Application of Deep Learning on the Task of Near-Duplicate Detection for Web Testing

Luca Kollmer

Seminar: Software Quality
Advisor: Prof. Dr. Andrea Stocco
Technical University of Munich
luca.kollmer@tum.de

Abstract. Automated web crawlers play a crucial role in End-to-End testing of web applications for generating *state*-based models, where states represent distinct functionalities and features of the application. However, a challenge in existing web crawling techniques is the inclusion of near-duplicate states, marginally different pages representing the same functionality. Near-duplicates adversely impact model-based testing tasks like test case generation. This study introduces a novel approach for near-duplicate detection in web application model inference, using a fine-tuned transformer-based deep learning model, specifically distilBERT, to enhance the accuracy of near-duplicate detection.
The research evaluates DistilBERT in two configurations: its raw embeddings and a fine-tuned version. We initially employ the raw distilBERT model to generate embeddings of web pages which are used to classify state-pairs as distinct or (near-)duplicated. Subsequently, we refine the model through fine-tuning on the task of near-duplicate detection. Our analysis compares both methods in terms of classification performance (F_1-Score) and inference time on a dataset of web page state-pairs.
The results show that the fine-tuned DistilBERT surpasses the raw embeddings in classification accuracy, achieving a F_1-Score of 0.97 compared to 0.73. Furthermore, the approach results in a +10% to +17% improvement in F_1-Score w.r.t. previous near-duplicate detection techniques. However, it is observed that the fine-tuned model, despite its superior accuracy, operates slower than existing state-of-the-art near-duplicate detection algorithms. Despite this, the fine-tuned DistilBERT's improved accuracy and efficiency indicate its potential for enhancing test generation in web applications, suggesting a move towards more robust and efficient model-based testing strategies.

Keywords: Near-Duplicate Detection · Web Testing · distilBERT.

1 Introduction

Web applications, central to modern digital experiences, underpin diverse facets of our lives, ranging from communication and leisure to education and business. Offering unparalleled convenience and accessibility, they reshape the way we interact, work, and engage with the world. This widespread digital integration necessitates thorough testing to ensure their quality and functionality. Testing web applications poses unique challenges, including high complexity and the need for compatibility across various devices and browsers [3]. End-to-End (E2E) Testing, a prevalent approach, simulates user actions such as clicks, scrolls, or text input. While manual E2E Testing is time-consuming and error-prone [19], automated E2E Testing, using tools like Selenium [21], has significantly improved efficiency by automating test case execution [18].

However, a persistent challenge in automated testing is the creation of the test cases. Depending on the application size, test development can become labor- and resource-intensive [1]. The approach of *Web Test Generation* aims to address this by generating tests automatically, thereby reducing manual effort. To this end, web crawlers dynamically explore the Application

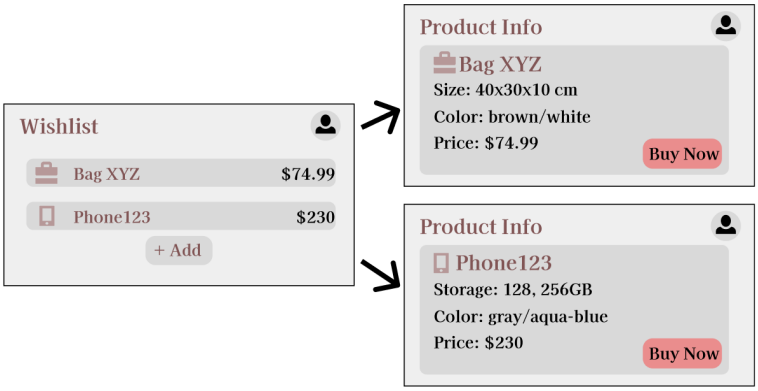

Fig. 1. Example of Near-Duplicates (Right)

Under Test (AUT) through interactions with User Interface (UI) elements, such as buttons. In the context of automated test generation, a crawler's goal is to construct a *concise model* of the web application [14]. This model consists of *states* that represent different aspects of the web application, such as the Login Page [23]. A concise model includes only relevant states and avoids clones or near-duplicates. While clones are self-explanatory and relatively easy to detect [12], *near-duplicate* states, characterized by marginal differences in specific segments of two webpages, pose a greater challenge [23]. An example of a near-duplicate state-pair in an E-commerce application is illustrated in *Figure 1*: During runtime, a crawler exploring the "Wishlist" page may encounter different product pages that represent new functional states or near-duplicate ones, which do not provide new functionality.

From a web test generation viewpoint, including duplicates could render the generated test suites ineffective due to the existence of redundant test cases that do not increase code coverage [23]. A study found that various state-of-the-art duplicate detection algorithms, employing different approaches (e.g., textual content, DOM-tree, visual screenshot), include near-duplicate states in their generated web-app models [23]. Thus, the main task revolves around stable and reliable web app model inference, particularly in near-duplicate detection *within* a web-app: deciding whether two crawled states of the AUT are distinct or near-duplicates/clones, and if they should be included in the web-app model. Addressing this challenge necessitates innovative solutions capable of processing and understanding complex web content with high accuracy.

The recent advancements in Deep Learning (DL) offer promising avenues in this regard. DL has seen a surge in interest due to its performance enhancements across a range of applications, particularly in Natural Language Processing (NLP). DL models have revolutionized how machines understand human language, with significant developments in architectures such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and, more recently, Transformers. These models have set new benchmarks in various NLP tasks. The introduction of models like BERT and GPT, pre-trained on vast corpora and fine-tuned for specific tasks, has demonstrated remarkable understanding and generation capabilities [16].

A recent approach by Stocco et al. [22] utilizes neural network embeddings produced by Doc2Vec [10] to enhance near-duplicate detection. Their approach outperforms existing near-duplication methods [22]. However, the field may benefit further from exploring the capabilities of deep neural networks, particularly those based on recent advancements in transformer architectures. These networks offer a more complex and nuanced understanding of language and context [2]. This work investigates whether these advanced deep learning models can outperform current state-of-the-art methods, such as the one proposed by Stocco et al. [22], in detecting near-duplicates. By leveraging the sophisticated linguistic and contextual processing power of

deep neural networks, we hypothesize that it is possible to achieve even higher accuracy and efficiency in identifying near-duplicate web pages.

