Generation of Regression Tests From Logs with Clustering Guided by Usage Patterns

Frédéric Tamagnan[1], Alexandre Vernotte[2], Fabrice Bouquet[1], and Bruno Legeard[1]

[1]Dept. DISC - Femto-ST, Univ. of Franche-Comté, Besançon, France
[2]R&D, Smartesting, Besançon, France
[*]Correspondence: Frédéric Tamagnan, Dept. DISC - Femto-ST, Univ. of Franche-Comté, Besançon, France, frederic.tamagnan@gmail.com

Abstract

Clustering is increasingly being used to select the appropriate test suites. In this paper, we apply this approach to regression testing. Regression testing is the practice of verifying the robustness and reliability of software by retesting after changes have been made. Creating and maintaining functional regression tests is a laborious and costly activity. To be effective, these tests must represent the actual user journeys of the application. In addition, an optimal number of test cases is critical for the rapid execution of the regression test suite to stay within the time and computational resource budget as it is re-run at each major iteration of the software development. Therefore, the selection and maintenance of functional regression tests based on the analysis of application logs has gained popularity in recent years. This paper presents a novel approach to improve regression testing by automating the creation of test suites using user traces fed into clustering pipelines. Our methodology introduces a new metric based on pattern mining to quantify the statistical coverage of prevalent user paths. This metric helps to determine the optimal number of clusters within a clustering pipeline, thus addressing the challenge of suboptimal test suite sizes. Additionally, we introduce two criteria, to systematically evaluate and rank clustering pipelines. Experimentation involving 33 variations of clustering pipelines across four datasets[1] demonstrates the potential effectiveness of our automated approach compared to manually crafted test suites. Then, we analyze the semantics of the clusters based on their principal composing patterns.

Keywords— Clustering, Pattern Mining, Regression Test Selection, User Traces

[1]All the experiments and data on Scanner, Spree and Booked Scheduler are available at https://github.com/frederictamagnan/STVR2024

1 Introduction and Background

1.1 Introduction

The primary goal of regression testing is to maintain the integrity of the software and ensure that it continues to meet its requirements and specifications throughout the development process, in harmony with the expected usage patterns. This involves re-running a set of predefined test cases on the modified code to ensure that the previously working features still work correctly after the changes. However, this process can be time-consuming and resource-intensive, as regression tests must be executed with each software development iteration. Quality Assurance engineers can take certain steps to minimize the associated costs. Firstly, they can automate the regression testing process, from creating the test suite to its execution. This automation helps streamline the testing procedure and save time and effort. Secondly, QA engineers can aim to find an optimal test suite that provides sufficient coverage for the software under test. By identifying a minimal yet effective set of tests, they can minimize redundancy and focus on the most critical aspects of the software. Regression test suites are typically assessed using metrics like the Average Percentage of Faults Detected (APFD) or code coverage. However, some of these metrics require code instrumentation, which may not be feasible in certain contexts, such as when using proprietary software products, or may not adequately capture the statistical significance of covering real user paths.

Given the substantial volume of data that IT systems now collect, including user execution traces, there is an opportunity to derive new tests from this resource [1].

Several approaches exist for generating regression test suites based on user traces [2, 3], and some of these methods employ clustering. Clustering, a machine learning technique already used for test suite prioritization [4–7], can also be directly applied to user traces. This involves segmenting the user traces into distinct behavioral groups and selecting representatives from each group to create a comprehensive regression test suite. When utilizing clustering for this purpose, two main challenges arise. First, one must decide on the clustering pipeline, encompassing aspects such as trace encoding, clustering models, and the extraction of representatives. Secondly, determining the number of clusters is a critical issue, as it directly correlates with the number of tests to be selected. Indeed, the effectiveness of clustering models in disentangling traces can vary depending on factors like size, sequential nature, dataset balance, and volume. The same clustering model will not yield similar performance when applied to a dataset with a very limited vocabulary but millions of sequences compared to a dataset with an extensive vocabulary and only a few hundred sequences. Additionally, it is necessary to establish a stopping criterion in the search for the optimal number of clusters and determine a sampling strategy for selecting user traces as test candidates. The evaluation of the partition's relevance and the ideal number of clusters is subject to disagreement among generic internal metrics [8–16]. Hence, a significant hurdle arises due to the lack of a domain-specific metric to assess how effectively the generated regression tests reflect the real software usage, which would allow to fine-tune clustering pipelines and benchmark them.

In previous work [16], we addressed this problem by introducing a statistical coverage metric based on action n-grams representativeness (API Calls, click on the web interface) in order to assess whether a regression test suite aligned with the actual usage of an application. Next, we used this

metric to extract tests: using a clustering pipeline, we performed clustering on the execution traces and then extracted one trace per cluster to create a test. We increased the number of clusters by 1 until the action n-grams statistical coverage was sufficient. This metric allowed us to determine the ideal number of clusters for each clustering pipeline and also to compare them, with the best one being the one that achieves better coverage with the smallest number of clusters/tests.

In this paper, we extend our previous work as follows:

- **Introduction of a novel metric:** we introduce a novel statistical coverage metric based on pattern mining to evaluate if a regression test suite is relevant with respect to user traces of the SUT. This metric is more robust than the one from [16] as patterns excel over n-grams in capturing long-term dependencies. This metric is absolute and bounded.

- **Fine-tuning of clustering pipelines:** we employ our novel metric to establish a fine-tuning process for clustering pipelines and determine the optimal number of clusters. This approach enhances the accuracy and efficiency of clustering-based test suite selection, ultimately improving the quality of regression testing.

- **Benchmarking of clustering pipelines:** Two criteria are introduced for comparing pipelines based on the testers' requirements. One emphasizes maximum coverage, and the other prioritizes the ratio between the number of tests and coverage. This approach allows benchmarking different clustering pipelines without the need to instrument the code. The experimentation includes 33 variations of clustering-based test selection pipelines, with a significant emphasis on the embeddings stage, recognized as a critical step in the clustering process[17]. These experiments were conducted on four datasets from different web applications, resulting in the creation of test suites with better coverage performance than human-written ones.

- **Open-source code on 2 public datasets:** The code of the experiments is provided in an open repository to allow the reproducibility of results.

To sum up, the methodology provides the means to enhance the test objective by incorporating as test cases the most frequent user behavior through clustering. Since a bug in an application is only problematic if encountered by users, focusing the testing effort on actual user behavior increases the likelihood of a bug-free user experience.

The remainder of the paper is structured as follows: sections 1.2 and 1.3 provides notations, definitions and research questions; section 2 presents related works; section 3 defines our new metric and outlines our approach for fine-tuning and benchmarking clustering pipelines along with associated criteria; section 4 presents experiments and results, measuring the performance of clustering pipelines on four datasets; section 5 encompasses discussions, and section 6 conclusion and perspectives, where we reflect on the findings and outline future directions in our research.

1.2 Definition and Notation

1.2.1 Traces and ML pipelines

From a web application perspective, a user trace is defined as a sequence of GUI user events and/or back-end API calls, also called *Events*, that a user has triggered while browsing a web application. In this paper, the following notation is used, x for a user trace and X for a set of traces. t denotes a test and T is a set of tests.For example $\mathcal{I} = \{login, logout, add, delete, changeQuantity\}$ would be the set of API Calls for an e-commerce software. A user trace could be $x = \langle login, add, add, delete, logout\rangle$ for a user browsing the website and adding items to their basket and a test $t = \langle login, logout\rangle$ testing the login and the logout of a user.

In the state-of-the-art, the general pipeline to select tests from user traces is the following:

- Preprocessing: Let $\rho(X)$ be the preprocessing stage that transforms a set of user traces X into a numerical representation X_e
- Clustering: Let $\phi(X_e)$ be the clustering stage that partitions numerical representation of traces into k clusters. ϕ takes X_e as input and returns a set of cluster labels Y. The values of Y range from 1 to k
- Sampling: Let $\psi(X, Y)$ be the sampling stage that takes a set of user traces X and their cluster labels and returns a set of Tests T. **In this paper, one test per cluster is sampled, resulting in a test suite of k tests.**

So in the following, reference to a certain clustering pipeline is noted as $\Omega(X)$, Ω being the combination of the three stages detailed above. Ω takes a set of traces X and return a set of tests T. $\Omega(X) = (\psi \circ \phi \circ \psi)(X)$. The pipeline applied to the user traces with k as the number of clusters for ϕ is denoted as $\Omega_k(X)$. As the result of $\Omega_k(X)$ can be different each time, the clustering model ϕ reaching a local optimum, or due to the randomness of the sampling stage ψ, the result of Ω for a specific run r is marked as $\Omega_k^r(X)$.

1.2.2 Sequential Pattern Mining (Closed Sequential Patterns)

Sequential pattern mining is the task of identifying all frequent subsequences within a sequence database that meet or exceed a user-defined minimum frequency. It is an enumeration problem, as it involves listing all subsequences that satisfy the given constraints. Such as in market basket analysis, it identifies frequently co-occurring items in transactions, revealing patterns like which products are regularly bought together. For example, it can determine that customers often purchase bread, milk, and eggs in that specific order, helping retailers optimize product placement and promotions. Various algorithms have been developed to perform this enumeration task, each consistently producing the same set of frequent sequential patterns under identical conditions [18]. However, these algorithms differ significantly in their methodologies and efficiency. Moreover, given the vast search space of possible subsequences, efficient sequential pattern mining algorithms must integrate techniques to avoid full search space exploration. To ensure clarity in the ensuing discussion, key terms used in sequential pattern mining are defined below.

Definition 1.1 (Sequence)**.** A sequence s is a tuple $s = \langle I_1, I_2 \dots I_n \rangle$ with $I_i \in \mathcal{I}$ a set of items, and $\forall i : 1 \leq i \leq n$. In our case, the set of items can be all the API calls of our system. For example $\mathcal{I} = \{login, logout, add, delete, changeQuantity\}$ would be the set of items for an e-commerce software. In this context, a sequence is a user trace, $s = \langle login, add, add, delete, logout \rangle$. In the general context of Sequential Pattern Mining, each element of a sequence can comprise multiple events. For instance, the following sequence $s = \langle login, (add, delete), (add, changeQuantity, logout) \rangle$ could represent several actions occurring simultaneously (elements within the same parenthesess). However, this does not apply to our framework.

Definition 1.2 (Subsequence)**.** $\alpha = \langle I_{a_1}, I_{a_2}, \dots I_{a_n} \rangle$ is a subsequence of another sequence $\beta = \langle I_{b_1}, I_{b_2}, \dots I_{b_m} \rangle$ (or β is a supersequence of α), denoted as $\alpha \preceq \beta$, if there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $I_{a_1} = I_{b_{j_1}}, I_{a_2} = I_{b_{j_2}}, \dots, I_{a_n} = I_{b_{j_n}}$.

