the TCP area, some used for analysis purposes, some providing code coverage information ((EMMA, nd), (Cobertura, nd), (JaCoCo, nd)), and some to specify the mutation adequacy score ((MutGen, nd), (muJava, nd)). The tool developed by (Cleanscape, nd) also provides the *Regress* tool from the toolkit for effective regression testing to identify a representative subset of tests to revalidate modified software. The tool proposed by (ATAC, nd) investigates how thoroughly a program is tested by a set of tests using data flow coverage techniques, identifies areas that are not well tested and identifies overlap among tests. The authors advocate for the development of practical tool support in the near future to demonstrate the real-world value of test-case prioritization.

4 PROPOSED MixTCP TOOL

An overview of the MixTCP tool is provided below, along with the architecture and prerequisites for running the tool with example commands.

4.1 Overview

This section explains the main functionalities of the MixTCP tool. The main components of the tool are: Mix TCP task, TCP Server, and NEUTRON model, as provided in Figure 1.

Communication between the task and the server is done using RPC (Remote Procedure Call) via Erlang

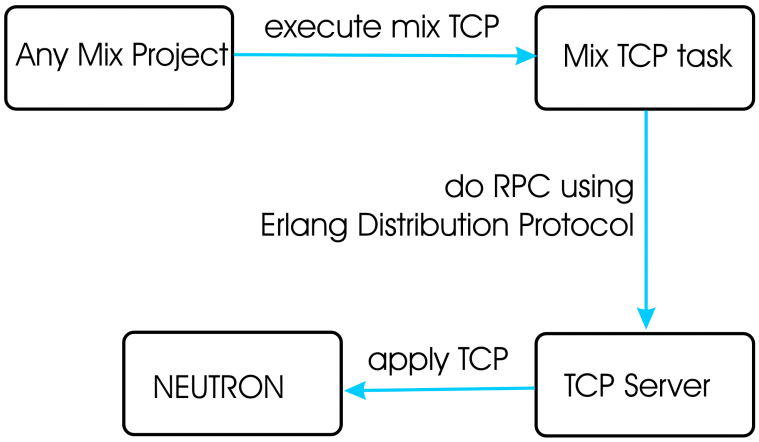

Figure 1: Overview of the approach.

Distribution Protocol (EPD, nd), while the communication between the server and NEUTRON is just a function call as we have integrated our neural network model in the server itself.

In what follows, we outline the architecture of the MixTCP tool, provide the prerequisites needed to run it and offer some examples on how to use it.

4.2 Architecture

As observed in Figure 1 our tool consists of 3 components: Mix TCP task, TCP Server and NEUTRON.

Firstly, the Mix task, which is installed as a Mix archive globally for multiple Mix projects, is the CLI (Command Line Interface) tool that the user uses to interact with MixTCP. The tool requires that the designated execution folder be a Mix project. Without this, it will be unable to execute the selected tests and will signal this. The console tool has four parameters:

1. number of test files to run (-n): this will specify how many test files to run
2. server (-s): address of the TCP Server to communicate with
3. cookie (-c): cookie used to be able to connect with the server (this always needs to be the same both on the client and the server)
4. test_runs_folder (-f): folder where previous test runs are stored (this is used by NEUTRON to efficiently prioritize the tests for the current run).

Communication between the task and the server occurs via Remote Procedure Calls (RPCs), using the Erlang Distribution Protocol (EPD, nd). Basically, every time an Elixir/Erlang system is started, the Erlang Port Mapper Daemon (EPMD) is also started. This OS-level process/service keeps track of the ports on which Erlang nodes are running. These nodes use it to locate each other. Hence, when the task seeks to establish a connection with the server, it approaches the EPMD, requests the port number of the server, and subsequently attempts to connect to the server. Connections are initiated through a TCP/IP handshake process during which cookies and versions are compared. If all criteria are successfully met, the connection is then established, and exchanging of messages, remote functions calls, or even spawning of new processes remotely will be possible. For our use case, we will call a function, passing the test runs folder, which will then carry out the test case prioritization on behalf of the Mix task.

Secondly, the TCP server, as mentioned above, exposes a function that opens and processes the folder, parses the cycles, and builds the data needed for the NEUTRON neural network. The server requires direct access to the folder, as it will navigate to the filesystem, enumerate all files within that folder, and ultimately read and parse them. Each file will contain the logs for multiple test files and multiple test cases per test file. During their processing the test cases are aggregated, and only test files are considered as an input for NEUTRON.