In this context, we propose a novel approach utilizing *distilBERT* [20], a streamlined version of the BERT model known for its efficient language understanding [2]. We initially assess the unmodified distilBERT model, achieving a preliminary accuracy of 77% and a F_1-Score of 0.73. Subsequently, we enhance the model's performance by fine-tuning it for the task of near-duplicate detection, resulting in a notable improvement with an accuracy of 97% and a F_1-Score of 0.97. Additionally, we evaluate the average inference time to determine the feasibility of implementing these methods in real-time web crawlers. The fine-tuned model, with an average inference time of 0.92 seconds per state-pair, is faster than the unmodified model, which averages 1.15 seconds per state-pair. However, despite being faster than the initial approach, the fine-tuned model still demonstrates slower processing times compared to existing techniques. Depending on the benchmark used, our approach is up to 16 times slower in inference. Despite the trade-off between speed and accuracy, integrating this approach into real-time web crawlers promises significant advancements in near-duplicate detection.

2 Related Work

2.1 Web Crawling and Duplicate Detection

In the past, extensive research has focused on detecting similarity in webpages across the internet, for example, in the indexing of web pages for search engines [5,7,8,12]. These studies concentrated on near-duplicates and the similarity of web pages *across* different applications in the context of web browsers.

This paper, however, shifts the focus to exploring webpage similarity *within* single web applications, particularly in the context of web testing. We leverage findings from existing studies and approaches for near-duplicate detection in this specific domain to inform our research.

Yandrapally et al. [23] classify different types of near-duplicates and study several near-duplicate detection techniques in the context of web crawling. They conclude that the evaluated near-duplicate detection algorithms, which use universal thresholds, fail to reliably detect certain types of near-duplicates, indicating a need for improved methods. Building on their findings, our study demonstrates that neural embeddings produced by a pre-trained transformer-based Deep Learning (DL) model do not improve the process of inferring an optimal threshold for near-duplicate detection across various applications. To overcome the limitations of using an universal threshold, we introduce a novel technique that fine-tunes a DL model specifically for near-duplicate detection. Additionally, Yandrapally et al. [23] introduce a dataset comprising approximately 100,000 labeled state-pairs from nine open-source web-apps. We use this dataset for training and evaluating our approaches and employ the most effective algorithms identified in their study as benchmarks.

Stocco et al. [22] present a threshold-free classifier trained on neural network embeddings for detecting near-duplicate web pages. They utilize a modified version of the Doc2Vec [10] model, tailored for HTML web pages, to generate embeddings. Their method is evaluated against two baselines, the best performing algorithms discussed in Yandrapally et al. [23], and notably surpasses these baselines, demonstrating the potential of neural networks in capturing the semantics of web pages [22]. Our research extends the exploration into the viability of neural networks for near-duplicate detection, focusing specifically on evaluating the performance of pre-trained deep learning models, such as DistilBERT [20], in this task. Additionally, their study utilizes the same dataset introduced by Yandrapally et al. [23], enabling a direct comparison in the context of near-duplicate detection.

2.2 Deep Learning on HTML

Over the last decade, Deep Learning has experienced a significant surge in interest, primarily due to its substantial impact across various technical and scientific domains [6,11]. In Deep Learning, computational models with multiple processing layers learn data representations at a high level of abstraction [11]. Natural Language Processing (NLP), a subfield of Deep Learning, focuses on learning the structure of language. In this work, DL models, originally pre-trained for NLP, are leveraged to learn representations of HTML for the task of near-duplicate detection. Literature on applying deep learning to HTML is limited.

Opara et al. [15] propose a phishing detection method using Convolutional Neural Networks (CNNs) to analyze semantic relationships within HTML text elements. Their results show that this approach effectively learns context features of HTML, yielding satisfactory outcomes in Phishing Detection. Although our work focuses on transformer-based architectures for near-duplicate detection, Opara et al.'s findings [15] provide a useful comparative perspective, indicating that deep learning techniques applied to HTML content analysis have significant potential for near-duplicate detection.

3 Approach

3.1 Classification of Web Page Relationships

This section elaborates on the specific labels (Classes) used to describe the relationship between two states in a state-pair, followed by a comprehensive discussion on the classification tasks these labels pertain to.

Classes A state-pair of webpages within an application can be categorized into three classes: *Distinct*, *Clone*, and *Near-Duplicate*. A visualization is presented in Figure 2.

Distinct refers to pairs of states within an Application Under Test (AUT) that demonstrate fundamentally different functionalities. For example, consider the Login Page and the User Review Page of a web application. These pages perform entirely separate functions and, therefore, should be individually represented in the web application model.

Clone describes pairs of states in a web application whose underlying HTML pages are identical. During a web crawler's runtime, let p_A be the HTML content of the initial page. An interaction with a UI element triggers the loading of a new page or state, p_B, also represented in HTML. If p_B is identical to p_A following the GUI event, this state-pair is labeled as a clone. An example of such a cloned state-pair could occur when p_A is the home page of the AUT, and the crawler clicks on the AUT's icon, typically located in the header. This action often redirects to the home page, which was already included in the model through p_A, rendering the "new" state p_B equal to the initial p_A. In the context of building a web application model for test generation, p_A should be included as it represents a distinct functional state. In contrast, p_B should be omitted, as it does not offer additional functionality. Standard checksumming techniques can efficiently detect such cloned states, as detailed in existing research [12].