For example, in the more practical context of traces, $s = \langle login, delete \rangle$ is a subsequence of $t = \langle login, add, delete, logout \rangle$, because $I_{s_1} = I_{t_1}$ and $I_{s_2} = I_{t_3}$ and the order in the items is preserved.

Definition 1.3 (Support)**.** The *support* (or *frequency*) of a sequence α, denoted as $\sigma(\alpha, D)$, represents the number of input sequences in the database D that include α. A sequence or pattern is considered frequent if it appears at least as many times as a user-defined threshold, min_sup, known as the minimum support. The entire set of frequent sequences is denoted by $\mathcal{FS}$. The task of frequent sequence mining involves identifying $\mathcal{FS}$ in a given database based on a specified minimum support threshold. We can also define the support as a fraction of the total length of D, such as 30%

Definition 1.4 (Closed Sequence)**.** A frequent sequence α is considered a closed sequence if there is no supersequence of α with the same support. Conversely, if a frequent sequence β has a supersequence γ with identical support, then β is termed a non-closed sequence, and γ is said to *absorb* β. The complete set of frequent closed sequences is represented as $\mathcal{FCS}$. Formally, $\alpha \in \mathcal{FCS}$ if $\forall \beta \in \mathcal{FS}$, $\alpha \preceq \beta \implies \sigma(\alpha, D) \neq \sigma(\beta, D)$. The task of closed sequence mining involves identifying $\mathcal{FCS}$ within a given input database, based on a specified minimum support threshold.

An example of the computation of frequent closed patterns is provided in Section 3.1.3.

1.3 Research Questions

There are three main research questions addressed in this paper:

- **RQ1: What is the representativeness of a test suite with respect to the actual usage?** As it is difficult to obtain a test suite that reflects the actual usage of a system, the creation of a metric that represents the disparity between user traces and regression tests to qualify the relevance of a test suite is needed.
- **RQ2: Can clustering of user sessions help to segment representative usage in an optimal number of clusters?**

By selecting test candidates among clusters of user traces, it is wished to get a representative test suite. For this task, it is essential to select the optimal number of clusters from user traces. Picking at least one test per cluster, having too many clusters may lead to a test suite with redundant test cases. Not having enough clusters may lead to a test suite that insufficiently covers the observed user behavior on the system. To address this question, a specific criterion needs to be defined that allows the detection of whether there are too many or too few clusters.

- **RQ3: To what extent do clustering pipelines produce different results?** This paper hypothesizes that various clustering models, including those utilizing neural embeddings, will yield divergent outcomes when employed on datasets with differing characteristics. If this is the case, selecting the most appropriate pipeline for each case study is essential. It is necessary to ascertain the validity of this assumption and establish methods for benchmarking these pipelines. Additionally, the paper aims to evaluate whether the use of neural embeddings offers significant advantages over traditional clustering approaches. After the completion of clustering on user traces, one or more traces are chosen from each cluster as potential test candidates. Various techniques from the state-of-the-art employ random sampling; the suitability of this approach, which seems limited, needs to be evaluated.

2 Related Works

In the following section, we explore related works, categorizing them into four primary areas: regression test selection and prioritization with clustering, test generation using clustering applied to user traces, logs embeddings leveraging NLP Models, and the evaluation of clustering methods.

2.1 Regression Test Selection and Prioritization with Clustering

Prior research has explored the application of semi-supervised clustering methods, such as SSKM, to enhance regression test selection, demonstrating improvements in test selection performance and underscoring the significance of pairwise constraints in this context [4]. Kandil et al. [5] unveiled an automated agile regression testing approach comprising two key techniques, WSTP and CRTS, which prioritize sprint test cases based on agile parameters and select release test cases through clustering and text mining, improving the efficiency and effectiveness of regression testing for agile projects. Almaghairbe et al.[6] investigated clustering techniques (Agglomerative hierarchical, DBSCAN, and EM) for automated test oracle creation. The study found that smaller clusters often contain a high proportion of failures, offering practical implications. Previous work has shown that clustering-based test case prioritization in an industrial software context can enhance fault detection rates and reduce fault escapes in truncated testing scenarios as well (Carlson et al. [7]).

2.2 Test Generation With Clustering Applied on User Traces

Clustering approaches can serve as a valuable tool for extracting insights from user traces. These techniques enable the identification of prominent user behaviors, facilitating the discovery of a com-

prehensive set of test cases that effectively cover a substantial portion of the system [19–23]. Moreover, clustering can aid in the detection of any absent tests within the realm of user traces [24].

Luo et al. [19] harnessed a user session-based testing technique that clusters user sessions using applications' service profiles and augments selected sessions to cover dependence relationships between web pages, resulting in a significantly reduced test suite size while maintaining fault detection rates, as demonstrated through empirical studies. Liu et al. wielded the USCHC (User Sessions Clustering based on Hierarchical Clustering algorithm) method for web application test case optimization, utilizing a bottom-up hierarchical clustering algorithm to cluster initial test cases, select representative test cases, and improve testing efficiency as demonstrated in experiments [20]. Similarly, with K-medoids, Li et al. [21] introduced a method for testing web applications based on user session data using clustering techniques, with the algorithm selecting less than 3% of real user sessions as test data and demonstrating its effectiveness in covering web application code and discovering faults. Dorcis et al. [22] utilized a Sequences String Comparison Clustering method, drawing inspiration from gene sequencing comparisons. This approach involves extracting string patterns from execution traces to choose user traces for testing purposes. Afshinpour et al. [23] deployed a test suite reduction approach using natural language processing techniques, particularly Word2Vec, to identify similarities between tests, resulting in significant reduction while maintaining fault detection capabilities, demonstrated through a case study involving mutants for evaluation. Utting et al. [24] employed the MeanShift algorithm to detect missing regression tests by conducting joint clustering of user traces and test traces, effectively identifying clusters that do not have associated tests.

2.3 Logs embeddings with NLP models

When feeding execution traces using trace execution, the textual information needs to be transformed into numbers. Naive NLP methods such as Bag-of-words or Tf-IDF can be employed, but the last ten years of advances in NLP can also help us build more meaningful representations of sequences of words. Stocco et al. [25] have trained a Doc2vec model on a large corpus of web pages and propose WEBEMBED, an innovative abstraction function that utilizes neural network embeddings and threshold-free classifiers to prune in a relevant way redundant states in a model-based test generation perspective. Transformers have been used as well to transform execution traces of tests into embeddings in order to prioritize them. In Jabbar et al. [26], the authors fine-tune CodeBert [27] with a classifying task (either a test pass or fail) to build representations of tests and rank them later. Embeddings of logs have been employed as well for anomaly detection with various forms, Transformers encoding [28] or Autoencoder embeddings [29].

2.4 Evaluation of Clustering

To assess clustering models, there are two evaluation approaches: internal and external evaluation. In the current state-of-the-art in clustering, external evaluation is commonly employed due to its suitability for benchmarking multiple models [8, 9] using pre-labeled data. For this purpose, the v-measure, presented by Rosenberg et al. [10] as a unified measure to gauge model performance,

reflects the harmonic mean of completeness and homogeneity, where perfect labeling achieves a score of 1.0. On the other hand, internal evaluation primarily assesses the similarity within clusters and dissimilarity between elements from different clusters, making it valuable for hyperparameter tuning, especially for determining the optimal number of clusters. Notable internal evaluation metrics include the David-Bouldin Index [12, 13] and the Silhouette coefficient [14, 15], which aid in validating cluster consistency. Despite internal evaluation metrics seeming ideal for selecting the number of clusters in a model, as demonstrated by Rendon et al. [11], applying these metrics to three distinct use cases involving K-means clustering with Bag-of-Words preprocessing resulted in varying cluster numbers, a phenomenon not uncommon in clustering, posing a challenge in the context of regression test selection.

While prior research has boosted our confidence in using clustering to identify regression tests from user traces, three critical issues remain unaddressed in clustering for test selection: 1) determining the optimal number of clusters, 2) ranking clustering pipelines, 3) employing a universally applicable, bounded metric that reflects the coverage of principal usage patterns.

3 Test Selection Coverage Function for tuning and benchmarking

3.1 Usage Pattern Coverage Metric

In this section, the Usage Pattern Coverage (UPC) metric, is introduced. It comprises four key parts, each dedicated to a specific aspect of the metric. In the first subsection, the goal of the metric is detailed. In the second subsection, the metric's design is explained with a focus on mining frequent closed patterns. The third subsection provides an illustrative example to demonstrate how the UPC metric functions in practice (following the workflow of Figure 1). In the fourth subsection, we delve into the application of the UPC metric for test suite evaluation.

3.1.1 Goal of the metric

The main goal is to build a metric that:

- Reflects if a regression test suite covers the most representative user flows from a statistical perspective
- Measures the quality of regression testing suites with an absolute (compared to the relativity of generic internal evaluation metrics) and bounded (ranging from 0% to 100%) metric
- Does not need to instrument the code, to avoid difficulties in implementing the metric because the code of the SUT is not always easily accessible to the QA engineers.

3.1.2 A Usage Pattern Coverage Metric Based on Frequential Closed Patterns

The UPC metric aims to measure how effectively a test suite captures the principal behaviors observed in user traces. To achieve this, we utilize sequential pattern mining to extract frequent

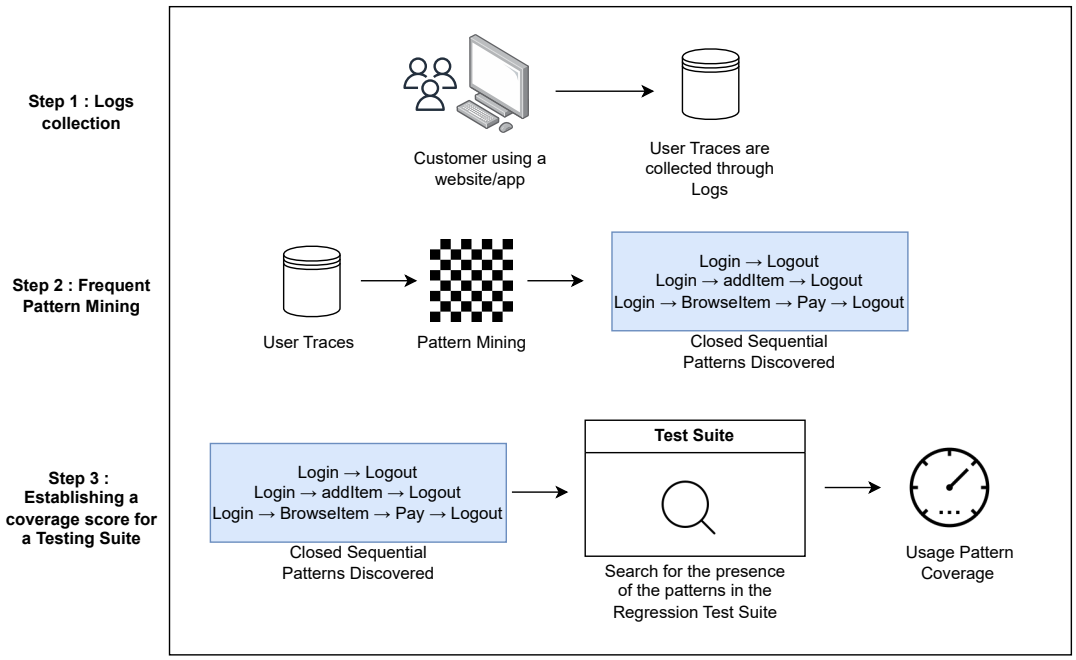


Figure 1: Workflow of how Usage Pattern Coverage metric is built and used

closed patterns, which are defined earlier in Section 1.2.2. Mining frequent closed patterns instead of mere frequent patterns ensures our outputs are both concise and representative, as closed patterns include all significant items without redundancy. We have selected the ClaSP algorithm [30] due to its efficiency and its implementation availability in the SPMF library [31] with Python wrappers. Once a set of frequent closed patterns P_X is extracted from a dataset of user traces, the UPC metric is defined to evaluate how well these patterns are represented in the test traces. The UPC metric is calculated as the ratio of the number of frequent closed patterns from P_X found in the test traces to the total number of frequent closed patterns in P_X from the user traces, weighted by the frequency of each pattern in the user traces. Essentially, this metric assesses the extent to which a test suite includes the principal behaviors observed in user traces, weighted by their occurrence.