Communication between the server and NEUTRON is straightforward, occurring via a local function call. **The third component refers to the NEUTRON model** that we have proposed in (Vescan et al., 2023). We have trained and built the NEUTRON neural network model in a separate environment and then just integrated it (together with its parameters) into the server system.

As shown in the paper of (Vescan et al., 2023), by training NEUTRON solely on the IOF/ROL dataset, it is still able to produce decent prioritizations on other, considerably different datasets. It uses the following

data organized per test file as input:

- cycles: number of cycles the test file was executed in;
- duration: first we compute the sum of the testcase durations, and then we compute the average over all cycles;
- total runs: number of runs over all cycles (as usually each test file is run only once per cycle this will be equal to cycles, but to have a more general solution we have also included this; also cycles can be defined or built differently);
- fault rate: if one of the testcases in a test file fails, the whole test file fails and then we compute the number of fails throughout all cycles divided by the number of total runs.

4.3 Prerequisites

As prerequisites to run the tool, we need to install Erlang OTP and Elixir, but if one intends to use the tool, it is likely that these are already installed, as we are going to use it in an already existing Mix Project. Steps needed to run:

1. install the Mix task as an archive in our local archives directory, so it can be called using Mix from any existing project without any additional setup;
2. start the TCP server, that will be used by the task to apply TCP on the tests from the given project;
3. run the Mix task with the desired parameters, that will connect to the server to obtain the prioritized tests and then take only the first ones and run them in the current project.

The server needs access to the folder with previous test runs. If there is no data from previous cycles all tests will be executed, regardless of how many we specified to the tool. This cycle will serve as data for the subsequent usages of the tool.

4.4 Commands to Execute the Tool

To start the server the following command can be used in the tool's Git repository (as seen in Figure 2):

```
iex --sname tcp@localhost --cookie 1234 -S mix
```

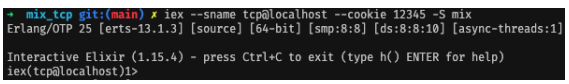

Figure 2: Running the TCP Server.

This will start a terminal with all the code from the project. The server's name or address and the cookie to use (as presented before these will be needed for the communication with the Mix task) are also specified.

After the server is started, the Mix task can be run in any other Mix project with this command (as seen in Figure 3 too):

```
MIX_ENV=test mix tcp\
   -n 10 \
   -s tcp@localhost \
   -c 1234 \
   -f $(pwd)/test_runs\
   > 2023_11_17_10_31
```

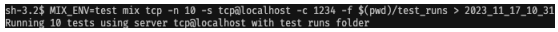

Figure 3: Running the Mix task.

4.5 Advantage of the *MixTCP* Tool

The proposed tool, *MixTCP* is a solution that addresses these difficulties by utilizing the power of artificial intelligence. *MixTCP* augments TCP by employing the NEUTRON model, a state-of-the-art neural network approach, to prioritize test cases. Unlike prior approaches, MixTCP can analyze large volumes of historical test data providing an intelligent and effective prioritization, ensuring that the most critical tests are executed first. By automating and improving the TCP process, *MixTCP* enhances testing accuracy and speed while significantly decreasing manual work. As a result, bugs are identified more quickly, testing resources are used more efficiently, and new products or updates to already existing products are released more quickly.

The contributions and advantages of *MixTCP* are as follows:

- Integration with NEUTRON, a state-of-the-art approach for TCP in CI environments. Machine learning has been explored in previous research and there are indeed some recent and notable contributions to TCP in CI contexts (Spieker et al., 2017; Bertolino et al., 2020; Pan R., 2022; Lima and Vergilio, 2022; Elbaum et al., 2014). Nevertheless, to our knowledge, none of the related works is integrated in a tool like *MixTCP*, making it rather difficult for these approaches to be rapidly adopted by the industry.
- Development in Elixir, which is a language known for its capabilities in handling concurrent processes and distributed systems. This could make *MixTCP* particularly effective in large-scale, complex testing environments.
- Empirical validation in industrial settings. The tool has been rigorously tested and validated in real-world industrial settings.
- User experience and integration. *MixTCP* is user-friendly and easy to integrate in existing projects.

5 EVALUATION

In this section, we first describe our evaluation setup and then present our experiments and results.

