Unveiling Code Clone Patterns in Open Source VR Software: An Empirical Study

Huashan Chen[1,2], Zisheng Huang[1,2], Yifan Xu[1,2], Wenjie Huang[1,2], Xuheng Wang[3], Jinfu Chen[4], Haotang Li[5], Kebin Peng[6], Feng Liu[1,2*], Sen He[4]

[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.
[2]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China.
[3]School of Computer Science and Technology, Beijing Jiaotong University, Beijing, China.
[4]School of Computer Science, Wuhan University, Wuhan, China.
[5]Department of Systems and Industrial Engineering, University of Arizona, Tucson, USA.
[6]Department of Computer Science, East Carolina University, Greenville, USA.

*Corresponding author(s). E-mail(s): liufeng@iie.ac.cn;
Contributing authors: chenhuashan@iie.ac.cn; huangzisheng@iie.ac.cn; xuyifan@iie.ac.cn; huangwenjie@iie.ac.cn; 22331104@bjtu.edu.cn; jinfuchen@whu.edu.cn; haotangl@arizona.edu; pengk24@ecu.edu; senhe@arizona.edu;

Abstract

Code cloning is frequently observed in software development, often leading to a variety of maintenance and security issues. While substantial research has been conducted on code cloning in traditional software, to the best of my knowledge, there is a lack of studies on cloning in virtual reality (VR) software that consider its unique nature, particularly the presence of numerous serialized files in conjunction with the source code. In this paper, we conduct the first large-scale quantitative empirical analysis of software clones in 345 open-source VR projects, using the NiCad detector for source code clone detection and large language models (LLMs) for identifying serialized file clones. Our study leads to

a number of insights into cloning phenomena in VR software, guided by seven carefully formulated research questions. These findings, along with their implications, are anticipated to provide useful guidance for both researchers and software developers within the VR field.

Keywords: Virtual Reality (VR), Code Clone, Large Language Model

1 Introduction

In recent years, Virtual Reality (VR) has gained substantial traction, finding significant applications across diverse domains such as gaming, healthcare, education, and entertainment[1–3]. An increasing number of applications are being created and distributed on leading app platforms (e.g., Google Play, Apple Store, Oculus), collectively reaching approximately 200 million downloads worldwide [4]. However, research on the quality of VR software remains limited, with code clone—a common yet often overlooked practice in software development—being particularly unexplored within this context.

Code clone generally refers to the repetition of identical or similar code snippets in multiple places within a software project, which sometimes may pose critical challenges for software security[5], maintainability[6] and changeability[7]. While there has been extensive research on the issue of code cloning in traditional software[8–16], to the best of my knowledge, our prior work[17] is the first quantitative empirical study of code cloning in open-source VR software. Due to space limitations, the study [17] limits its scope to source-code level code cloning. Nonetheless, VR software exhibits a unique characteristic compared to traditional software, that is, it incorporates a large number of serialized files, which are pivotal for data representation and storage in VR projects. Cloning of these files may also introduce various issues and potential risks. For instance, in cases where identical small balls are created by copying the same `.prefab`, any updates (e.g., to size or material) must be applied to each copy manually, leading to potential inconsistencies. Additionally, flaws in the original `.prefab`, like missing colliders or script errors, will affect all copies, making debugging more difficult. This reinforces the need for clone detection in such non-traditional code files in tandem with source code, a topic that is still surprisingly under-researched.

In this study, we substantially extend our previous work[17] through three key advancements: (i) we pioneer the exploration of clone-related issues in serialized files of VR software, a distinctive feature of VR systems compared to conventional software, whereas the prior study only examined source code clones. This is, to the best of our knowledge, the first work to analyze cloning in both source code and serialized files within VR software; (ii) we restructure the research framework by refining the original research questions (RQs) and evaluation metrics, and by formulating new VR-oriented questions, with the goal of shedding light on the mechanisms underlying code cloning in VR software and how these differ fundamentally from those observed in non-VR contexts; (iii) we considerably enlarge the dataset from 83 to 345 open-source VR projects, enhancing the robustness and generalizability of the findings.