Near-Duplicate refers to a state-pair of two webpages that represent the same logical state but exhibit marginal differences in specific segments of the sites [23]. An illustrative example of near-duplicate states in an e-commerce application is depicted in Figure 1. Suppose the "Wishlist" page, listing two products, is crawled. Clicking on the first product leads to a new state, p_C (shown in the upper right of the figure), displaying the product info for this first product. p_C represents an unseen functional state and is thus included in the final web application model. However, when navigating back to the "Wishlist" page and clicking on the second product in the list, the crawler obtains state p_D, the product info for the second product, which is conceptually the same page as state p_C. As such, the state-pair of p_C and p_D is labeled as near-duplicate,

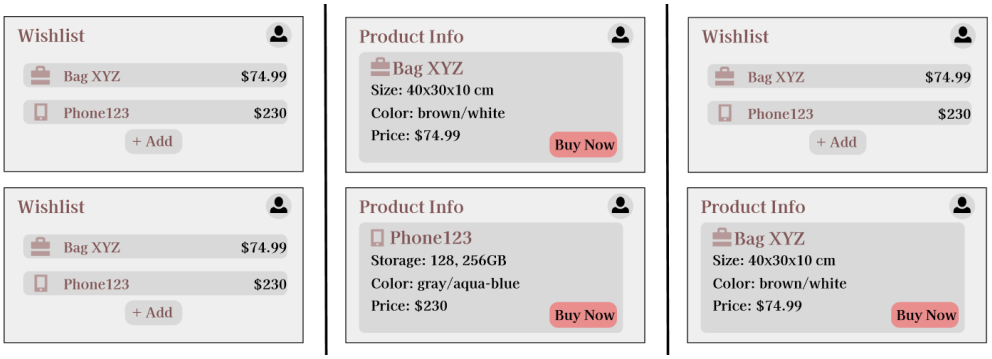

Fig. 2. Different Classes of State-Pairs: Clone (left), Near-Duplicate (middle), Distinct (right)

and p_D should not be included in the final model of the web application, as it can lead to redundant test cases in the context of web test generation.

Classification Tasks The main task of near-duplicate detection can be framed as either a binary classification task or a multiclass classification task, each serving different purposes and offering insights from various perspectives:

Binary Classification Task: Predict whether a given state-pair of web pages is best described as *logically similar*, i.e., clone or near-duplicate (1), or *distinct* (0). This task is crucial from a web test generation perspective, where the primary concern for crawlers is to avoid including near-duplicates and clones in the inferred model. The inclusion of functional duplicates renders the generated test suites ineffective due to the existence of redundant test cases that do not increase code coverage [23].

Multiclass Classification Task: Adopting a more nuanced classification of web pages into the categories *clone* (0), *near-duplicate* (1), and *distinct* (2) shifts the approach from binary to multiclass classification. This refined method is more informative from a content analysis perspective, offering deeper insights into the nature and extent of duplication.

This paper focuses on the binary classification task to delve into the core challenges of near-duplicate detection within the scope of runtime web crawling. By evaluating the binary classification task, which discerns between similar (clone or near-duplicate) and distinct state-pairs, the study aims to directly address the principal concerns crucial to a crawler's efficiency.

3.2 Choosing a Pre-Trained DL Model

Transformer-based models represent a significant advancement in natural language processing. For a comprehensive understanding of transformers, readers are encouraged to explore the research by Khan et al. [9]. This study provides an extensive overview of transformer models, highlighting their efficiencies and innovations. We will assess their capabilities for complex tasks like near-duplicate detection in HTML pages. We hypothesize that such models can understand and process the intricate structure and content of HTML and thus can reliably detect logical similarity in two webpages. Among these transformer-based models, *Bidirectional Encoder Representations from Transformers (BERT)* based models stand out and will be utilized in this work. The selection of a BERT-based model is influenced by its robust transfer learning capabilities, state-of-the-art performance across various NLP benchmarks, and the simplicity of fine-tuning a pre-trained BERT model [2]. The widespread availability of pre-trained models and the extensive community support further make BERT-based models a compelling choice for our research.

More specifically, we employ *DistilBERT*, a distilled version of the larger BERT model, which retains most of the BERT model's performance while being 40% smaller and 60% faster [20]. This efficiency makes it more practical for processing large sets of HTML document representations at the runtime of a crawler, where computational resources and time are important factors. Additionally, DistilBERT can be fine-tuned on a smaller dataset while still leveraging the broad knowledge it has gained during pre-training. This is particularly beneficial for the task of near-duplicate detection, where labeled data in the form of state-pairs of web pages categorized as clone, near-duplicate, and distinct is limited and costly to obtain, as there is no automated method for it [23]. Specifically, *uncased* DistilBERT is leveraged in this work. This variant does not differentiate between uppercase and lowercase letters, making it more generalized and robust to variations in text casing.

By utilizing DistilBERT, this research aims to leverage the mentioned strengths to effectively identify near-duplicate HTML states, thereby improving the accuracy of web application model inference. The choice of DistilBERT reflects a balance between performance and practicality, aiming to provide a deep and contextualized understanding of HTML while maintaining manageable computational requirements.

3.3 HTML Page Representations

As introduced by Stocco et al. [22], this study will adopt three distinct types of token sequence representations of HTML for the model's input:

1. *Content*: Captures only the textual elements within HTML, such as text within $<p>$ or $<h1>$ tags, focusing on the semantic content of the web page.
2. *Tags*: Focuses on the structural HTML tags themselves, like $<div>$ and $<a>$, to understand the page's layout and organization.
3. *Content and Tags*: Combines the previous two representations, providing a comprehensive view that includes both text content and structural tags.

Varying the model inputs enhances comparability with the methodology outlined in Stocco et al. [22] and offers a valuable opportunity to identify the most effective techniques and representations for our analysis.

Furthermore, considering the limitations associated with the tokenizers used with DistilBERT, which typically have a maximum token length of 512[1], optimizing the HTML page representations for efficient processing is necessary. In a data preprocessing step, the study employs a "trimming" approach to reduce the size of the HTML representations from state-pairs without losing critical information. "Trimming" specifically refers to the removal of identical HTML segments at the beginning and end of the representations within a state-pair. For instance, consider a state-pair where the first state's representation is denoted by $HABCF$ and the second state's representation by $HDEF$. Here, each capital letter represents a block of HTML code; for example, H might be the header and F the footer. Trimming now involves comparing the two states and removing the common parts from the front and the back. Consequently, after trimming, State 1 would be shortened to ABC and State 2 to DE, both missing the header (H) and footer (F). By eliminating these commonalities, input size can be significantly reduced, allowing the model to focus on the unique and differing parts of the HTML. This trimming not only adheres to the token length constraint of DistilBERT but also potentially enhances the model's ability to capture and learn from the most discriminative features of the HTML content.