5.1 Evaluation Setup

To empirically validate our tool, we employed it in a private industrial project characterized by the following attributes: (also seen in Table 1): 59 test files, 297 test cases throughout the test files, failing rate was not measured, but is moderate throughout development cycles. For this evaluation, we have selected the top N test cases to run with a random N (that is not too big or too small), but in a real-world scenario one would select the top test cases based on the budget/time allocated for running the tests.

Table 1: Dataset properties.

Property Name	Property Value
# of test files	59
# of test cases	297
failing rate	moderate

5.2 Experiments and Results

We have designed an experiment that considers various execution cycles as specified in the following and as see in Figure 4).

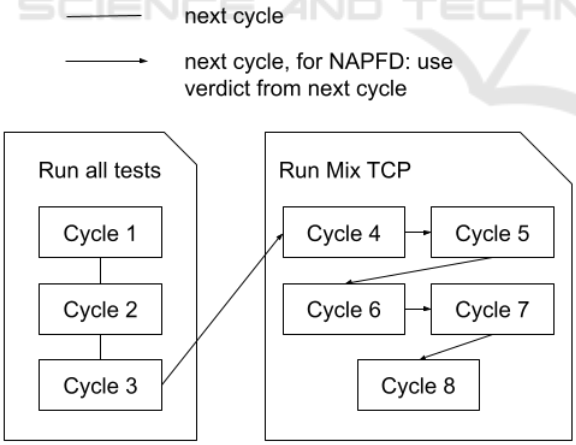

Figure 4: Experiment Cycles.

The experiment design steps are:

- the tests were executed multiple times, throughout 8 cycles;
- each cycle is a single execution of a subset of tests;
- for the first 3 cycles we have executed all tests, without using the tool;
- starting with the fourth cycle, we have started using MixTCP, that prioritized the tests by analyzing data from all previous cycles, thus, we conducted only a select subset of these tests, choosing them based on their assigned priority order;
- in order to compute NAPFD we take the next cycle's test runs to decide each test file's verdict;
- NAPFD is computed for cycles between 3 and 7, the 3rd cycle's NAPFD will tell how well the tests were predicted in the 4th cycle;
- as there are only 8 cycles, no score can be computed for the 8th cycle yet;
- there were 8 faults exposed throughout the experiment.

In order to measure the results, we used the same metric as in the NEUTRON (Vescan et al., 2023) approach, namely NAPFD. This metric provides a sufficient overview of the general performance of a TCP approach. The results are shown in Table 2. As one can observe, the performance of the tool is considerably better than running the tests in a Random way (which is the default behavior in Elixir).

Table 2: Random and NEUTRON NAPFD per cycle.

Cycle	Random NAPFD	NEUTRON NAPFD
3	5.82%	12.18%
4	13.15%	22.25%
5	18.81%	35.06%
6	32.16%	48.20%
7	45.86%	81.46%

Figure 5 graphically illustrates the outcomes for Random and NEUTRON across the cycles, clearly showing a notable enhancement in performance for both.

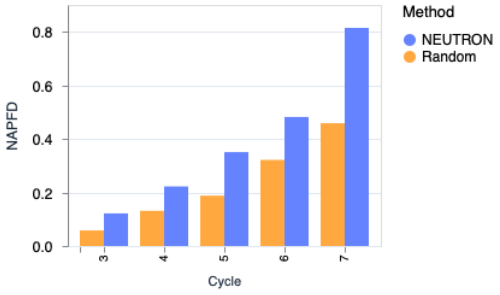

Figure 5: NAPFD per cycle.

This improvement can be attributed to two primary factors:

1. as we run more and more cycles, it becomes more and more likely to find failing tests; this is why both performances grow;

2. NEUTRON has another reason and that is the size of the data; as we are trying to predict tests that fail, having more data makes this prediction more accurate as the tool will choose tests that are more likely to fail.

In conclusion, *MixTCP*, by using NEUTRON, helps faulty tests to be found earlier, its performances increase as there are more and more historical data, and because it prioritizes tests, it is even possible not to run the whole test suite every time. In this way, it also demonstrates the application and integration of NEUTRON in real-world projects.

6 THREATS TO VALIDITY

Experiments may be sensitive to particular threats to validity, and the outcome of this research may be influenced by a variety of factors. Next, several points are indicated that may have influenced the results obtained, stating the action taken to mitigate them.