Our study uncovers a number of insights into cloning phenomena in VR software and highlights the unique challenges of VR software development, offering practical guidance for researchers, developers, and industry practitioner within the VR field. Some of our insights are highlighted as follows: (i) Both source code cloning and serialized file cloning in VR software show a consistent pattern, with larger projects generally exhibiting more clones. (ii) Code and file cloning issues are most prevalent in gaming and education software, with gaming being the most significantly impacted, highlighting the need for targeted attention in this domain. (iii) Unlike traditional software, most VR projects exhibit significantly lower source code cloning due to their heavy reliance on serialized asset files. (iv) While C# is the most commonly used programming language in open-source VR projects, C-based projects exhibit a higher proportion of code cloning. (v) The introduction of third-party libraries in VR applications often contributes to source code cloning issues, sometimes constituting the majority of the cloning results. (vi) Inter-version code cloning is more common than intra-version code cloning, introducing significant security risks due to the potential propagation of vulnerabilities across multiple versions of the software. (vii) The cloning behavior of asset files varies with complexity. Complex files like scenes have lower clone levels, simpler ones like materials have higher, and prefab files fluctuate with development style.

To sum up, this paper provides the following main contributions:

- We highlight the unique nature of code cloning in VR software relative to traditional software and conduct a thorough quantitative empirical analysis of both source code clones and serialized asset file clones in our constructed dataset of 345 open-source VR projects.
- We propose a detection method based on large language models to identify clones in the unique serialized asset files of VR software, presenting a novel approach to code clone detection in this domain.
- We derive several insights from the experimental results that deepen our understanding of VR software, providing practical guidance for researchers, developers, and industry practitioner in the VR domain.

Paper Organization. The paper is structured as follows. Section 2 provides the background of VR software. Section 3 explains our methodology. Section 4 presents the results of seven research questions. Section 5 suggests future work. Section 6 discusses related work. Section 7 concludes the paper.

2 Background

In this section, we provide a brief introduction to VR software and discuss methods for detecting code cloning issues within it.

2.1 VR Software Architecture

VR software tightly integrates source code with serialized asset files, collectively shaping the application's functionality and immersive features. During runtime, source

code engages directly with assets to perform essential tasks, such as loading and rendering 3D models, triggering and managing animations, and processing user inputs to deliver interactive experiences. In the realm of VR software development, platforms like Unity [18–20] play a pivotal role, offering a comprehensive architectural model that synthesizes code-directed operational behaviors with immersive, asset-structured environmental elements.

Source Code. In VR software, the source code primarily consists of scripts that define the core functionality of the application. Typically written in C#, these scripts govern object behavior, manage user interactions, and implement key features such as rendering, physics simulations, and input handling. In Unity, source code adopts a content-logic decoupling and component-based architecture, where individual scripts are assigned to game objects to regulate specific aspects of their functionality. While this design enhances reusability, it can also result in redundancy, particularly in large-scale or collaborative development contexts.

Serialized Asset Files. In VR software, asset files serve as the backbone for delivering visual, auditory, and interactive content. In Unity, assets are serialized for efficient storage and retrieval. Common serialized files in VR software include asset files such as material files (`.mat`, `.shader`, `.cginc`), scene files (`.unity`, `.scene`, `.navmesh`), resource files (`.fbx`, `.wav`, `.png`), and template files (`.prefab`), along with configuration files (`.json`, `.ini`) and metadata files (`.meta`) bound one-to-one with asset files [18]. While the serialized structure ensures platform interoperability and smooth resource integration, its component-based design can result in redundancy across projects or scenes.

2.2 VR Code Clone

In light of the aforementioned VR software architecture, this paper extends the concept of code clone beyond its conventional scope at the source code level to include the cloning of data embedded in serialized asset files.

2.2.1 Source Code Clone

In the context of software development, source code clone denotes the duplication of similar code fragments within a codebase, where code fragment refers to a continuous segment of source code. Generally, there are four clone forms[14, 21–24]: (i) *identical clone*, referring to identical code fragments except for variations in comments and layout; (ii)*lexical clone*, referring to changes in identifier names and lexical values on the basis of the identical clone; (iii)*syntactic clone*, referring to syntactically similar code fragments that add statements of mutual addition, modification, or deletion on the basis of the identical and lexical clone; (iv)*semantic clone*, referring to syntactically different code fragments that implement the same functions. Note that the first three clone types indicate textual similarity whereas the last type reflects functional similarity.