This approach allows the UPC metric to capture the (sub)sequential nature of user interactions rather than merely counting individual API calls or n-grams of API calls. By focusing on frequent closed patterns, the UPC metric provides a robust measure of test suite coverage, ensuring that the primary user behaviors are thoroughly tested.

Definition 3.1 (Usage Pattern Coverage)**.** Let T be a test suite needed to be evaluated over X a set of user traces. Let P_X be the list of frequent closed patterns in X present above $min\_sup = \theta$, P_T the list of frequent closed patterns from P_X present in T, and $Count_X(p)$ the number of occurrences of a specific pattern p in X (number of traces from X where p is present). Then the Usage Pattern

Coverage of T with respect to X is defined as the following:

$$UPC_\theta(T, X) = \frac{\sum_{p \in P_T} Count_X(p)}{\sum_{p \in P_X} Count_X(p)} \quad (1)$$

By definition, the Usage Pattern Coverage (UPC) is bounded between 0 and 100%. 0% if none of the closed frequent patterns found in the user traces are present in the evaluated test suite. 100% if all those patterns are present in the test suite.

3.1.3 An illustrative Example

- Let a dataset X of 5 user traces x_1, x_2, x_3, x_4 and x_5 recorded from a SUT (Table 1a)
- The exhaustive list of possible API calls is *login*, *add*, *delete* , *logout* and *changeQuantity*
- Let a test suite T composed of 2 tests t_1 and t_2 (Table 1b)
- User traces or test traces are sequences of API calls and share the same vocabulary

Session id	Api Call Sequence
x_1	login add logout
x_2	login add add logout
x_3	login add delete logout
x_4	login add add
x_5	login add changeQuantity logout

(a) List of user traces

Test id	Api Call Sequence
t_1	login add delete logout
t_2	login login

(b) List of tests

Table 1: Elements of illustrative example

Computation of the UPC of T with respect to D of Equation (1) with minimum support equals to 30%:

1. Extract with the ClaSP algorithm the frequent closed patterns contained in the user traces data (column patterns of Table 2)
2. Count the occurrence of each frequent closed pattern in the user traces data (column A of Table 2)
3. Identify the presence of frequent closed pattern in the test suite (column B of Table 2)
4. Compute the weighted presence of each pattern in the test suite with respect to the user traces data (A times B, column A·B of Table 2)
5. Sum the A column and the A·B column separately. Dividing the sum of A·B by the sum of A gives us the UPC of the test suite. In this example, the usage coverage of the test suite is 81.8%

As noted, some patterns, such as *login* → *add* → *changeQuantity* were not extracted as part of frequent closed patterns because they appeared in fewer sequences, resulting in a support below the chosen minimum threshold of 30% (e.g., 20% for this specific pattern).

Frequent Closed Patterns	A	B	A·B
	# in user traces	In Test Suite ?	
$login \rightarrow add \rightarrow add$	2	False	0
$login \rightarrow add \rightarrow logout$	4	True	4
$login \rightarrow add$	5	True	5
sum	11	-	9

Table 2: List of frequent closed patterns of the illustrative example and their counts in the user traces A, their presence in the test suite B, and their weighted presence in the test suite A·B

3.2 Test Selection Coverage Function for tuning and benchmarking

3.2.1 Proposed Method and Examples

In the following subsections, we introduce the Test Selection Coverage function, which is derived from the UPC metric. To aid in the comprehension of our methodology, we commence with intuitive examples that shed light on its underlying principles. Subsequently, we move from intuition to a formal description of the Test Selection Coverage function and its properties, the methodology to tune a clustering pipeline and the criteria for benchmarking. To underscore its efficacy, we carry out a comparative analysis of the UPC metric against other generic metrics. This comparative study showcases the superior suitability of the UPC metric for the purposes of pipeline tuning and benchmarking. A metric has been introduced above that gives the coverage of a test suite with respect to the most frequent closed patterns present in the usage traces. The idea is to use this method to 1) tune our clustering pipelines to find the right number of clusters and 2) benchmark them to elect the best one.

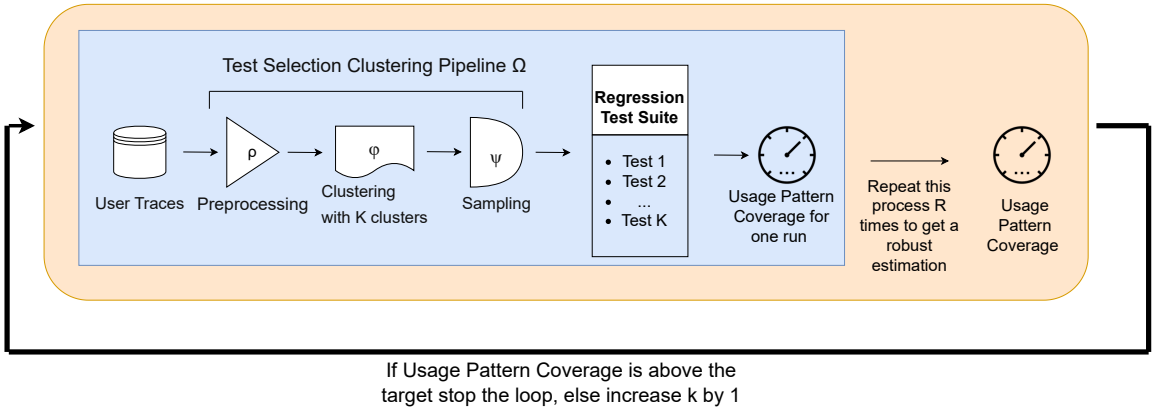


Figure 2: Tuning the Clustering Pipeline to find the best k: k^*

Tuning : Finding The Right Number Of Clusters For tuning the clustering pipelines and finding the best number of clusters, the idea is similar to the elbow method with distortion: increase the number of clusters until a point at which our objective is satisfied. In this case, the objective is

coverage (UPC) above a threshold. This method is described in Figure 2. After setting a number of clusters k_0, a test suite with k tests is extracted, one test per cluster. Then, the UPC of the test suite for one run of the clustering pipeline is computed. This operation is repeated a great number of times and averaged to obtain a robust estimate of UPC. If the UPC has reached a target threshold, the optimal k, k^* has been found. If not, k is raised by 1. This process coincides to study of the values of the curve representing the UPC with respect to the number of clusters (fig 3).

3.2.2 Benchmarking : Which Clustering Pipeline Is Better ?

Between two clustering pipelines Ω_1 and Ω_2, it is needed to elect which one is the best for a specific use-case, for a pre-determined maximum test budget.

First criterion: smallest k^* The first criterion is to consider that for a target coverage fixed, the pipeline reaching this coverage with the smallest k^* is the best pipeline, as this pipeline offers a smaller test suite than the other (as one test per cluster is extracted). This criterion puts the emphasis on the target coverage being necessarily reached in terms of software requirements.

Second criterion: Area Under the UPC Curve Some examples show that the first criterion is very sensitive to the target coverage threshold chosen.

Example 1: In Figure 3, two clustering pipelines are studied, Ω_1 and Ω_2. If a target coverage of 88% is chosen, their performance is equal, k^* is equal to 11 in both cases. They reach the target coverage of 88% in 11 clusters/tests. If a target coverage of 89% is chosen, the Ω_2 is better, reaching the target coverage of 89% in 11 clusters/tests whereas Ω_1 is reaching the target coverage in 13 clusters/tests.

Example 2: In Figure 4, with two other clustering pipelines, if a target coverage of 80% is chosen, the Ω_1 is better, reaching the target coverage in 4 clusters/tests whereas Ω_2 is reaching the target coverage in 10 clusters/tests. Whereas if a target coverage of 95% is chosen, the Ω_2 is better, reaching the target coverage in 12 clusters/tests. The Ω_1 is not able to reach the target coverage even in 25 clusters/tests. To sum up:

1. The choice of the best pipeline is sensitive to the target coverage chosen
2. When the smallest k^* is chosen as the criterion, two pipelines can have the same k^*, and it can be an equality
3. In the event that two pipelines fail to achieve the target coverage, comparing them becomes challenging.

A second criterion is to compare them with respect to the entire range of target coverage. This conforms to the idea of comparing the area under the curve (AUC) drawn. In this case, a normalized X-axis is employed. So that the AUC is bounded between 0 and 1. With an AUC of 1 being the best performance. This idea has already been employed for the ROC curve machine learning metric

[32]. The normalized AUC is computed for the pipelines Ω_1 and Ω_2 in Figure 5 and the pipeline Ω_1 obtains a better normalized AUC than Ω_2 (0.85 vs 0.75). This second criterion emphasizes on the ratio number of tests vs coverage for a pre-determined maximum test budget.