Internal. One potential threat to validity is the presence of bugs in our implementation. However, through comprehensive testing and code review, we hope to have significantly reduced the likelihood of such issues. Moreover, the tool is validated in real-world industrial settings, thus further minimizing this risk.

Construct. The metrics used to evaluate performance may not fully describe how effective the tool is in real-world scenarios because such measurements do not capture the user friendliness of the tool and its ease of integration in an existing project. However, we hope that the explanations accompanied by some screenshots provide sufficient details in this sense.

External. The initial results of the NEUTRON approach were based on three industrial datasets. Indeed, we should have used more datasets, but to our knowledge, there were no other datasets that have the required data, especially historical information regarding the execution of the test cases. To mitigate this situation, the *MixTCP* tool was applied in real world industrial contexts. Nevertheless, applications of the tool in more projects perhaps from diverse domains would help in ensuring that it generalizes properly.

7 CONCLUSIONS AND FUTURE WORK

Regression testing, a subset of software testing, involves re-running functional and non-functional tests to ensure that the previously developed and tested software still performs after a change. Test Case Prioritization (TCP) becomes particularly challenging in Continuous Integration (CI) contexts because as the number of test cases grows, running the entire test suite for every change becomes time-consuming and resource-intensive. TCP aims to address this by determining the most critical tests to run first to detect faults earlier. However, traditional TCP methods, often based on heuristics, may not be able to effectively adapt to the frequent changes and evolving requirements, leading to inefficiencies and delays in the testing process. This makes the development of more intelligent and adaptive TCP methods essential for modern CI practices. This paper focuses on the integration of the NEUTRON model, a state-of-the-art neural network-based approach for Test Case Prioritization in Continuous Integration environments. The NEUTRON model has been validated using an industrial dataset and incorporated into the *MixTCP* tool for practical use by software developers.

The aim of this paper is twofold: firstly, to validate the NEUTRON model by applying it in real-world industrial settings; and secondly, to integrate this neural network-based model into the *MixTCP* tool, thereby facilitating its practical application by software developers in the field.

Developed using Elixir, a functional programming language well known for its concurrency and fault tolerance, *MixTCP* stands out for its practical usefulness and proven effectiveness in real-world industrial settings. It is composed of three main loose coupled components (Mix TCP task, TCP Server, NEUTRON model), making it a versatile tool for various software development environments.

The evaluation of MixTCP in real-world industrial settings validates its benefits in improving the TCP process. The tool has shown a clear improvement in the prioritization of the test cases, which ultimately leads to earlier fault detection and also more efficient use of testing resources. This has significant implications for the speed and quality of software development, as it enables quicker releases and more reliable software products. The empirical validation of *MixTCP* outlines its practical applicability and effectiveness, confirming its potential as a valuable tool in modern software development practices.

Our future efforts will be directed towards ensuring that *MixTCP* successfully integrates in any development environment. We also plan to integrate other state-of-the-art TCP approaches into our tool, facilitating faster adoption by the software industry. Another idea for future work is to validate the efficacy of the tool with practitioners by conducting an empirical

study in which the results and overall experience with and without using the tool are compared.

ACKNOWLEDGMENT

This work was funded by the Ministry of Research, Innovation, and Digitization, CNCS/CCCDI - UEFISCDI, project number PN-III-P1-1.1-TE2021-0892 within PNCDI III.

REFERENCES

Almaghairbe, R. and Roper, M. (2017). Separating passing and failing test executions by clustering anomalies. *Software Quality Journal*, 25(3):803–840.

Ammann, P. and Offutt, J. (2016). *Introduction to Software Testing*. Cambridge University Press, 2 edition.

ATAC (n.d.). Atac: a test coverage analysis tool. https://invisible-island.net/atac/atac.html. Accessed: 2023-11-19.

Bertolino, A., Guerriero, A., Miranda, B., Pietrantuono, R., and Russo, S. (2020). Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 1–12, New York, NY, USA. Association for Computing Machinery.

Cleanscape (n.d.). Suds software visualization and test toolkit. https://stellar.cleanscape.net/products/testwise/help/introduction.html. Accessed: 2023-11-19.

Cobertura (n.d.). Cobertura: A code coverage utility for java. https://cobertura.github.io/cobertura/. Accessed: 2023-11-19.

Dang, V. and Zarozinski, M. (2020). Ranklib. https://sourceforge.net/p/lemur/wiki/RankLib/. Accessed: 2023-11-19.