In the dataset under consideration (refer to Section 4.2), applying the trimming approach for every state-pair results, on average for each state, in a 20% trimming of the representation. This can be interpreted as a 20% size reduction on average for every state. For more detailed information, see Table 1.

[1] `https://huggingface.co/distilbert-base-uncased`

Table 1. Average trimming of states across the different HTML representations

HTML Representation	Avg % trimmed
`Content`	14.87
`Tags`	26.90
`Content + Tags`	18.18
`Average`	19.98

3.4 Approach 1: Calculating and Comparing Embeddings

In this initial experiment, we evaluate the unmodified form of *DistilBERT* [20] to determine its inherent ability to capture the semantic and structural nuances of HTML content, without specifically fine-tuning the model for near-duplicate detection. The primary goal is to understand the extent to which the embeddings generated by the pre-trained DistilBERT model can be utilized in detecting similarities and differences in HTML and to assess their efficacy in near-duplicate detection. The methodology includes the following steps:

- **Chunking and Tokenizing HTML Content**: The selected representation is loaded and chunked to adhere to the maximum sequence length constraint of DistilBERT. Each chunk is then tokenized using HuggingFace's AutoTokenizer[2], tailored for the DistilBERT model, converting the chunk into a format compatible with the model.
- **Generating Embeddings**: Embeddings are generated for each tokenized chunk using the DistilBERT model in a no-gradient context. These embeddings aim to capture the distilled semantic and structural essence of the HTML chunks.
- **Generating Unified Embedding Representations**: For a single HTML state, the embeddings generated from its segmented chunks undergo a combination process to construct a unified representation. Initially, these embeddings are concatenated to preserve the sequential context of the content. Subsequently, the concatenated tensor is subjected to an averaging operation, which distills the embeddings into a singular vector that captures the essence of the HTML content. This vector is then normalized to ensure consistency in scale and direction. The final output is a uniform vector comprised of 768 scalars, a dimensionality characteristic of the DistilBERT model's embedding output.
- **Similarity Calculation**: Both HTML states in the state-pair undergo the steps above independently. Then, the cosine similarity between the unified embeddings of the two states is calculated, providing a quantifiable measure of their similarity.
- **Threshold Inference**: An optimal threshold for classifying near-duplicates is determined by analyzing Receiver Operating Characteristic (ROC) curves. These curves plot the true positive rate against the false positive rate at various thresholds. The optimal point is identified as the threshold that maximizes the difference between these rates, balancing true positive and false positive rates effectively.

This approach allows for a detailed analysis of DistilBERT's raw embeddings in capturing the intricacies of HTML content, serving as a benchmark for assessing its native capabilities in near-duplicate detection tasks.

Remark 1. In their examination of near-duplicate detection, Yandrapally et al. [23] concluded that deriving a universal threshold applicable across multiple applications is impracticable, attributing this to the unique attributes inherent to each web application. Nonetheless, for the purpose of establishing a benchmark in this study, a universal threshold will be employed as a provisional measure, primarily to facilitate a comparative baseline for the more refined approach detailed in Section 3.5.

[2] `https://huggingface.co/docs/transformers/main_classes/tokenizer`

3.5 Approach 2: Fine-Tuning DistilBERT

After establishing benchmarks by computing embeddings with DistilBERT, this experiment involves fine-tuning the model for the task of near-duplicate detection. Fine-tuning further trains the pre-trained deep learning model specifically for the binary near-duplicate detection task, as discussed in Subsection 3.1. This process enables the model to refine its general language understanding capabilities to more effectively address the specialized task of detecting subtle differences and similarities characteristic of near-duplicate web pages.

For this purpose, we utilize Hugging Face's Transformers Library [4], which significantly simplifies the training process and data handling. It also provides mechanisms for training the model with custom datasets, allowing DistilBERT to learn from the labeled data introduced by Yandrapally et al. [23]. Hugging Face's extensive documentation and community support make the fine-tuning process more accessible and efficient. The high-level steps include:

- **Tokenization**: As in the *Raw Embeddings* approach (Section 3.4), we use AutoTokenizer from the Transformers library to tokenize the trimmed HTML content of each state-pair, ensuring compatibility with DistilBERT's format and sequence length constraints.
- **Data Collation**: DataCollatorWithPadding[3] is used for dynamic padding, ensuring uniform batch sizes during model training.
- **Model Setup**: An uncased DistilBERT sequence classification model is initialized for binary classification using AutoModelForSequenceClassification[4] from the HuggingFace Transformers library.
- **Training Configuration**: Training parameters are set using the default values recommended by Huggingface, including a learning rate of 2e-5, a batch size of 8, and two training epochs. These settings balance training efficiency with the need for model convergence and optimization.
- **Training Process**: Model training is conducted using the Trainer class[5], integrating the prepared datasets, model, and training configurations. The final models are published on HuggingFace (refer to Section 4.6.3 for reproducibility details).

Through fine-tuning DistilBERT, this study aims to develop a more effective tool for near-duplicate detection, leveraging the model's advanced deep learning capabilities for nuanced and context-aware analysis of HTML content. We anticipate that this fine-tuned model will outperform the *Raw Embeddings* approach, as discussed in Subsection 3.4.

4 Evaluation

4.1 Research Questions

This study aims to enhance near-duplicate detection in web testing and evaluate the feasibility of two methodologies: *raw embeddings* (refer to Section 3.4) and *fine-tuning the model* (see Section 3.5). To achieve this, the following research questions are posed:

1. RQ_1 **(Near-Duplicate Detection)**: How effective are the approaches in deciding whether a state-pair of web pages within an Application Under Test (AUT) is distinct or similar (i.e. clone or near-duplicate)?
2. RQ_2 **(Time Efficiency)**: What is the overall time efficiency of these approaches, and how does it influence their suitability for practical implementation in real-world web crawling environments?