Various approaches for source code clone detection have been proposed in the literature, as elaborated below. (i)Textual-based clone detection [25] compares the raw text of code to identify clones, typically relying on string-matching techniques. This

approach is suitable for detecting explicit and simple clones, such as exact copy-pasted code. Representative implementations of this method include Simian [26] and Duplo [27]. (ii)Token-based clone detection [28, 29] transforms the source code into tokens (lexical units) before comparison. By ignoring non-semantic elements such as whitespace, comments, and optionally identifiers, it identifies clones based on token sequences. Representative implementations of this method include NiCad [15, 30] and SourcererCC [24]. (iii)Syntax-based [31, 32] clone detection analyzes the abstract syntax tree (AST) of source code to compare the syntactic structure of code fragments, allowing it to eliminate formatting differences and detect syntactically similar code. Representative implementations of this method include CloneDR [33, 34] and Deckard [35]. (iv)Semantic-based clone detection [36, 37] identifies clones by analyzing the semantic structure of code fragments, focusing on whether the code fragments are functionally equivalent rather than merely syntactically similar. Representative implementations of this method include SCAM [38]. (v)Learning-based clone detection [32, 39] leverages machine learning or deep learning techniques to train models that automatically identify code clones. Recently, a rising number of studies have explored the use of Large Language Models (LLMs) for detecting code clones. [40]. Representative implementations of this method include CodeBERT [41, 42], BigCloneBench [43] and CodeGPTSensor [44].

It is worth noting that the first four detection methods are capable of detecting identical clones, lexical clones, and certain types of syntactic clones. In contrast, learning-based methods, including those utilizing LLMs, excel at identifying partial semantic clones but are less effective in detecting the other three types of clones compared to traditional approaches[40, 45–47]. Considering these facts, this study intends to employ Nicard, the state-of-the-art technique within traditional methods, for source code clone detection.

2.2.2 Serialized File Clone

Recall that serialized asset files in VR software serve as data containers, systematically defining the core visual and interactive aspects of a virtual environment. The structured nature of asset files is prone to cause "data clone", where configurations, asset references, or component settings are duplicated across files. Therefore, serialized asset file clone focuses on identifying duplicated or near-duplicated asset data, such as repeated material definitions, component hierarchies, or configuration settings. Existing detection techniques used for identifying data clone include: (i)Content-based methods, which typically construct feature vectors of the content for similarity comparison [48–51]. Representative implementations of this method include Scikit-learn [52]. (ii)Fingerprint-based methods, such as rolling hashes [53], which maps similar content to an identical hash value, and MD5 [54], which detects file clones by comparing the similarity of hash values. Representative implementations of this method include LSH [53] and Duplicate File Finder [55]. (iii)Deep learning-based methods, such as word vector models [56–58] and Transformer-based models [59], which train neural networks to identify textual similarity. Representative implementations of this method include Hugging Face Transformers [60].

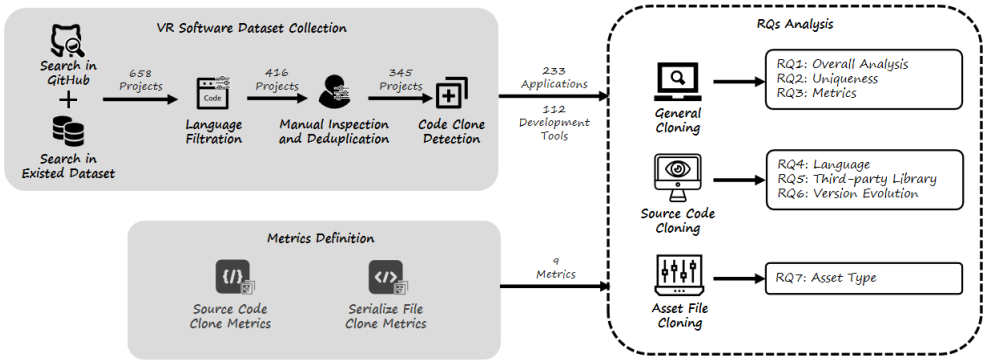


Fig. 1: Overview of Our Empirical Study.