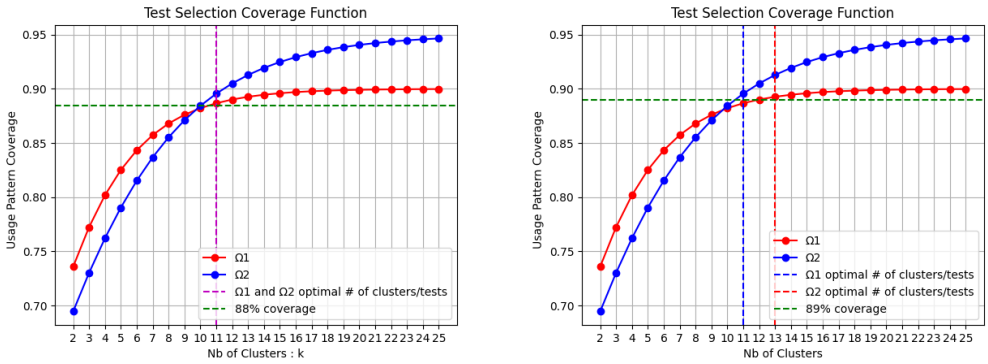


Figure 3: Examples of 2 clustering pipelines with their Test Selection Coverage Function, for a target coverage of 88%, the two pipelines perform equally. For a target coverage of 89%, Ω_1 has a smaller k^*

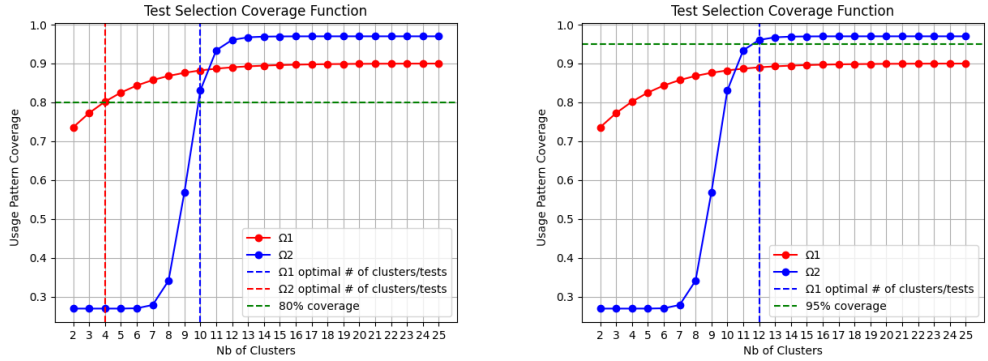


Figure 4: Examples of 2 clustering pipelines with their Test Selection function coverage, for a target coverage of 80%, Ω_1 has a smaller k^*. It is the contrary for a target coverage of 90%.

To summarize, those definitions of tuning and benchmarking align to the properties of the curve drawn by increasing the number of clusters of Ω, which will be called Test Selection Coverage Function in the next Part.

3.2.3 Test Selection Coverage Function

Definition 3.2 (Test Selection Coverage Function)**.** Let Ω_k be a clustering pipeline whose the number of clusters hyperparameter is k, X a set of user traces, T_{Ω_k} the test suite extracted with the clustering pipeline Ω_k, R the number of runs.

The test selection coverage function is

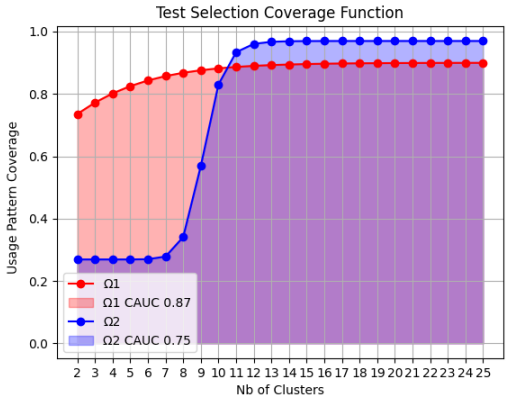


Figure 5: The pipeline Ω_1 obtains a better normalized CAUC than Ω_2 (0.85 vs 0.75).

$$TSC_\Omega(k) = \frac{1}{R}\sum_{r=1}^{R} UPC(\Omega_k^r(X)) = \frac{1}{R}\sum_{r=1}^{R} UPC(T_{\Omega_k}) \tag{2}$$

The Test Selection Coverage function measure the evolution of the average value for R runs of the UPC of a clustering pipeline Ω_k with respect to the number of clusters. Empirically, this function is increasing. It will be used to detect what is the optimal number of clusters/tests (as we extract one test per cluster), which is reached when the function starts to be asymptotic.

3.2.4 Tuning a clustering pipeline

Definition 3.3 (Optimal number of cluster for Ω, k^*)**.** Let Ω_k be a clustering pipeline whose the number of clusters hyperparameter is k, X a set of user traces, T_{Ω_k} the test suite extracted with the clustering pipeline Ω_k, TSC_Ω the test selection coverage function declared above, and K_{max} the maximum budget of tests.

k^* where TSC_Ω reaches a target coverage UPC^* is defined as follows:

$$k^* = \begin{cases} \min\{k \,|\, TSC_\Omega(k) \geq UPC^*, k \leq K_{\max}\} & \text{if such } k \text{ exists} \\ \infty & \text{otherwise} \end{cases}$$

In other words, the optimal number of clusters k^* for a clustering pipeline is defined as the smallest value of the number of clusters when the UPC of a clustering pipeline reaches a target coverage, for example 80%. This k^* exists only if it is under a maximum budget of test $K_{\max}$ fixed in advance by the tester.

3.2.5 Benchmarking the clustering pipelines

First Criterion: Smallest k^*

Definition 3.4 (Ranking of clustering pipelines based on k^*)**.** Let Ω_1, Ω_2 be two clustering pipelines, Ω_1 is considered better than Ω_2 if: $k^*(\Omega_1) < k^*(\Omega_2)$

	Davies Bouldin Index	Silhouette Score	V-Measure	Defect Detection Rate	Line Coverage	UPC
P1: Bounded	×	✓	✓	✓	✓	✓
P2: Does not need ground truth labels	✓	✓	×	✓	✓	✓
P3: Does not need system instrumentation	×	×	×	×	×	✓
P4: Quantify testing effectiveness	×	×	×	✓	✓	✓
P5: Statistically reflects user paths	×	×	×	×	×	✓

Table 3: Properties of the UPC and other generic metrics used in machine-learning and software testing

It is assessed that one clustering pipeline outperforms another in a specific use-case if it achieves the desired coverage with a smaller number of clusters, and consequently, a reduced number of tests.

Second Criterion: Best CAUC

Definition 3.5 (CAUC)**.** Let Ω be a clustering pipeline, the area under the curve of the Test Selection Coverage function is defined as

$$CAUC(\Omega) = \sum_{i=2}^{K_{max}-1} \frac{1}{2} \left(TSC_\Omega(k_i) + TSC_\Omega(k_{i+1})\right) \cdot \Delta k$$

The area under the curve of the Test Selection Coverage Function is a good measure of the performance of a clustering pipeline regardless a specific target coverage. It measures the clustering pipeline's ability to achieve a high ratio of UPC per test. The CAUC can be interpreted as the mean value of the UPC calculated across all possible number of clusters values

Definition 3.6 (Ranking of clustering pipelines based on CAUC)**.** Let Ω_1, Ω_2 be two clustering pipelines, Ω_1 is considered better than Ω_2 if: $CAUC(\Omega_1) > CAUC(\Omega_2)$

To summarize, a clustering pipeline is considered better than another if it has a higher ratio of UPC per test

3.3 Comparison of UPC with other Generic Metrics

Five properties can be listed to assess the relevancy of metrics to tune and benchmark clustering pipelines.

1. *P1 - Bounded:* Metrics with well-defined boundaries allow us to determine if we are close to the optimal number of clusters or if a clustering pipeline is suitable for production. When a property is bounded, it helps in understanding how close the result is to perfection. In contrast, metrics that range from 0 to infinity can be used to rank models but do not provide clear indications of whether a model is good or optimal for practical deployment.

2. *P2 - No Ground Truth Labels Needed:* Metrics should not depend on ground truth labels, enabling the evaluation of clustering effectiveness in the absence of labeled data, which is often the case for QA engineers generating regression test suites.

3. *P3 - No System Instrumentation Needed:* Metrics should not rely on system instrumentation, allowing efficient evaluation of clustering pipelines without the need for resource-intensive activities like obtaining code coverage or running multiple test iterations.

4. *P4 - Quantify Testing Effectiveness:* Metrics should accurately measure testing effectiveness, ensuring tests align with specified criteria beyond traditional machine learning measures.

5. *P5 - Statistically Reflect User Paths:* Metrics should statistically represent user behavior patterns, capturing both the structural and statistical characteristics of user paths within the data.

UPC metric is bounded between 0 and 1 as the denominator will be always superior or equal to the numerator and verifies P1. UPC metric does not need ground truth labels such as V-measure which needs pre-labeled data to compare ground truth labels with labels produced by the clustering (P2). UPC metric does not need to instrument the code to be computed and only need the user traces logs (P3). UPC metric quantifies testing effectiveness as it is one of the possible representation of the user paths coverage whereas pure machine learning metrics such as Davies Bouldin index (P4). UPC reflects user paths statistically as it is based on a frequent closed pattern mining algorithm (P5). To summarize, the UPC metric possesses the key properties discussed (Table 3), in contrast to machine learning metrics that may not align perfectly with software testing requirements. Unlike conventional software testing metrics, the UPC metric eliminates the need for software integration, simplifying the process. Furthermore, it enables to statistically reflect user paths, providing an advantage in assessing clustering for regression testing suites.

4 Experiments

This section focuses on the experimental evaluation of the proposed metric for assessing regression test suites derived from clustering on test traces. The primary emphasis lies in the preprocessing stage and the generation of embeddings, as these steps are considered among the most critical factors influencing the overall effectiveness of the clustering process. By carefully examining and optimizing these stages, the impact on the performance of the entire pipeline is demonstrated. Notably, in the current state of the art, neural embeddings utilizing autoencoders and transformers have not yet been extensively evaluated for this specific application. This work is innovative in that it explores these advanced embedding techniques within the context of clustering test traces, providing new insights and potential advancements in the field. Given the diverse nature of software datasets, different preprocessing and embedding techniques can yield varying levels of performance. Therefore, the experiments are designed to evaluate the extent to which these pipelines perform across different datasets, highlighting their robustness and adaptability. This evaluation is crucial since the optimization of the UPC corresponds to an NP-hard problem similar to the weighted set cover, which is akin to the test minimization problem. To provide a comprehensive evaluation, a vanilla baseline is included for comparison. Additionally, three pipelines from existing literature that address similar topics have been adapted [23][21][20]. These adaptations enable benchmarking the

approach against established methods, providing a clear context for the evaluation of the proposed metric.

4.1 Use cases and Datasets

In this section, we present the four datasets used to evaluate the approach and we finish with a synthesis.