[3] `https://huggingface.co/docs/transformers/main/en/main_classes/data_collator`
[4] `https://huggingface.co/transformers/v3.0.2/model_doc/auto.html`
[5] `https://huggingface.co/docs/transformers/main_classes/trainer`

Table 2. Details on the subject apps in *DS*

Name	State-Pairs	Clones	Near-Duplicates	Distinct
Addressbook	8,515	26	2,347	6,142
PetClinic	11,175	2	1,613	9,411
Claroline	17,766	2,707	71	14,988
Dimeshift	11,628	375	570	10,683
PageKit	9,730	0	3,948	5,782
Phoenix	11,175	1	4,605	6,569
PPMA	4851	64	467	4,320
MRBS	11,325	27	4,044	7,254
MantisBT	11,325	2	1,117	10,206
Total	97,490	3,204	18,782	75,355

RQ_1 aims to identify the most effective methodology and configuration for improving performance in near-duplicate detection. This question will explore various input modalities, including the type of representation (tags, content, and content plus tags) and the extent of the representation (trimmed versus whole), as outlined in Section 3.3. Additionally, this research will compare these approaches against existing near-duplicate detection techniques detailed in Section 2, ensuring a comprehensive evaluation.

RQ_2 assesses the practicality of employing the proposed approaches in real-world web crawlers, with a particular focus on time efficiency. This aspect is critical since a web crawler must rapidly determine during runtime whether to incorporate a newly crawled state into the web application model. This decision is based on assessing the similarity of every state already included in the model with the new state. In applications with numerous states, this necessitates an efficient near-duplicate detection technique to ensure timely processing.

4.2 Dataset

The dataset, referred to as *DS* and introduced by Yandrapally et al. [23], is used as the input for the model's training and evaluation. *DS* comprises 97.5k within-app state-pairs, which have been manually labeled as distinct, near-duplicate, or clone. These state-pairs were extracted from nine open-source web applications, varying in domain and size (as detailed in Table 2).

For training purposes, *DS* is partitioned into an 80%-20% split for training and test data, respectively. This split is executed individually for each web app to ensure a balanced representation of all applications in the dataset, regardless of their individual size or the number of state-pairs they contribute.

4.3 Procedure and Evaluation Metrics

4.3.1 RQ_1 (Near-Duplicate Detection) For both the *Raw Embeddings* (refer to Section 3.4) and the *Fine-Tuning* (see Section 3.5) approaches, we experimented with varying the model input. Specifically, for each approach, three distinct types of HTML representations were used, as detailed in Section 3.3: (1) *Content*, (2) *Tags*, and (3) *Content + Tags*.

Specifics - Raw Embeddings: In the initial approach, where DistilBERT generates "raw" embeddings for the two states in a state-pair, we experimented with varying lengths of HTML representations. As discussed in Section 3.3, this involved using either the entire token sequence or a trimmed version that omits common tokens from both the beginning and end of the documents. For consistency and to facilitate comparison with the second approach, we employ the trimmed representations also for the analysis of this first approach. For a training set of labeled state-pairs, we generated embeddings for each state in the pair and calculated the

cosine similarity between these embeddings. To determine an optimal threshold for classifying a state-pair as duplicated, ROC curve analysis was employed. This analysis involved identifying the maximum difference between the true positive rate and the false positive rate across a range of thresholds. Specifically, this analysis included every unique predicted score from the respective setting as a potential threshold. A universal, optimal threshold applicable across all nine subject apps in *DS* was inferred. Additionally, as discussed in Remark 1, a unique threshold for each subject app was also determined for comparative purposes. Due to space constraints, the respective optimal thresholds from the ROC curve analysis are reported online. Refer to Section 4.6.3 for the full results. The model's performance was evaluated using metrics such as precision, recall, F_1-Score, and accuracy.

Specifics - Fine-Tuning: In the Fine-Tuning approach, the F_1-Score was employed as the primary metric during training, with hyperparameter settings as recommended by Hugging Face. Specifically, the learning rate was set to 2e-05, and the training and evaluation batch size was set to 8. Gradient accumulation over 4 steps was employed, effectively simulating a larger batch size of 32. The Adam optimizer, with betas of (0.9, 0.999) and an epsilon of 1e-08, was selected for optimization. A linear type learning rate scheduler was used, and the models underwent training for a total of 2 epochs. For performance evaluation, metrics including precision, recall, F_1-Score, and accuracy were utilized. Due to the input length constraints of the DistilBERT tokenizer, training was conducted exclusively on the trimmed representation of the data. For further details on the various HTML representations, refer to Section 3.3.

4.3.2 RQ_2 (Time Efficiency) In evaluating RQ_2, the study assesses the time efficiency of both the Raw Embeddings and Fine-Tuning approaches, focusing on analyzing their inference and training times. The inference time analysis was conducted on a subset of 10,000 randomly selected state-pairs from *DS*. This involved recording the total inference time and averaging it across the number of inferences to provide a detailed understanding of each approach's efficiency.

Specifics - Raw Embeddings: As this approach does not involve a traditional "training phase", the efficiency analysis for Raw Embeddings centers on the average inference time required to generate and compare embeddings for two states in a state-pair of web pages. To identify the most time-efficient configuration, we varied the modalities, including both the length and type of input representations—trimmed versus whole representations, and those containing content, tags, or a combination of content and tags. For detailed information on these modalities, refer to Section 3.3.

Specifics - Fine-Tuning: The evaluation of time efficiency for Fine-Tuning is centered on assessing the average inference time, with the model training time also documented as a one-time investment. This inclusion serves to provide a complete overview, though the primary focus remains on inference efficiency. While the inference time assessment mirrors the methodology used in Raw Embeddings, the training involves the entirety of *DS*, split per-app into 80% for training and 20% for testing. The model fine-tuning includes trimming the states, tokenizing them, and subsequently training the model for the downstream task of near-duplicate detection. In line with RQ_1, we considered variations in the type of representation but not in the length of representation for model input.