Among these methods, content-based methods are simple and easy to implement, but they require significant computational resources and time, limiting their applicability to small-scale texts. Fingerprint-based methods, while computationally efficient, are sensitive to local changes and noise in the text. Deep learning-based methods, on the other hand, require substantial data and time for training, but with the advent of large models, this limitation has been significantly alleviated. In most scenarios, deep learning-based methods outperform the other approaches.

The nature of serialized asset files—often semi-structured, heterogeneous, and less amenable to syntax-based analysis—makes LLMs more suitable for capturing semantic similarities in this context. The latest studies indicate that GPT-4o, owing to its powerful contextual understanding and generative capabilities, often surpasses traditional embedding-based methods and other generative models in text similarity detection tasks, particularly those requiring reasoning and handling contextual variations [61, 62]. Given these facts, we propose using GPT-4o to identify clones in serialized asset files.

3 Methodology

The research methodology overview is shown in Figure 1. We begin by searching for VR projects in GitHub and existing datasets. After selecting 658 projects, we filter them by programming language, reducing the dataset to 416 projects. We then manually inspect and deduplicate these projects, resulting in a final dataset of 345 unique projects. We then propose detection methods for both source code cloning and serialized asset file cloning. Next, we formulate a set of metrics for measuring VR software cloning from various perspectives. Afterward, we conduct code cloning analysis from three angles to answer seven carefully formulated research questions.

3.1 Dataset Construction

Despite the availability of several datasets [17, 63–65] from prior studies, we have reconstructed a dataset tailored to meet four key criteria crucial for the objectives of this study: coverage to include a wide range of VR software projects, diversity to

encompass varied types of VR software projects, popularity to focus on prominent GitHub repositories, and usability to enable efficient detection. The generation of the dataset follows four phases, as described below.

Project selection. Given the dynamic accessibility of open-source projects on GitHub, we construct our dataset by combining the datasets disclosed in two latest studies [17, 63] and augmenting them with the latest projects published afterward. The dataset provided by [63] includes 314 open-source VR software projects, comprising 164 independent projects, 63 organizational projects, and 83 academic projects, with 4 of them no longer accessible. The dataset reported by [17] initially consists of 326 popular open-source VR software projects prior to any further processing. We further apply the same filtering criteria as outlined in [66] to incorporate the most recent data, resulting in the inclusion of 22 additional projects to enhance the dataset. By integrating the results of the three methods, we compile a preliminary dataset consisting of 658 open-source VR software projects.

Language filtration. Considering the wide range of development environments for open-source VR projects on GitHub, we propose to identify the projects that can be analyzed by off-the-shelf clone detection tools. For this purpose, we consider NiCad [15], a publicly available, state-of-the-art tool for source code clone detection, supporting languages such as C, C#, Java, Python, PHP, Ruby, Swift, ATL, and WSDL. By applying this tool to the previously obtained dataset, we exclude projects implemented in languages not supported by NiCad, resulting in 416 projects suitable for further analysis.

Manual inspection and deduplication. To ensure dataset accuracy, we manually review the tags and "README.md" files of each project, filtering out non-VR applications. Subsequently, we eliminate duplicate entries within the 416 projects, producing a final dataset of 345 projects, referred to as the VR-345 dataset. The dataset comprises 233 applications and 112 development tools.

3.2 Code Clone Detection

In this section, we provide detailed approaches for detecting cloning in both source code and serialized asset files, and define a set of metrics to quantify cloning from multi-faceted perspectives.