Elbaum, S., Rothermel, G., and Penix, J. (2014). Techniques for improving regression testing in continuous integration development environments. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, page 235–245, New York, NY, USA. Association for Computing Machinery.

Elixir (n.d.). Elixir programming language. https://elixir-lang.org. Accessed: 2023-11-19.

EMMA (n.d.). Emma: a free java code coverage tool. http://emma.sourceforge.net/. Accessed: 2023-11-19.

EPD (n.d.). Erlang distribution protocol. https://www.erlang.org/doc/apps/erts/erl_dist_protocol.html. Accessed: 2023-11-19.

Erlang (n.d.). Erlang programming language. https://www.erlang.org/. Accessed: 2023-11-19.

Graves, T. L., Harrold, M. J., Kim, J., Porters, A., and Rothermel, G. (1998). An empirical study of regression test selection techniques. In *Proceedings of the 20th International Conference on Software Engineering*, pages 188–197.

Islam, M. M., Marchetto, A., Susi, A., Kessler, F. B., and Scanniello, G. (2012). Motcp: A tool for the prioritization of test cases based on a sorting genetic algorithm and latent semantic indexing. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 654–657.

JaCoCo (n.d.). Jacoco java code coverage library. https://www.eclemma.org/jacoco/. Accessed: 2023-11-19.

Kandil, P., Moussa, S., and Badr, N. (2017). Cluster-based test cases prioritization and selection technique for agile regression testing. *Journal of Software: Evolution and Process*, 29(6):e1794. e1794 JSME-15-0111.R1.

Khalid, Z. and Qamar, U. (2019). Weight and cluster based test case prioritization technique. *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1013–1022.

Lima, J. A. P. and Vergilio, S. R. (2022). A multi-armed bandit approach for test case prioritization in continuous integration environments. *IEEE Transactions on Software Engineering*, 48(2):453–465.

Medhat, N., Moussa, S. M., Badr, N. L., and Tolba, M. F. (2020). A framework for continuous regression and integration testing in iot systems based on deep learning and search-based techniques. *IEEE Access*, 8:215716–215726.

Mix (n.d.). Mix. https://hexdocs.pm/mix/Mix.html. Accessed: 2023-11-19.

muJava (n.d.). mujava: a mutation system for java programs. https://cs.gmu.edu/~offutt/mujava/. Accessed: 2023-11-19.

MutGen (n.d.). Mutgen - motif based mutation simulation library. https://github.com/fhcrc/mutgen. Accessed: 2023-11-19.

Omri, S. and Sinz, C. (2022). Learning to rank for test case prioritization. In *2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing (SBST)*, pages 16–24.

Pan, R., Ghaleb, T. A., and Briand, L. (2022). Atm: Black-box test case minimization based on test code similarity and evolutionary search. *arXiv preprint arXiv:2210.16269*.

Pan R., Bagherzadeh M., G. T. e. a. (2022). Test case selection and prioritization using machine learning: a systematic literature review. *Empir Software Eng*, 29:1 – 43.

Pradeepa, R. and VimalDevi, K. (2013). Effectiveness of test case prioritization using apfd metric: Survey. In *International Conference on Research Trends in Computer Technologies (ICRTCT—2013). Proceedings published in International Journal of Computer Applications®(IJCA)*, pages 0975–8887.

Qu, X., Cohen, M., and Woolf, K. (2007). Combinatorial interaction regression testing: A study of test case generation and prioritization. In *2007 IEEE International Conference on Software Maintenance*, pages 255—264, Los Alamitos, CA, USA. IEEE Computer Society.

Sampath, S., Bryce, R. C., Jain, S., and Manchester, S. (2011). A tool for combination-based prioritization and reduction of user-session-based test suites. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 574–577.

Singhal, S., Jatana, N., Suri, B., Misra, S., and Fernandez-Sanz, L. (2021). Systematic literature review on test case selection and prioritization: A tertiary study. *Applied Sciences*, 11(24).

Spieker, H., Gotlieb, A., Marijan, D., and Mossige, M. (2017). Reinforcement learning for automatic test case prioritization and selection in continuous integration. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2017, page 12–22, New York, NY, USA. Association for Computing Machinery.

Vescan, A., Gaceanu, R., and Szederjesi-Dragomir, A. (2023). Neural network-based test case prioritization in continuous integration. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, pages 68–77, Los Alamitos, CA, USA. IEEE Computer Society.