4.4 Baselines

As a foundational baseline, this paper employs RTED (Robust Tree Edit Distance) [17], a robust algorithm that leverages the Document Object Model (DOM) for calculating tree edit distances. Specifically, we utilize the implementation of RTED used in Crawljax [13], a state-of-the-art crawler widely recognized in web testing. The inclusion of RTED as a baseline is motivated by findings from Yandrapally et al. [23], who identified it as the most effective structural algorithm

for near-duplicate detection. Additionally, RTED was the fastest among all the techniques they studied.

Another baseline used is *WebEmbed*, a method proposed by Stocco et al. [22]. This approach, built on a modified Doc2Vec model tailored for HTML pages, generates neural embeddings for pairs of web page states. These embeddings form the basis for assessing similarity, followed by a classifier (SVM) that determines whether the state-pair is distinct or duplicate. Given its superior performance in near-duplicate detection compared to previous techniques and the fact that it trains on the same data, WebEmbed serves as a solid benchmark for this study.

Remark 2. Regarding RQ_1 (near-duplicate detection), directly comparing the aforementioned baselines [17,22,23] with the Fine-Tuning approach is challenging due to slight differences in training methodologies. Specifically, Stocco et al. [22] trained their classifier in a 'leave-one-out' fashion, meaning they trained on eight apps and tested on one. In contrast, our approach involves a 80/20 percent train-test split across the entire dataset. While this methodological variance complicates a direct comparison, the results from Stocco et al. [22] offer a valuable benchmark for a broader context. In terms of the Raw Embeddings approach, a direct comparison is more feasible. Here, the methodology for training the classifier in Stocco et al. [22] matches our methodology for finding the optimal threshold. Specifically, there is a 80/20 percent train-test split inside every app which is utilized for training the classifier in Stocco et al. [22] and for finding the optimal threshold in this study.

4.5 Results

4.5.1 RQ_1 (Near-Duplicate Detection) Table 3 and 4 present the results on near-duplicate detection for the two novel approaches discussed in this paper as well as baselines by reporting the F_1-Score (F_1) and accuracy (Acc.) for the different techniques being compared. Due to limited space, for the raw embeddings approach, we only present the results using an optimal threshold for every app and then averaging the metrics across all apps. For the complete results, please refer to Section 4.6.3.

Raw Embeddings: The raw embeddings approach demonstrated relatively weak performance, which can be observed in more detail in Table 3. Using app specific thresholds, the best F_1-Score, 0.73, was produced by the whole tags configuration. This is a decrease in F_1-Score of -22% w.r.t. WebEmbed and a decrease in F_1-Score of -13% w.r.t RTED.

Fine-Tuning: The outcomes of the fine-tuning approach as well as the baselines are presented in Table 4. While direct comparisons with existing approaches present certain challenges (as noted in Remark 2), the trained model demonstrates +10% increase in F_1 w.r.t. WebEmbed and +17% w.r.t. RTED.

Remark 3. In the comparative analysis presented in Tables 3 and 4, the variances observed in the performance metrics for the same baseline, namely WebEmbed and RTED, can be attributed to the differing training methodologies employed. Specifically, the approach taken for the results in Table 3 involved training a distinct classifier for each application as it matches our approach of inferring the optimal threshold. The results in Table 4 stem from a more generalized training approach. Here, the classifier was trained across eight applications, with the ninth application reserved exclusively for testing. This more general way of training the classifier aligns more closely with the training procedure employed in our approach. The different methodology of training the classifier in the baselines explains the differences in the reported performance metrics of the same baseline. Please refer to Remark 2 for further details.

Table 3. RQ_1 (Near-Duplicate Detection) — ***Raw Embeddings***

Technique	Trimmed	Acc.	F1
`content + tags`	*true*	0.82	0.62
`content`	*true*	0.82	0.63
`tags`	*true*	0.83	0.68
`content + tags`	*false*	0.83	0.68
`content`	*false*	0.82	0.71
`tags`	*false*	0.84	0.73
`WebEmbed`	*false*	0.93	0.95
`RTED`	*false*	0.84	0.86

Table 4. RQ_1 (Near-Duplicate Detection) — ***Fine-Tuning***

Technique	Acc.	F1
`content + tags`	0.95	0.95
`content`	**0.97**	**0.97**
`tags`	0.92	0.93
`WebEmbed`	0.83	0.87
`RTED`	0.77	0.80

Table 5. Raw Embeddings: Average inference time in *seconds* - **trimmed** representation

Representation	Duration
`content + tags`	6.42
`content`	1.15
`tags`	5.05

Table 6. Raw Embeddings: Average inference time in *seconds* - **whole** representation

Representation	Duration
`content + tags`	7.93
`content`	1.32
`tags`	6.94

4.5.2 RQ_2 (time-efficiency) This section presents the findings related to the time efficiency of the two approaches, *Raw Embeddings* and *Fine-Tuning*, and assesses their practicality for use in real-time web crawlers for near-duplicate detection. Time recordings for inference tasks were conducted using a CPU provided by Google Colab[6] (Intel Xeon CPU @ 2.20 GHz).

Raw Embeddings: The average time taken to generate and compare two embeddings of a state-pair using the *Raw Embeddings* approach was recorded for each representation type: content, content+tags, and tags. The results can be observed in detail in Table 5 (trimmed representations) and in Table 6 (whole representations). The fastest configuration, trimmed content, achieves an average of 1.15 seconds per inference. This is 1.1 seconds or 23 times slower w.r.t. RTED.