3.2.1 Source Code Clone Detection

We utilize various parameters of NiCad-6.2 to detect source code clones within the constructed dataset, followed by a detailed evaluation of the detection outcomes to compute relevant clone metrics. Recall that code clones are typically classified into four types: *identical*, *lexical*, *syntactic*, and *semantic* clones, as described in Section 2.2.1. As shown in Table 1, NiCad-6.2 is capable of detecting the first three types of clones, which are further refined into six distinct subtypes: Type1 represents identical clones; Type2 and Type2c correspond to lexical clones, differing in their handling of identifiers through two methods: blind and consistent; Type3-1, Type3-2, and Type3-2c correspond to syntactic clones, where a dissimilarity threshold, $1-\tau$, with τ denoting

Table 1: NiCad Settings for Source Code Clone Detection

Parameters	Target Clone Types					
	Type1	**Type2**	**Type2c**	**Type3-1**	**Type3-2**	**Type3-2c**
Dissimilarity Threshold	/	/	/	0.3	0.3	0.3
Identifier Renaming	none	blind	consistent	none	blind	consistent
Granularity	function	function	function	function	function	function

the similarity threshold defined above, is applied to enable the detection of near-miss clones. For example, a dissimilarity threshold of "0.1" allows for up to 10% difference. We set the dissimilarity threshold to "0.3", which is the default value. The cloning granularity may be set at the function or block level. We opt for function-level granularity to achieve results with greater detail and specificity.

We perform source code clone detection on the VR-345 dataset. The process involves normalizing and parsing source code files for NiCad compatibility, followed by clone analysis to identify clone pairs and classify them into clone classes. For each project, we calculate key metrics, including NCC, NCF, rcf, and rcc, to evaluate the clone detection results.

3.2.2 Serialization File Clone Detection.

Unlike traditional source code clones, asset clones often contain structured metadata, binary-encoded content, and heterogeneous formats. These characteristics demand semantic-level interpretation beyond surface-level syntax, thereby increasing the complexity of clone detection. While previous studies have demonstrated GPT-4o as the most effective LLM for text similarity detection tasks, it remains necessary to assess whether this conclusion holds in the context of asset file clone detection. Given the absence of a standardized dataset for this specific task, we design a three-step validation procedure: (i) independently evaluating 200 randomly selected file pairs using three different detection models; (ii) adopting the majority opinion (i.e., when two models agree) as a provisional benchmark in cases of conflicting results; and (iii) performing manual verification through a consensus-based review involving three researchers. As shown in Table 2, the final performance metrics confirm that GPT-4o remains the most effective model for asset file clone detection in our evaluation.

Table 2: Comparison of File Clone Detection Performance for Different LLMs.

Model	**Precision(%)**	**Recall (%)**	**F1-score (%)**	**Accuracy (%)**
GPT-4o	98.0	97.5	97.7	98.0
Claude 3.5	95.0	96.0	95.5	95.5
Gemini 1.5 pro	94.5	93.0	93.7	94.0

Note: (i) The consistency rate of the three models: 92.5% (185/200); (ii) GPT-4o got 11 edge cases correct.

We employ a two-step Chain of Thought (CoT) [67] technique using the GPT-4o large language model to identify similarities in serialized files for clone detection. Unlike traditional one-step question-answering prompts, this method decomposes complex problems into smaller, sequential steps, where each step's output informs the next.

Studies [68–70] support the effectiveness of CoT prompting in enhancing reasoning tasks, including clone identification.

Table 3: Step 1 of the CoT.

Step	Types	Prompts
1	Simple	*Please analyze the file pairs and determine if they are clone files.*
	Structure	*Please analyze the file pairs based on their component structures and determine if they are clone files.*
	Semantic	*Please analyze the file pairs based on their functional semantics and determine if they are clone files.*
	Similarity	*Please analyze the similarity of the file pairs and provide a similarity score between 0% and 100%.*

Step 1 of the CoT framework requires the LLM to evaluate whether a single file pair qualifies as a clone pair. To enable this process, we design four distinct prompt types: simple binary analysis, structural analysis, semantic analysis, and similarity analyses. Details of each prompt are provided in Table 3. We assess the effectiveness of these prompts by testing them on 200 randomly selected file pairs. The final performance is characterized using four standard metrics: precision, recall, F1-score, and accuracy. These metrics are derived through majority voting (using GPT-4o, Claude 3.5, Gemini 1.5 Pro), followed by manual inspection, in accordance with the three-step validation framework previously applied in LLM comparisons.