4.1.1 Scanner

A supermarket scanner enables customers to scan product barcodes as they add items to their cart for later self-checkout. Our SUT pertains to the back-end API of such a supermarket scanner device, comprising 10 distinct *Events*, including actions like `Unlock` when a customer accesses a scanner, `Scan` for adding products with barcode recognition, and `Delete` for removing items from the internal shopping list. A second software, employing a probabilistic finite state machine, models customer behaviors during supermarket scanner use and facilitates the artificial generation of user traces, all while utilizing this API. These user traces are recorded as logs (see Table 4) by the SUT. The software was developed by a University of Franche-Comté teacher in Java for a software engineering class, and comes with regression tests provided by the same teacher.

Timestamp	UserId	API Method	ReturnCode	Param1	Param2
1674724050	client0	Unlock	0	[]	scan0
1674724050	client0	Scan	-2	[35705901109324]	scan0
1674724050	client0	Scan	0	[3017800238592]	scan0
1674724050	client0	Transmission	0	[Cashier3]	scan0
1674724050	client0	Abandon	?	[]	scan0
1674724050	client0	OpenSession	0	[]	Cashier3
1674724050	client0	Add	0	[35705901109324]	Cashier3
1674724050	client0	CloseSession	0	[]	Cashier3
1674724050	client0	Pay	5.5499	[90]	Cashier3

Table 4: Example of a user trace in the Scanner logs

4.1.2 Spree: Headless Ecommerce API

Spree is an open-source, modular e-commerce platform tailored for global brands, leveraging a REST API framework. It affords brands the flexibility to implement a headless e-commerce model, allowing them to easily integrate their preferred customer-facing front-end solution. Additionally, Spree provides an administrative interface for efficient management of inventory, products, orders, and customer accounts. The API is composed of 56 various methods, grouped by categories, including `Authentication` for managing the authentication of a user, `Cart, Line Items` for allowing a guest customer or a logged user to initialize a cart and start adding products, `Checkout` for managing the various check-out steps such as updating billing, shipments, and payment information, and `Shipments` for retrieving the various shipments proposed and computing the shipping rates.

UserId	Abstract Name	Method	URL	ReturnCode
312458	createACart	POST	api /v2/storefront/cart	201
312458	listAllTaxons	GET	api/v2/storefront/taxons/?per_page=100	200
312458	SortByPrice	GET	api/v2/storefront/products	200
312458	retrieveAProduct	GET	api/v2/storefront/products/long-sleeve-jumper	200

Table 5: Example of a user trace in the Spree logs

A second software was developed, utilizing a probabilistic finite state machine to model customer shopping behaviors via the API, enabling the artificial generation of customer execution traces for this use case as well (see Table 5).

4.1.3 Teaming

Teaming is a modular solution for IP telephony, conferences, and office automation tools developed by Orange. Teaming has a set of REST API web services whose API is composed of 97 various requests. Orange gave us real customers and test execution traces of this use case. APIs will not be documented for teaming for industrial privacy reasons.

4.1.4 Booked Scheduler

Booked Scheduler is a web-based scheduling software developed by Tinkle Toes Software Company designed to help organizations manage and schedule resources such as rooms, equipment, and personnel. Booked Scheduler provides a user-friendly interface for creating, viewing, and managing bookings or reservations. Anonymized user traces are obtained by monitoring the application in operation at the Femto-ST Laboratory at the Université de Franche-Comté. This application is used by over 300 researchers and technicians, and counts over 30 user traces per day. Some key features of Booked Scheduler include `Resource Management`, which allows you to define and manage various resources that can be scheduled, such as meeting rooms, equipment, and vehicles; `Booking Management`, where users can check the availability of resources, make bookings, and view their own or others' bookings, with administrators having the ability to manage and modify bookings as needed; and `Calendar Integration`, which synchronizes with external calendar applications like Google Calendar to provide seamless integration and avoid scheduling conflicts.

UserId	PathName	Type	TargetElement	TargetType	GenSelector
47ef34d0	Web/reservation.php	change	select	#BeginPeriod	-
47ef34d0	Webreservation.php	click	option	#BeginPeriod	:nth-child(5)
47ef34d0	Web/reservation.php	change	select	#EndPeriod	-
47ef34d0	Web/reservation.php	change	select	#EndPeriod	:nth-child(10)
47ef34d0	Webreservation.php	click	button	button.reservation	Buttons .pull-right-sm .btn-success

Table 6: Example of a user trace in the Booked Scheduler logs

For this dataset, an open-source front-end events collector made by the company smartesting is used. This allows to capture actions such as clicking on a button, clicking on a dropdown list, etc. with the associated locators.

4.1.5 Datasets Summary

- `Scanner`: 7000 generated user traces and 27 regression tests
- `Spree`: 4999 generated user traces
- `Teaming`: Orange provided a dataset of 9662 real user traces and 236 regression tests
- `Booked Scheduler`: collection of 4302 real user traces

Metrics	WMT 2014 English (0.44% subset)	Scanner	Spree	Teaming	Booked Scheduler
# of Samples	20000	7079	4999	9662	4302
Vocabulary Size	34046	14	29	95	267
Average Token Frequency per Sample	0.00	1.01	0.75	0.02	0.03
Maximum Consecutive Repetitions of a Token	11	19	2	3915	54
# of samples w/ more than 5 consecutive repetitions	2	5252	0	214	123
Average Number of Different Tokens per Sentence	22.57	7.12	11.99	1.42	5.6
Number of Rare Elements (one appearance only)	16679	0	1	8	0

Table 7: Analysis and comparison of datasets with respect to a set of NLP metrics

Several metrics are computed on those datasets to characterize their specificity (Table 7). Alongside, the same metrics are featured for a 0.44% subset of the English part of the WMT 2014 English-German dataset.

The rationale underlying these metrics is as follows: The number of samples is the total count of individual data points in a dataset. Additionally, the vocabulary size for an NLP model is the total number of unique tokens present in the dataset. The average token frequency per sample measures how often tokens recur across samples in a dataset. The maximum consecutive repetition for a token quantifies the highest number of times a token appears consecutively within a dataset. The number of samples where there is more than 5 consecutive repetitions for a token tracks the frequency of such repetitive patterns across different instances in a dataset. Moreover, the Average Number of Different Tokens per Sentence measures the diversity of vocabulary used in each sentence. Finally, the Number of Rare Elements metric counts how many unique tokens appear only once in a dataset.

The first thing to notice is the size of the trace datasets with respect to the NLP dataset. The whole WMT 2014 English dataset, which is used for training for Vaswani et. al [33], is larger than this paper's trace datasets. While the trace datasets contain under 10,000 samples, the standard WMT 2014 dataset contain 4.5 million. The vast difference in size between the WMT 2014 dataset (4.5 million samples) and the smaller trace datasets (less than 10,000 samples) suggests that models trained on the WMT can learn more complex and nuanced patterns due to greater data diversity, supporting more complex models and better generalization. In contrast, the smaller trace datasets may require simpler models to avoid overfitting, focusing on specialized pattern recognition within a narrower scope.

The other characteristic that is specific to trace datasets is the redundancy: there are often the same tokens recurring in sentences. Moreover, there is a substantial part of samples where the same token is repeated more than 5 times consecutively. The high redundancy and frequent consecutive repetitions of tokens in trace datasets make them more challenging than typical NLP datasets, as

they require specialized preprocessing and modeling techniques to effectively manage noise and avoid overfitting. These metrics computed over datasets highlight the differences between trace datasets and NLP datasets, and also reveal significant variations within the trace datasets themselves. As a result, the best NLP models from the state-of-the-art are not necessarily expected to encode trace datasets better.

4.1.6 Test suite evaluation with UPC metric

The minimum support of ClaSP has been tuned to extract each time 100 frequent closed patterns for each dataset (Table 8). For 3 datasets, all the frequent closed patterns above a threshold of 20% were extracted which seems reasonable in terms of looking for the most representative patterns of each use case. For Spree, it was not possible to set a lower threshold without leading to a combinatorial explosion. As human-written test suites for Scanner and Teaming are available, the UPC of each test suite is computed which gives respectively 77.2 % and 99.6 % . As those scores are below 100%, and with a relatively high number of tests, it gives a baseline to improve them.

Dataset Name	Scanner	Teaming	Spree	Booked Scheduler
Minimum Support	20%	0.5%	50%	6%
# of frequent patterns	10079	659	5183	455
# of frequent closed patterns	86	98	125	114
UPC of Human Test Suite	72.2%	99.6 %	-	-

Table 8: Number of patterns extracted for each dataset

In this section, the performance of diverse clustering pipelines is explored, with a focus on investigating the effectiveness of various combinations of encoding methods, clustering models, and sampling strategies across four distinct use cases thanks to the two criteria presented before. The overarching goal is to address the research questions and assess whether the top-performing pipelines within each use case can surpass human-written test suites. This begins with a detailed examination of the encoding techniques employed, followed by a comprehensive overview of the clustering models in use, and concludes with an examination of the sampling strategies in our analysis. Then, the results are analyzed.

4.2 Preprocessing

Each *Event* is the concatenation of the API method and the return code i.e. for the scanner dataset, an *Event* can be *scan_2* (see Figure 4). For the other datasets, we use the concatenation of all the columns featured in the examples of logs provided earlier. The preprocessing section serves the purpose of transforming a sequence of events into a vector of real numbers. This encoded representation is subsequently fed into a clustering model for further processing.

Bag of Words (BoW)

The Bag of Words approach encapsulates text by capturing word occurrence frequencies within a given document, neglecting word order. It yields a sparse vector where each dimension corresponds to a unique word, reflecting its frequency within the document. While BoW excels in simplicity and computational efficiency, it lacks context awareness and struggles with handling semantic nuances.

Term Frequency-Inverse Document Frequency (TF-IDF, or TermFreq in Tables)

TF-IDF refines BoW by weighing term frequencies against their inverse document frequencies. This amplifies the importance of terms that are distinctive to individual documents while diminishing the impact of ubiquitous words. TF-IDF succeeds in capturing document-specific relevance, making it valuable for information retrieval tasks. However, like BoW, it falls short of capturing semantic relationships.

Word2Vec

Word2Vec is a widely used natural language processing technique that transforms words into dense vectors, effectively capturing semantic relationships within a continuous vector space, with the primary task being to predict the context (surrounding words) in which a given word is likely to appear, making it valuable for various NLP applications. Reproducing [23], Word2Vec averaging is used to encode the user traces.

Auto-encoders and variations

Autoencoders (AE), Variational Autoencoders (VAE), and Denoising Autoencoders with Adversarial Training (DAAE) stand out in NLP embeddings for their ability to capture semantic information and reduce noise in text data. AEs effectively reduce dimensionality and distill meaningful features, while VAEs provide a probabilistic framework for generating coherent text embeddings. DAAEs enhance robustness through noise reduction in input text and emphasize discriminative features. These autoencoder variants contribute to the development of informative, context-rich embeddings, elevating their utility in various NLP tasks like sentiment analysis, text classification, and text generation. In the process of encoding traces, Autoencoders variations (implementation from [34]) are trained in a language modeling fashion and means from the latent space are extracted to form the embeddings.