Fine-Tuning: The time required for training on a T4 GPU was recorded for each HTML representation. Additionally, the average time for inference, which includes both trimming and actual model inference, was measured using an Intel Xeon CPU @ 2.20 GHz. The detailed timings are as follows:

- **Inital Effort of Training the Model**: Table 7 details the preparation times in minutes across all representations. Preparation time includes the time needed for preparing the data, i.e. trimming the representation and tokenizing it, as well as the time needed for the actual training across two epochs. Using only the content representation yields the fastest duration of 152 minutes.
- **Inference Time**: The precise inference time is detailed in Table 8. The model trained only on the content representation results in the best inference duration, averaging 0.92 seconds per inference. In direct comparison with the fastest *Raw Embeddings* configuration, *Fine-Tuning* is around 25% faster. Nevertheless, the pure average inference time of RTED (0.05 seconds) is almost 20 times faster than the model trained on the content representation. Yandrapally et. al [23] reported that a crawler using RTED as near-duplicate detection technique at runtime crawls about 25 states per minute. In a theoretical ceteris paribus

[6] `https://colab.research.google.com`

Table 7. Time in *minutes* required for fine-tuning the model using *DS*

HTML Representation	Trimming	Tokenization	Training	Total
`Content`	3	14	135	**152**
`Tags`	5	21	130	156
`Content + Tags`	6	26	150	182

Table 8. Fine-Tuning: Average inference time in *seconds*

Method	Duration
`Content`	0.9172
`Tags`	0.9349
`Content + Tags`	0.9457
`RTED`	0.0553

comparison, a crawler utilizing our trained model for near-duplicate detection would crawl about 1.5 states per minute. The slowest near-duplicate detection technique reported by Yandrapally et al. [23] is more than twice as fast (4 states per minute).

4.6 Threats to Validity

4.6.1 Internal Validity A key concern regarding internal validity arises from the chunking process in the Raw Embeddings approach using the whole representation. Chunking is needed, because the generated embeddings are produced by distilBERT, which has a tokenizer with a maximum token length of 512. The whole HTML representations which form the input are mostly larger (refer to Section 3.3). By breaking down web pages into chunks for processing, there is a potential loss of contextual information across chunks. This fragmentation might affect the approach's ability to accurately discern near-duplicates, as the continuity of the page's content and structure can be integral to its overall meaning and classification.

Another significant challenge impacting internal validity in the "Raw Embeddings"-approach relates to the threshold determination. Due to the extensive computational time (see Table 6), it was not feasible to establish this threshold across the entire dataset, but rather just 20% of the data. The inability to learn the threshold on the full dataset might result in a less accurate near-duplicate detection, as the threshold may not be representative of the diverse range of webpage pairs.

4.6.2 External Validity Several factors threaten the external validity of the findings. Firstly, the dataset used for training is relatively small, including only 9 web applications and around 97.5k state-pairs. The limited size and scope of the dataset may not adequately represent the diversity of real-world web applications, thus potentially affecting the generalizability of the findings.

Moreover, all nine web applications in *DS* provide CRUD (Create, Read, Update, Delete) functionality. This homogeneity raises concerns about the model's applicability and performance across a broader range of web applications with completely different functionalities and structures.

Additionally, the study's findings are derived from experiments utilizing a specific transformer model (distilBERT). It is uncertain whether these results would generalize to other transformer models, which may have different architectural nuances and pre-training methodologies. The performance demonstrated in this study may not be indicative of the behavior of other transformer based models when applied to the task of near-duplicate detection.

Lastly, the recorded times for model training and inference are specific to the dataset in question. It is conceivable that the performance times may vary when applied to other web applications, especially those differing significantly in complexity and content from the ones in the dataset. This variability could impact the practicality of employing these approaches in different real-world scenarios, affecting the broader applicability of the conclusions.

4.6.3 Reproducibility The fine-tuned models are available at `https://huggingface.co/lgk03`. All results, the scripts utilized for data preparation, generating the embeddings, training and inference are available on GitHub at `https://github.com/lgk03/near-duplicate-detection`.

5 Discussion

5.1 Superior Performance of Content-Based Fine-Tuning

The fine-tuned DistilBERT model demonstrates its highest performance when utilizing only the content representation of web pages as input, suggesting that the model's strengths lie in its linguistic capabilities rather than its understanding of HTML structure. This observation aligns with the inherent design of transformer-based models, which are adept at capturing contextual nuances in natural language, enabling them to discern semantic similarities effectively. The fine-tuning process, when applied exclusively to textual content, allows DistilBERT to leverage its pre-trained language understanding to identify near-duplicates with higher precision. This suggests the model capitalizes on its language processing capabilities rather than learning the intricacies of HTML. By focusing on the textual content, the model is less distracted by the variability and noise that HTML tags may introduce, which could be why the content representation as model input performs best.

Another factor explaining the superior performance of the content-only fine-tuned model may relate to DistilBERT's maximum sequence length limitation. Given that transformer-based models like DistilBERT have a set limit on the number of tokens they can process in a single sequence (see Section 3.3), inputs exceeding this length must be truncated. The content-only input, averaging the shortest representation length, is less likely to undergo such truncation, thereby preserving the full informational content of the web pages. This is contrasted with the content + tags and tags representations, which, due to their longer sequence lengths, may suffer from truncation. Consequently, important semantic information at the end of these sequences might be lost, impairing the model's ability to perform accurate near-duplicate detection.

5.2 DistilBERT vs. RTED in Near Duplicate Detection — Example

Yandrapally et al. [23] identified a significant challenge in near-duplicate detection with the RTED algorithm, especially in scenarios involving dynamically created content, such as web elements added or removed through user interactions. This type of near-duplicate, referred to as Nd3, is particularly challenging for RTED to detect. Nd3 near-duplicates not only consume significant testing time but can also lead to a continuous loop of duplicate creation during web crawling. Therefore, detecting these duplicates is crucial for ensuring concise web app model inference.

Consider an example of a Nd3 near-duplicate from *DS*, where the state-pair in question falls into the Nd3 category, as both states include a table in their HTML, but the second state has a differing row count. As is typical for these kinds of Nd3 duplicates, RTED, specifically the implementation proposed in Crawljax [13], fails to detect it. We computed the respective distance, which came out to be 0.671. The optimal threshold for RTED, as reported by Yandrapally et al. [23], is 0.041, where the distance needs to be smaller than this threshold to be considered a

duplicate. This highlights the significant gap in detection capability. In contrast, our fine-tuned DistilBERT model, trained on the content representation, correctly classifies this state-pair as a duplicate with 99.7% confidence.