Table 4: Comparison of File Clone Detection Performance for Different Prompts.

Prompt	Precision(%)	Recall (%)	F1-score (%)	Accuracy (%)
Simple	52.5	40.0	45.0	45.0
Structure	60.0	80.0	68.6	55.0
Semantic	70.0	60.0	64.6	65.0
Similarity	98.0	97.5	97.7	98.0

Table 4 summarizes the experimental evaluation of the prompts. The simple prompt leads to the lowest performance, with precision at 52.5%, recall at 40.0%, F1-score at 45.0%, and accuracy at 45.0%, as the model struggles to effectively detect inter-document clones. The structure prompt achieves a higher recall of 80.0% but is hindered by a high false positive rate, with precision at 60.0%. This suggests that the model is overly sensitive to structural similarities within the files. The semantic prompt demonstrates moderate performance, with precision at 70.0%, recall at 60.0%, F1-score at 64.6%, and accuracy at 65.0%. This may be due to the simplified structure of serialized asset files compared to natural language. In contrast, the similarity analysis prompt excels with near-perfect precision at 98%, recall at 97.5%, F1-score at 97.7%, and accuracy at 98.0%, validating its ability in generating reliable similarity scores for file pairs.

After calculating the similarity among file pairs, we employ Step 2 of the CoT to compute the NCA, NCG and CI for each project, as detailed in Table 5. We set the clone threshold (τ') to 80%, following the default recommendation of GPT-4o. The results are subsequently refined through manual inspection and standardization to

Table 5: Step 2 of the CoT.

Step	Metrics	Prompts
2	NCA, NCG	*Please analyze the files in the folder and output the number of clone files and clone groups, referencing the cloning information of each file pair from the Output of Step 1.*
	CI	*Please analyze the files in the folder and output the clone index of the files, referencing the cloning information of each file pair from the Output of Step 1.*

ensure reliability. Note that the detected serialized files comprise scene files (`.unity`, `.scene`, `.navmesh`), template files (`.prefab`), material files (`.mat`, `.shader`, `.cginc`), and configuration files (`.json`, `.ini`), but exclude resource files (`.fbx`, `.wav`, `.png`) and `.meta files`. This is because resource files are typically not stored in serialized text format, while `.meta files` are tightly bound to their corresponding assets.

3.2.3 Metrics for Measuring Code Clone

For source code clone, we focus on code clone at the function granularity. Let F denote the universe of functions in a VR software, and let $\mathsf{Sim}(func_i, func_j)(i \neq j)$ denote the similarity between function $func_i \in F$ and function $func_j \in F$. We define the concepts of *clone function* and *clone class* below. It is important to note that the similarity between functions is computed using the NiCad tool, as described in Section 3.2.1.

Clone function. A function $func_i \in F$ is deemed a clone function if there exists at least one other function $func_j \in F(i \neq j)$ such that the similarity between $func_i$ and $func_j$ exceeds a predefined similarity threshold τ. Mathematically, a function $func_i \in F$ is a clone function under the following condition:

$$\exists func_j \in F(i \neq j), \mathsf{Sim}(func_i, func_j) > \tau \tag{1}$$

Clone class. A clone class is defined as a set of two or more functions where the similarity between every pair of functions exceeds a predefined similarity threshold τ. Mathematically, a set $C \subseteq F$ ($|C| \geq 2$) is a clone class under the following condition:

$$\forall func_i, func_j \in C(i \neq j), \mathsf{Sim}(func_i, func_j) > \tau \tag{2}$$

Grounded in the above definitions, we employ four straightforward metrics to evaluate the degree of source code cloning from both absolute and relative perspectives, following the existing studies [5, 17, 23, 71]. The metrics applied to assess the absolute degree of code cloning are as follows:

- **Number of Clone Functions (NCF):** This metric measures the total number of *clone functions* in the source code of a VR software.
- **Number of Clone Classes (NCC):** This metric measures the number of *clone classes* in the source code of a VR software.