Transformer-based Embeddings (TF)

Transformers are effective for NLP embeddings because their self-attention mechanism captures intricate contextual relationships in text. This enables them to generate embeddings that reflect the nuanced meaning and context of words within sentences and documents. In the process of encoding traces, Transformers are trained in a language modeling fashion and embeddings are extracted from the encoder layer (implementation from [33]).

Pattern One Hot (POH)

Pattern One-Hot encoding is a binary representation used for detecting the presence or absence of specific patterns within each trace. Each frequent closed pattern is assigned a dedicated binary column, where '1' indicates the presence of the pattern and '0' denotes its absence. This encoding offers an efficient and interpretable means of capturing pattern-related information in data.

Combination of Embeddings

In addition to embeddings, combinations (concatenations) of these embeddings are utilized, denoted by the '+' sign.

4.3 Clustering Models and Sampling Strategies

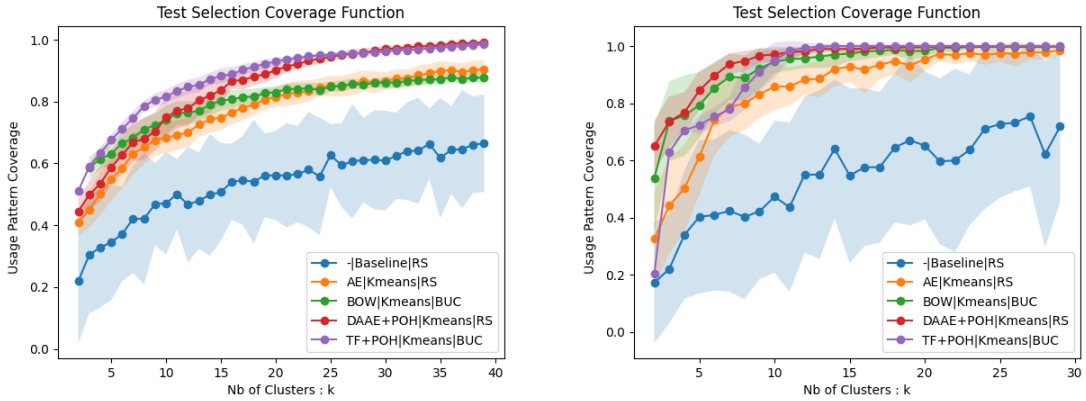
Clustering models take the sequence of events encoded into a vector of real numbers from the encoding stage and output labels. For clustering models, K-means is used for a large part of the pipelines due to its ease and the fact that the number of clusters is a hyperparameter. Moreover, reproducing [21] and [20], K-Medoids and Agglutinate Hierarchy Clustering (AHC) are used as clustering models based on their similarity measure.
Sampling Strategies are used to elect a representative user traces from each cluster to convert them later into a test suite. For the *Random Sampling* strategy, a test is sampled randomly from each cluster. In the *Best Usage Choice* sampling strategy, for every cluster, the user trace that best represents the cluster's frequent closed patterns is chosen. To do this, the computation of the relative UPC is done, which involves dividing the frequent closed pattern frequencies within the cluster by the frequent closed pattern frequencies in the overall dataset. Following TF-iDF intuition, the trace covering the frequent closed patterns frequent and specific to this cluster is searched, not the pattern frequent in all the dataset even if they are frequent inside the cluster. The downside of the random strategy is that it may end up selecting only the most frequent patterns, as they are present in all clusters. This is why a method that selects the most frequent patterns unique to each cluster, thereby representing the cluster, is preferable.

Definition 4.1 (Relative Usage Pattern Coverage)**.** Let X be a set of user traces and X_c, the traces among X belonging to a specific cluster c. Let P_X be the list of frequent closed patterns in X present above a threshold $min\_sup = \theta$, P_{X_c} the list of frequent closed patterns from P_X present in X_c. Let $Count_X(p)$ be the number of occurrences of a specific pattern p in X, $Count_{X_c}(p)$ the number of occurrences of a specific frequent closed pattern p in X_c and P_x the frequent closed pattern in a trace x. The relative UPC of a trace x of c is defined by

$$rUPC_\theta(x, X_c) = \sum_{p \in P_x} \frac{Count_{X_c}(p)}{Count_X(p)} \tag{3}$$

Definition 4.2 (Best Usage Choice)**.**

$$BUC(X_c) = \text{argmax}_{x \in X_c} rUPC(x, X_c) \tag{4}$$



(a) Teaming

(b) Booked Scheduler

Figure 6: Example of the Test Selection Coverage Function for 4 pipelines on Teaming and Booked Scheduler. The lighter area around the curve indicates the standard deviation.

All the pipelines are evaluated against a Baseline pipeline with no clustering and a pure Random Sampling strategy only.

4.4 Experimental Setup

Experiments were run on a high-performance computing enter using one node for each set of parameters (total of nodes: 1596, nodes running in parallel: 50, average node execution: 6.5 min). It took us a total of 2.5 hours. Deep learning models such as AutoEncoders, VAE, DAAE, and Transformers were tuned with a hyperparameter optimization framework. Several adaptations for [20, 21, 23] were made to apply them to this paper's use cases. For [23], the t-SNE stage was skipped for time constraints. The similarity measure of [20, 21] was adapted by taking into account only API call names and return codes instead of all parameters as it was more relevant here. For [20], a *Random Sampling* strategy was used instead of theirs, as their sampling strategy was extracting more than one test by cluster. We determine the maximum number of clusters, denoted as K_{max}, by identifying the point at which one of the 33 different pipelines achieves a UPC value of 95%.

4.5 Results

The pipelines are featured in Tables 9-10. In Table 9, the pipelines are ranked based on their normalized CAUC values for each case study. In Table 10, two metrics are displayed: the k^* for a target coverage of 80% and 90% (first criterion), and their normalized CAUC (second criterion). Pipelines are not ranked in this table. Cells which represent the lowest k^* or the higher normalized CAUC are colored in blue. An example of the Test Selection Coverage Function is featured in Figure 6. In this figure, we can see empirically the asymptotic behavior of the pipelines, and the need to find the optimal point in terms of a number of clusters. We can observe from the table 9 that certain pipelines, such as the one employing K-medoids, can emerge as the best performers, as

Rank	Teaming				Booked Scheduler				Spree				Scanner			
	ρ	ϕ	ψ	CAUC	ρ	ϕ	ψ	CAUC	ρ	ϕ	ψ	CAUC	ρ	ϕ	ψ	CAUC
1	TF+POH	Kmeans	BUC	0.836	POH	Kmeans	RS	0.83	Custom	Kmedoids	RS	0.969	TF	Kmeans	BUC	0.974
2	DAAE+POH	Kmeans	BUC	0.832	VAE+POH	Kmeans	RS	0.83	W2V+POH	Kmeans	RS	0.965	W2V	Kmeans	BUC	0.873
3	POH	Kmeans	BUC	0.832	DAAE+POH	Kmeans	RS	0.83	TF	Kmeans	BUC	0.965	POH	Kmeans	RS	0.856
4	TermFreq+POH	Kmeans	BUC	0.832	W2V+POH	Kmeans	RS	0.827	BOW+POH	Kmeans	RS	0.964	TermFreq+POH	Kmeans	RS	0.853
5	VAE+POH	Kmeans	BUC	0.831	BOW+POH	Kmeans	RS	0.822	VAE	Kmeans	RS	0.961	VAE+POH	Kmeans	RS	0.849
6	W2V+POH	Kmeans	BUC	0.83	TermFreq+POH	Kmeans	RS	0.821	POH	Kmeans	RS	0.961	DAAE+POH	Kmeans	RS	0.849
7	BOW+POH	Kmeans	BUC	0.816	BOW	Kmeans	RS	0.805	DAAE+POH	Kmeans	RS	0.96	AE+POH	Kmeans	RS	0.849
8	TF	Kmeans	BUC	0.788	AE+POH	Kmeans	RS	0.805	VAE+POH	Kmeans	RS	0.96	BOW+POH	Kmeans	RS	0.846
9	VAE+POH	Kmeans	RS	0.784	TermFreq+POH	Kmeans	BUC	0.8	AE	Kmeans	RS	0.959	DAAE	Kmeans	RS	0.84
10	DAAE+POH	Kmeans	RS	0.784	POH	Kmeans	BUC	0.8	TermFreq+POH	Kmeans	RS	0.959	TF+POH	Kmeans	BUC	0.837
11	TF+POH	Kmeans	RS	0.783	DAAE+POH	Kmeans	BUC	0.795	TF+POH	Kmeans	RS	0.958	TF+POH	Kmeans	RS	0.835
12	TermFreq+POH	Kmeans	RS	0.782	VAE+POH	Kmeans	BUC	0.793	AE+POH	Kmeans	RS	0.953	TermFreq+POH	Kmeans	BUC	0.834
13	POH	Kmeans	RS	0.782	W2V+POH	Kmeans	BUC	0.781	W2V	Kmeans	RS	0.951	BOW+POH	Kmeans	BUC	0.834
14	W2V+POH	Kmeans	RS	0.779	BOW	Kmeans	BUC	0.78	DAAE	Kmeans	RS	0.947	VAE+POH	Kmeans	BUC	0.834
15	W2V	Kmeans	BUC	0.76	BOW+POH	Kmeans	BUC	0.776	DAAE	Kmeans	BUC	0.943	POH	Kmeans	BUC	0.834
16	BOW	Kmeans	BUC	0.759	AE+POH	Kmeans	BUC	0.768	BOW	Kmeans	RS	0.94	DAAE+POH	Kmeans	BUC	0.834
17	AE+POH	Kmeans	RS	0.759	TF+POH	Kmeans	RS	0.73	TF	Kmeans	RS	0.94	W2V+POH	Kmeans	BUC	0.834
18	BOW+POH	Kmeans	RS	0.758	TF+POH	Kmeans	BUC	0.69	AE	Kmeans	BUC	0.937	AE+POH	Kmeans	BUC	0.834
19	AE+POH	Kmeans	BUC	0.757	Custom	Kmedoids	RS	0.685	VAE	Kmeans	BUC	0.937	W2V+POH	Kmeans	RS	0.833
20	TF	Kmeans	RS	0.735	TF	Kmeans	BUC	0.62	Custom	AHC	RS	0.936	W2V	Kmeans	RS	0.832
21	VAE	Kmeans	RS	0.725	TF	Kmeans	RS	0.619	TermFreq	Kmeans	RS	0.933	TF	Kmeans	RS	0.831
22	VAE	Kmeans	BUC	0.722	AE	Kmeans	RS	0.608	DAAE+POH	Kmeans	BUC	0.925	Custom	Kmedoids	RS	0.83
23	AE	Kmeans	RS	0.713	AE	Kmeans	BUC	0.6	TF+POH	Kmeans	BUC	0.92	TermFreq	Kmeans	RS	0.806
24	TermFreq	Kmeans	BUC	0.706	DAAE	Kmeans	RS	0.598	TermFreq	Kmeans	BUC	0.917	VAE	Kmeans	RS	0.802
25	DAAE	Kmeans	RS	0.697	W2V	Kmeans	BUC	0.571	BOW+POH	Kmeans	BUC	0.915	VAE	Kmeans	BUC	0.801
26	BOW	Kmeans	RS	0.689	W2V	Kmeans	RS	0.539	BOW	Kmeans	BUC	0.911	Custom	AHC	RS	0.788
27	DAAE	Kmeans	BUC	0.689	VAE	Kmeans	RS	0.528	W2V	Kmeans	BUC	0.896	AE	Kmeans	RS	0.783
28	TermFreq	Kmeans	RS	0.675	DAAE	Kmeans	BUC	0.505	AE+POH	Kmeans	BUC	0.886	BOW	Kmeans	RS	0.774
29	AE	Kmeans	BUC	0.663	TermFreq	Kmeans	RS	0.498	VAE+POH	Kmeans	BUC	0.872	AE	Kmeans	BUC	0.737
30	W2V	Kmeans	RS	0.652	-	Baseline	RS	0.327	W2V+POH	Kmeans	BUC	0.872	DAAE	Kmeans	BUC	0.719
31	Custom	Kmedoids	RS	0.642	TermFreq	Kmeans	BUC	0.32	TermFreq+POH	Kmeans	BUC	0.872	BOW	Kmeans	BUC	0.697
32	Custom	AHC	RS	0.604	VAE	Kmeans	BUC	0.302	POH	Kmeans	BUC	0.872	TermFreq	Kmeans	BUC	0.696
33	-	Baseline	RS	0.497	Custom	AHC	RS	0.25	-	Baseline	RS	0.581	-	Baseline	RS	0.46