The success of our fine-tuned DistilBERT model can be attributed to its refined ability to understand semantic similarities. Unlike RTED, which relies on structural comparisons, DistilBERT's nuanced approach to content context and subtleties enables it to effectively recognize similarities in dynamically altered web elements. This emphasizes the capability of advanced language models, like DistilBERT, in managing complex near-duplicate scenarios where traditional structural comparison methods, like RTED, fail.

5.3 Accuracy-Time Tradeoff

In this study, we evaluate the efficacy of near-duplicate detection in web application model inference, focusing on two key metrics: accuracy and time efficiency. Accuracy is paramount to ensuring a complete and non-redundant web app model, while time efficiency is crucial for the scalability and speed of the crawling process.

Our findings indicate a tradeoff between these metrics. The fine-tuned models, though slower in inference compared to traditional methods like RTED, demonstrate markedly higher accuracy. For example, the content-based fine-tuning approach yielded a F_1 score of 0.97, outperforming RTED's score of 0.80. However, this superior accuracy results in longer inference times, with our method averaging 0.92 seconds per state-pair comparison, as opposed to RTED's 0.06 seconds. We suggest that the extended duration of the crawl due to longer inference times is a worthwhile exchange for maintaining a concise model.

Additionally, the high accuracy in detecting logically similar web pages may lead to fewer states in the model during a crawler's runtime. This reduction in the number of states could, in turn, decrease the total number of inferences required when crawling new states, potentially reducing the overall time for a crawl. Exploring this possibility represents an intriguing direction for future research, as it suggests that enhanced accuracy might indirectly contribute to greater time efficiency in certain crawling scenarios.

6 Conclusions and Future Work

This study aims to enhance the detection of logically similar web pages during the runtime of a crawler tasked with web application model inference. Termed near-duplicate detection, this challenge is addressed through the application of a transformer-based, pre-trained deep learning model, specifically DistilBERT. We investigate the utility of DistilBERT by both leveraging the model's embeddings and fine-tuning it for near-duplicate detection. The effectiveness of these approaches in identifying near-duplicates and their time efficiency as practical solutions are assessed. While the embeddings approach did not result in any improvement, fine-tuning DistilBERT demonstrated superior near-duplicate detection capabilities compared to existing methods. However, it also incurred significantly longer inference times, underlining the trade-off between near-duplicate detection accuracy and time efficiency.

Future work will integrate the fine-tuned model within an existing web crawler to evaluate its real-world performance and to assess the quality of the inferred web application model by employing it in the downstream task of *Web Test Generation*. Furthermore, future work should explore varying the underlying deep learning models, for example, by utilizing BERT or GPT.

References

1. Candea, G., Godefroid, P.: Automated software test generation: some challenges, solutions, and recent advances. Computing and Software Science: State of the Art and Perspectives pp. 505–531 (2019)

2. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
3. Di Lucca, G.A., Fasolino, A.R., Faralli, F., De Carlini, U.: Testing web applications. In: International Conference on Software Maintenance, 2002. Proceedings. pp. 310–319. IEEE (2002)
4. Face, H.: Hugging face: State-of-the-art natural language processing, `https://huggingface.co/`
5. Fetterly, D., Manasse, M., Najork, M.: On the evolution of clusters of near-duplicate web pages. In: Proceedings of the IEEE/LEOS 3rd International Conference on Numerical Simulation of Semiconductor Optoelectronic Devices (IEEE Cat. No. 03EX726). pp. 37–45. IEEE (2003)
6. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
7. Haveliwala, T.H., Gionis, A., Klein, D., Indyk, P.: Evaluating strategies for similarity search on the web. In: Proceedings of the 11th international conference on World Wide Web. pp. 432–442 (2002)
8. Henzinger, M.: Finding near-duplicate web pages: a large-scale evaluation of algorithms. In: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 284–291 (2006)
9. Khan, S., Naseer, M., Hayat, M., Zamir, S.W., Khan, F.S., Shah, M.: Transformers in vision: A survey. ACM computing surveys (CSUR) **54**(10s), 1–41 (2022)
10. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International conference on machine learning. pp. 1188–1196. PMLR (2014)
11. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
12. Manku, G.S., Jain, A., Das Sarma, A.: Detecting near-duplicates for web crawling. In: Proceedings of the 16th international conference on World Wide Web. pp. 141–150 (2007)
13. Mesbah, A., Van Deursen, A., Lenselink, S.: Crawling ajax-based web applications through dynamic analysis of user interface state changes. ACM Transactions on the Web (TWEB) **6**(1), 1–30 (2012)
14. Mesbah, A., Van Deursen, A., Roest, D.: Invariant-based automatic testing of modern web applications. IEEE Transactions on Software Engineering **38**(1), 35–53 (2011)
15. Opara, C., Wei, B., Chen, Y.: Htmlphish: Enabling phishing web page detection by applying deep learning techniques on html analysis. In: 2020 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2020)
16. Otter, D.W., Medina, J.R., Kalita, J.K.: A survey of the usages of deep learning for natural language processing. IEEE transactions on neural networks and learning systems **32**(2), 604–624 (2020)
17. Pawlik, M., Augsten, N.: Efficient computation of the tree edit distance. ACM Transactions on Database Systems (TODS) **40**(1), 1–40 (2015)
18. Ramya, P., Sindhura, V., Sagar, P.V.: Testing using selenium web driver. In: 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT). pp. 1–7. IEEE (2017)
19. Ricca, F., Leotta, M., Stocco, A.: Three open problems in the context of e2e web testing and a vision: Neonate. In: Advances in Computers, vol. 113, pp. 89–133. Elsevier (2019)
20. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108 (2019)
21. SeleniumHQ: Selenium: Browser automation framework, `https://www.selenium.dev/`
22. Stocco, A., Willi, A., Starace, L.L.L., Biagiola, M., Tonella, P.: Neural embeddings for web testing. arXiv preprint arXiv:2306.07400 (2023)
23. Yandrapally, R., Stocco, A., Mesbah, A.: Near-duplicate detection in web app model inference. In: Proceedings of the ACM/IEEE 42nd international conference on software engineering. pp. 186–197 (2020)