The metrics applied to assess the relative degree of code cloning include:

- **Ratio of Clone Functions (rcf):** This metric reflects the share of *clone functions* in a VR software, which is defined as the ratio of the number of *clone functions* (NCF) to the total number of functions ($|F|$) in the software.
- **Ratio of Clone Classes (rcc):** This metric reflects the share of *clone classes* in a VR software, which is defined as the ratio of the number of *clone classes* (NCC) to the total number of functions ($|F|$) in the software.

For serialized asset files clone, we focus on data clone at the file granularity. Let A denote the universe of asset files in a VR software, and let $\mathsf{Sim}'(file_i, file_j)(i \neq j)$ denote the similarity between asset file $file_i \in A$ and asset file $file_j \in A$. We define the concepts of *clone file* and *clone group* as follows. It is important to note that the similarity between files is computed using the GPT-4o, as described in Section 3.2.2.

Clone file. A serialized asset file $file_i \in A$ is deemed a clone file if there exists at least one other asset file $file_j \in A(i \neq j)$ such that the similarity between $file_i$ and $file_j$ exceeds a predefined similarity threshold τ'. Mathematically, a serialized asset file $file_i \in A$ is a clone file under the following condition:

$$\exists file_j \in A(i \neq j), \mathsf{Sim}'(file_i, file_j) > \tau' \tag{3}$$

Clone group. A clone group is defined as a set of two or more serialized asset files where the similarity between every pair of files exceeds a predefined similarity threshold τ'. Mathematically, a set $G \subseteq A$ ($|G| \geq 2$) is a clone group under the following condition:

$$\forall file_i, file_j \in A(i \neq j), \mathsf{Sim}'(file_i, file_j) > \tau' \tag{4}$$

Analogous to source code clone metrics, we introduce a set of metrics to evaluate the extent of serialized asset cloning from both absolute and relative perspectives. The metrics applied to assess the absolute degree of asset cloning are as follows:

- **Number of Clone Assets (NCA):** This metric measures the total number of *clone files* in the serialized asset files of a VR software.
- **Number of Clone Groups (NCG):** This metric measures the number of *clone groups* in the serialized asset files of a VR software.

The metrics applied to assess the relative degree of asset cloning include:

- **Ratio of Clone Assets (rca):** This metric reflects the share of *clone files* in a VR software, which is defined as the ratio of the number of *clone files* (NCA) to the total number of asset files ($|A|$) in the software.
- **Ratio of Clone Groups (rcg):** This metric reflects the share of *clone groups* in a VR software, which is defined as the ratio of the number of *clone groups* (NCG) to the total number of asset files ($|A|$) in the software.

To sum up, the proposed metrics (NCF, rcf) and (NCC, rcc) are based on *clone functions* and *clone classes*, as defined in Equations (1) and (2), respectively, in the context of code clones; whereas (NCA, rca) and (NCG, rcg) are based on *clone files*

and *clone groups*, as defined in Equations (3) and (4), respectively, in the context of file clones.

To evaluate the overall similarity of asset files in a VR software, we introduce the **Clone Index (CI)**, a metric calculated by dividing the sum of pairwise similarities by the total number of asset files, namely

$$\mathsf{CI} = \frac{\sum_{file_i, file_j \in A, i \neq j} \mathsf{Sim}'(file_i, file_j)}{|A|} \tag{5}$$

3.3 Research Questions Definition

The objective of this study is to systematically investigate the phenomenon of code clones in VR software, with the aim of clarifying their characteristics, root causes, and implications. By doing so, we aim to produce actionable insights for researcher, developers, and industry practitioners, ultimately improving the quality, maintainability, and performance of VR software. To achieve this goal, we propose seven research questions (RQs) from three analytical perspectives: (i) general cloning in software, (ii) cloning specific to source code, and (iii) cloning specific to serialized asset files.

General cloning in software (RQ1∼RQ3) provides a comprehensive analysis of the overall cloning landscape in VR software, its intrinsic characteristics, and the measurement methods.