Table 9: Ranked pipelines with respect to the Normalized CAUC for the 4 case studies

seen in the case of Spree, while nearly being the worst performers in the case of Teaming. In Table 10, it is apparent that certain pipelines exhibit identical performance based on the first criterion for selecting the best k^* value in the case of Spree. However, the CAUC criterion allows us to distinguish between them. Additionally, the table reveals that achieving a UPC increase from 80% to 90% in the case of Teaming requires nearly doubling the number of tests. Notably, like the baseline each time, some pipelines fail to attain the target coverage within the specified budget.

5 Analysis and Discussions

The upcoming section provides an in-depth analysis of the research outcomes, comparing findings with the current state of the art, addressing the research questions, and assessing potential threats to the validity of our work.

5.1 Analysis of results

Baseline is beaten

The large majority of clustering-based test selection pipelines consistently outperform the Baseline pipeline with respect to the mean of the normalized CAUC over the four use cases which confirms that clustering is suitable rather than a pure random sampling approach. In three out of four datasets, one clustering pipeline is demonstrated to outperform existing state-of-the-art methods.

In sampling strategies, not all are equal

The performance of the various sampling strategies is not equivalent at all. On Scanner and Teaming, BUC seems to lead to better performance whereas on Booked Scheduler and Spree, Random sampling

ρ	ϕ	ψ	Teaming			Booked Scheduler			Spree			Scanner		
			CAUC	# Clusters UPC >80%	#Clusters UPC>90%	CAUC	# Clusters UPC >80%	# Clusters UPC >90%	CAUC	# Clusters UPC >80%	# Clusters UPC >90%	CAUC	# Clusters UPC >80%	# Clusters UPC >90%
AE	Kmeans	BUC	0.663	∞	∞	0.6	**5**	**6**	0.937	**2**	3	0.737	5	6
AE+POH	Kmeans	BUC	0.757	16	22	0.768	6	7	0.886	**2**	4	0.834	3	3
BOW	Kmeans	BUC	0.759	16	∞	0.78	**5**	9	0.911	**2**	3	0.697	5	5
BOW+POH	Kmeans	BUC	0.816	11	22	0.776	6	7	0.915	**2**	3	0.834	3	3
DAAE	Kmeans	BUC	0.689	19	∞	0.505	9	10	0.943	**2**	3	0.719	6	6
DAAE+POH	Kmeans	BUC	0.832	11	18	0.795	6	**6**	0.925	**2**	3	0.834	3	3
POH	Kmeans	BUC	0.832	11	19	0.8	6	**6**	0.872	**2**	4	0.834	3	3
TF	Kmeans	BUC	0.788	11	24	0.62	10	14	0.965	**2**	3	**0.974**	**2**	**2**
TF+POH	Kmeans	BUC	**0.836**	**9**	**17**	0.69	7	9	0.92	**2**	3	0.837	3	3
TermFreq	Kmeans	BUC	0.706	20	∞	0.32	∞	∞	0.917	**2**	3	0.696	4	4
TermFreq+POH	Kmeans	BUC	0.832	11	19	0.8	6	**6**	0.872	**2**	4	0.834	3	3
VAE	Kmeans	BUC	0.722	18	24	0.302	12	∞	0.937	**2**	3	0.801	3	3
VAE+POH	Kmeans	BUC	0.831	10	19	0.793	6	**6**	0.872	**2**	∞	0.834	3	3
W2V	Kmeans	BUC	0.76	14	∞	0.571	10	13	0.896	**2**	3	0.873	**2**	**2**
W2V+POH	Kmeans	BUC	0.83	11	19	0.781	6	**6**	0.872	**2**	∞	0.834	3	3
AE	Kmeans	RS	0.713	19	∞	0.608	9	14	0.959	**2**	**2**	0.783	3	5
AE+POH	Kmeans	RS	0.759	14	23	0.805	**5**	7	0.953	**2**	**2**	0.849	3	3
BOW	Kmeans	RS	0.689	24	∞	0.805	**5**	7	0.94	**2**	3	0.774	4	5
BOW+POH	Kmeans	RS	0.758	15	24	0.822	**5**	7	0.964	**2**	**2**	0.846	3	3
DAAE	Kmeans	RS	0.697	18	∞	0.598	10	∞	0.947	**2**	**2**	0.84	3	3
DAAE+POH	Kmeans	RS	0.784	13	20	0.83	**5**	7	0.96	**2**	**2**	0.849	3	3
POH	Kmeans	RS	0.782	14	20	**0.83**	**5**	**6**	0.961	**2**	**2**	0.856	3	3
TF	Kmeans	RS	0.735	17	∞	0.619	10	14	0.94	**2**	3	0.831	3	4
TF+POH	Kmeans	RS	0.783	13	20	0.73	7	10	0.958	**2**	**2**	0.835	3	3
TermFreq	Kmeans	RS	0.675	23	∞	0.498	12	∞	0.933	**2**	3	0.806	3	4
TermFreq+POH	Kmeans	RS	0.782	14	20	0.821	**5**	7	0.959	**2**	**2**	0.853	3	3
VAE	Kmeans	RS	0.725	19	∞	0.528	12	∞	0.961	**2**	**2**	0.802	3	4
VAE+POH	Kmeans	RS	0.784	13	20	0.83	**5**	7	0.96	**2**	**2**	0.849	3	3
W2V	Kmeans	RS	0.652	24	∞	0.539	12	∞	0.951	**2**	3	0.832	3	4
W2V+POH	Kmeans	RS	0.779	14	20	0.827	**5**	**6**	0.965	**2**	**2**	0.833	3	3
Custom	Kmedoids	RS	0.642	∞	∞	0.685	7	10	**0.969**	**2**	**2**	0.83	3	4
Custom	AHC	RS	0.604	∞	∞	0.25	∞	∞	0.936	**2**	3	0.788	3	∞
-	Baseline	RS	0.497	∞	∞	0.327	∞	∞	0.581	∞	∞	0.46	∞	∞

Table 10: Normalized CAUC and k^* for all the variations of pipelines for the 4 case studies

is better.

Pattern One Hot Encoding increases a lot the performance of the pipelines

Pattern One Hot encoding explicitly encodes the presence or absence of each pattern, thus directly aligning with the UPC metric. For this reason, this approach consistently ensures a fair performance.

Neural Embeddings can help increase the performance of Pattern One Hot Encoding

While Pattern One-Hot Encoding is already an effective approach for capturing usage patterns and optimizing test selection pipelines, our research demonstrates that it can be further improved by integrating neural embeddings. This combination enhances the representation of user traces, leading to more accurate results.

Clustering-based test selection pipelines are more efficient than human-written test suite

For scanner, the pipeline TF manages to obtain more than 90% of UPC with 2 tests, while the human written test suite reaches 81.8% in 27 tests. For teaming, the pipeline TF+POH reaches more than 99.9% of UPC with 102 tests, while the human-written test suite contains 236 tests and achieves 99.6 % of UPC. Moreover, to compare the two suites of tests with a lower target coverage and a limited budget of 17 tests, 17 tests were taken randomly among the human-written test suite and their UPC was computed. This process is repeated 10,000 times and a maximum UPC of 83% and a mean UPC of 61% is obtained, whereas the pipeline TF+POH achieve more than 90% of UPC with the same budget.

Clustering-based test selection pipelines perform differently depending on the datasets

Given the unique characteristics of trace datasets, including variations in vocabulary, long-term dependencies, and token repetitions, there is no universally superior clustering model. Consequently, it is advisable for Quality Assurance engineers to benchmark clustering-based test selection pipelines when faced with new use cases, recognizing that each dataset may yield distinct performance outcomes.

Metric Consistency with Results

The metric UPC and the derivated criteria unequivocally highlight the baseline as the weakest performer. Furthermore, it discerns that the optimal number of clusters tends to be lower for our simpler datasets, such as artificial user traces, and higher for more intricate, real-world use cases. Notably, the metric consistently yields scores above 80% for human-generated test suites which seems realistic. In summary, the metric consistency and its alignment with high human-generated test suite scores reinforce the metric's reliability and suitability for assessing clustering pipelines

5.2 Comparison with state of the art

In comparison to the state of the art, we assessed regression test selection using clustering with an external metric based on UPC, as opposed to the conventional metrics like APFD, block or function coverage, or internal metrics commonly employed in the state of the art, which are less suitable for benchmarking clustering pipelines. We delved into a critical aspect of this subject, which is determining the optimal number of clusters, a topic not typically addressed in the state of the art. Furthermore, we conducted benchmarking across various encoding methods, including neural embeddings, an approach that has seen limited exploration in prior research. Additionally, we investigated whether the performance of the extracted test suites was influenced by the algorithm used to select tests within the clusters.

5.3 Answers to research questions

The relevance of our approach is verified through experimental validation at three distinct levels: first, on a controlled application (Scanner example); second, on a large open-source eCommerce software, simulating user interactions with artificial users (Spree); and finally, on two industry cases study involving real users (Teaming and Booked).

RQ1: What is the representativeness of a test suite with respect to the actual usage? The coverage of user paths is translated into a statistical metric ranging from 0 to 100%, signifying the quality of a test suite in relation to user traces, with 100% indicating excellence. This methodology enables to address RQ1. **RQ2: Can clustering of user sessions help to segment representative usage in an optimal number of clusters?** The UPC metric permits to introduce and study the Test Selection Coverage Function associated with each pipeline and to find the optimal point where a target coverage is reached. This method works just like elbow method but in a specific way meeting our software requirements explicitly. Thanks to UPC, more efficient test

suites than human-crafted ones have been found. This allows us to answer RQ2. **RQ3: To what extent do clustering pipelines produce different results?** The application to the 4 datasets with 33 variations of clustering pipelines leads to various performances of each pipeline. Neural Embeddings and BUC permit us to reach the best results on 2 datasets whereas simpler methods perform better on the 2 others. This reinforces our view a benchmark is needed every time a new use case is introduced and answers RQ3.

5.4 Threats to validity

The ideal minimum support chosen for the UPC has not been addressed in this paper. However, the threshold chosen enabled to show that Clustering-based Test Selection Pipelines were producing test suites with fewer tests and better UPC than human ones, as with the threshold chosen, human test suites having a UPC of less than 100%. In this paper, it was deliberately chosen to extract a single test per cluster. Other approaches, such as extracting multiple tests per cluster, could be considered. In this paper's scenario, the decision was made to compel the clustering model to increasingly separate the dataset to achieve highly disjointed partitions, enabling the extraction of a test suite that fulfills our coverage objectives. Moreover, the benchmark focuses on a specific metric, and results could differ with metrics like APFD. However, the chosen measure aligns with regression testing inspired by usage, and the benchmarking framework requires no heavy instrumentation, ensuring practical applicability.

5.5 Discussion on Practical Implications: Real-world Application of Regression Testing Processes

With the prevalence of log collection in contemporary systems, accessing logs for regression testing purposes is typically straightforward, barring privacy concerns such as GDPR. The primary challenge in creating new regression tests lies in deriving user traces selected as potential tests, a process that remains largely manual and not yet automated. Nonetheless, this methodology can provide testers with valuable insights into which tests need implementation, their quantity, and the relevance of existing tests.

6 Conclusion and perspective

6.1 Conclusion

This article addresses the critical challenge of employing clustering techniques for regression test generation, recognizing that the choice of clustering pipeline plays a pivotal role in achieving optimal results tailored to each unique case study. To facilitate this decision-making process, we have introduced a novel and robust statistical coverage metric based on pattern mining. This metric, applicable to both front-end and back-end traces, offers a comprehensive evaluation of the relevance of a regression test suite in relation to user traces of the SUT.Our approach leverages this metric to identify the optimal number of clusters at which a clustering pipeline achieves an optimal

UPC. This process of studying the Test Selection Coverage Function, mirrors the Elbow method, and allows us to fine-tune and benchmark clustering pipelines effectively. To meet the diverse requirements of testers, we have introduced two distinct criteria for evaluating clustering pipelines: one focusing on the overall coverage attained and the other prioritizing the ratio of the number of tests to the target coverage.Through extensive experimentation involving 33 variations of clustering based test selection pipelines across four diverse datasets from different web applications, we have systematically ranked these pipelines based on their performance. The results demonstrate that the best-performing pipelines outperform human-written test suites, highlighting the efficacy of our approach. In summary, our work underscores the importance of selecting the most suitable clustering model for regression test generation and provides a robust framework for evaluating and benchmarking clustering pipelines. The introduction of our novel UPC metric and the insights gained from our experiments contribute to advancing the state-of-the-art in automated test generation, bridging the gap between written tests and actual user behavior. Moreover, we make three out of four datasets and code publicly available.

6.2 Perspectives

In the future, it will be necessary to investigate the factors contributing to the varying performances of different clustering pipelines on specific datasets. This research aims to understand why certain pipelines excel in particular scenarios and identify dataset characteristics that help prioritize specific pipelines. Moreover, while we used only two sampling methods in this study, future research could explore test selection methods by clusters that perform well across diverse datasets.

References

1. Blanc X, Degueule T, and Falleri JR. Diffing E2E Tests against User Traces for Continuous Improvement. 2022.
2. Tiwari D, Zhang L, Monperrus M, and Baudry B. Production monitoring to improve test suites. IEEE Transactions on Reliability 2021.
3. Thummalapenta S, De Halleux J, Tillmann N, and Wadsworth S. DyGen: Automatic generation of high-coverage tests via mining gigabytes of dynamic traces. In: *International Conference on Tests and Proofs*. Springer. 2010:77–93.
4. Chen S, Chen Z, Zhao Z, Xu B, and Feng Y. Using semi-supervised clustering to improve regression test selection techniques. In: *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. 2011:1–10. DOI: 10.1109/ICST.2011.38.
5. Kandil P, Moussa S, and Badr N. Cluster-based Test Cases Prioritization and Selection Technique for Agile Regression Testing. Journal of Software: Evolution and Process 2016;29.
6. Almaghairbe R and Roper M. Separating passing and failing test executions by clustering anomalies. Software Quality Journal 2017.

7. Carlson R, Do H, and Denton A. A clustering approach to improving test case prioritization: An industrial case study. In: *27th IEEE International Conference on Software Maintenance (ICSM)*. 2011:382–91. DOI: 10.1109/ICSM.2011.6080805.

8. Xu J, Wang P, Tian G, et al. Short Text Clustering via Convolutional Neural Networks. In: *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*. Denver, Colorado: Association for Computational Linguistics, 2015:62–9. DOI: 10.3115/v1/W15-1509. URL: https://aclanthology.org/W15-1509.

9. Xu J, Xu B, Wang P, et al. Self-Taught convolutional neural networks for short text clustering. Neural Networks 2017;88:22–31.

10. Rosenberg A and Hirschberg J. V-measure: A conditional entropy-based external cluster evaluation measure. In: *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*. 2007:410–20.

11. Rendón E, Abundez I, Arizmendi A, and Quiroz EM. Internal versus external cluster validation indexes. International Journal of computers and communications 2011;5:27–34.

12. Singh AK, Mittal S, Malhotra P, and Srivastava YV. Clustering Evaluation by Davies-Bouldin Index (DBI) in Cereal data using K-Means. In: *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE. 2020:306–10.

13. Hosseini SMS, Maleki A, and Gholamian MR. Cluster analysis using data mining approach to develop CRM methodology to assess the customer loyalty. Expert Systems with Applications 2010;37:5259–64.

14. Dinh DT, Fujinami T, and Huynh VN. Estimating the Optimal Number of Clusters in Categorical Data Clustering by Silhouette Coefficient. In: *Knowledge and Systems Sciences*. Ed. by Chen J, Huynh VN, Nguyen GN, and Tang X. Singapore: Springer Singapore, 2019:1–17.

15. Zhou HB and Gao JT. Automatic method for determining cluster number based on silhouette coefficient. In: *Advanced materials research*. Vol. 951. Trans Tech Publ. 2014:227–30.

16. Tamagnan F, Bouquet F, Vernotte A, and Legeard B. Regression Test Generation by Usage Coverage Driven Clustering on User Traces. In: *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 2023:82–9. DOI: 10.1109/ICSTW58534.2023.00026.

17. Ahmed MH, Tiun S, Omar N, and Sani NS. Short text clustering algorithms, application and challenges: A survey. Applied Sciences 2022;13:342.

18. Fournier-Viger P, Lin JCW, Kiran RU, Koh YS, and Thomas R. A survey of sequential pattern mining. Data Science and Pattern Recognition 2017;1:54–77.

19. Luo X, Ping F, and Chen MH. Clustering and Tailoring User Session Data for Testing Web Applications. In: *2009 International Conference on Software Testing Verification and Validation*. 2009:336–45. DOI: 10.1109/ICST.2009.51.

20. Liu Y, Wang K, Wei W, Zhang B, and Zhong H. User-Session-Based Test Cases Optimization Method Based on Agglutinate Hierarchy Clustering. In: *ITHINGSCPSCOM'11, International Conference on Internet of Things*. 2011:413–8. DOI: 10.1109/iThings/CPSCom.2011.135.

21. Li Jh and Xing Dd. User Session Data based Web Applications Test with Cluster Analysis. In: *CSIE 2011, International Conference on Computer Science and Information Engineering*. Vol. 152. 2011:415–21. DOI: 10.1007/978-3-642-21402-866.

22. Dorcis V, Bouquet F, and Dadeau F. Clustering of Usage Traces for Regression Test Cases Selection. In: *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. 2022:138–45.

23. Afshinpour B, Groz R, Amini MR, Ledru Y, and Oriat C. Reducing Regression Test Suites using the Word2Vec Natural Language Processing Tool. In: *SEED/NLPaSE@ APSEC*. 2020:43–53.

24. Utting M, Legeard B, Dadeau F, Tamagnan F, and Bouquet F. Identifying and generating missing tests using machine learning on execution traces. In: *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE. 2020:83–90.

25. Stocco A, Willi A, Starace LLL, Biagiola M, and Tonella P. Neural Embeddings for Web Testing. 2023. arXiv: 2306.07400 [cs.SE].

26. Jabbar E, Zangeneh S, Hemmati H, and Feldt R. Test2Vec: An Execution Trace Embedding for Test Case Prioritization. 2022. arXiv: 2206.15428 [cs.SE].

27. Feng Z, Guo D, Tang D, et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. 2020. arXiv: 2002.08155 [cs.CL].

28. Nedelkoski S, Bogatinovski J, Acker A, Cardoso J, and Kao O. Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs. 2020. arXiv: 2008.09340 [cs.LG].

29. Golczynski A and Emanuello JA. End-To-End Anomaly Detection for Identifying Malicious Cyber Behavior through NLP-Based Log Embeddings. 2021. arXiv: 2108.12276 [cs.AI].

30. Gomariz A, Campos M, Marin R, and Goethals B. ClaSP: An Efficient Algorithm for Mining Frequent Closed Sequences. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Pei J, Tseng VS, Cao L, Motoda H, and Xu G. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013:50–61.

31. Fournier-Viger P, Lin CW, Gomariz A, et al. The SPMF Open-Source Data Mining Library Version 2. In: *Proc. 19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III*. Vol. 9853. Springer LNCS. 2016:36–40.

32. Bradley AP. The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognition 1997;30:1145–59.

33. Vaswani A, Shazeer N, Parmar N, et al. Attention Is All You Need. 2017. arXiv: 1706.03762 [cs.CL].

34. Shen T, Mueller J, Barzilay R, and Jaakkola T. Educating Text Autoencoders: Latent Representation Guidance via Denoising. 2020. arXiv: 1905.12777 [cs.LG].