- **RQ1: Which VR projects are subjected to heavy cloning?** The motivation behind RQ1 is to identify VR projects with significant levels of code and asset cloning, enabling developers to prioritize refactoring efforts and manage technical debt more effectively, providing researchers with insights into why certain projects accumulate more clones, and helping managers allocate resources toward maintaining high-risk systems. To answer this question, we first evaluate how various detection methods influence cloning results, and then assess the extent of cloning across VR projects at both the source code and asset file levels.
- **RQ2: What are the main differences in code cloning between VR and non-VR software?** The motivation behind RQ2 is to understand how the unique structural and interactive features of VR software influence code cloning, since VR's unique demands (e.g., real-time rendering, physics simulation, device I/O) may lead to distinct cloning patterns that differ from traditional software, thereby enabling developers to adopt best practices from non-VR domains where applicable, guiding researchers to design VR-specific clone detection tools if distinctive patterns are observed, and assisting software architects in deciding whether to invest in dedicated infrastructure for VR development. To answer this question, we first analyze the structural distinctions between VR and non-VR software, then assess how these differences contribute to variations in code cloning.
- **RQ3: How to measure and analyze code cloning in VR software development?** The motivation behind RQ3 is to explore and evaluate the suitability of various cloning metrics in the context of VR software, as the unique characteristics of VR systems may require domain-specific measurement approaches, enabling developers to accurately assess code and asset maintainability, and helping researchers

and tool builders establish standardized, context-aware evaluation criteria for clone analysis. To answer this question, we offer an in-depth analysis of the physical interpretations and applicability of the proposed metrics for measuring cloning.

Cloning specific to source code (RQ4∼RQ6) explores the factors involved in the introduction of source code cloning.

- **RQ4: Which programming languages dominate VR software development and are more susceptible to code cloning?** The motivation behind RQ4 is to investigate how the choice of programming languages in VR development influences code cloning, since some languages may inherently promote more duplication than others, enabling developers to mitigate language-specific risks through targeted practices and guiding researchers in creating language-aware clone detection methods. To answer this question, we begin by identifying the dominant programming languages in VR development, then investigate how they contribute to code cloning and uncover the root causes behind these patterns.
- **RQ5: How do third-party libraries impact code cloning in VR software?** The motivation behind RQ5 is to examine whether third-party libraries in VR development lead to specific cloning behaviors due to boilerplate or API constraints, helping developers identify dependency-related duplication risks, enabling researchers to distinguish between library-induced and developer-induced cloning patterns for more accurate detection, and informing library authors on improving API design to minimize clone proliferation. To answer this question, we explore the third-party libraries frequently utilized in VR projects and analyze whether their usage leads to code cloning and the specific patterns of cloning that arise.
- **RQ6: How do intra-version and inter-version code cloning evolve across different versions of a VR project?** The motivation behind RQ6 is to examine the temporal dynamics of code cloning in VR projects, as understanding when and why duplication accumulates can help developers schedule refactoring milestones and assist researchers in identifying patterns that inform version-aware maintenance. To answer this question, we investigate the presence of code cloning both within the same software version and between consecutive versions, exploring the degree of cloning and particularly focusing on the differences in cloning across different granularities of version changes.

Cloning specific to serialized asset files (RQ7) examines the influence of asset types on file cloning.

- **RQ7: How do cloning practices vary across different types of asset files in VR software?** The motivation behind RQ7 is to investigate cloning differences among various asset types in VR software, since VR-specific asset types (e.g., 3D models, scenes, shaders) may exhibit different cloning characteristics than traditional software artifacts, helping developers implement asset-specific maintenance strategies, aiding researchers in expanding non-code clone detection in content-rich environments, and assisting asset managers in refining asset control workflows. To answer this question, we analyze the cloning discrepancies of various types of serialized asset files across multiple distinct projects.

4 Evaluation

4.1 Overall Analysis

4.1.1 RQ1: Which VR projects are subjected to heavy cloning?

To answer this RQ, we utilize NiCad to detect code clones within the VR-345 dataset and present the most cloned projects from the development tools and applications in Table 6. Specifically, PID 1-10 represent the top ten projects with the highest number of cloned functions in the development tool category, while PID 11-20 are the top ten projects in the VR application category with the most cloned functions. For ease of observation, these projects are arranged in descending order of the total number of functions N within each category.

We begin by examining the impact of different detection methods on the results, revealing a clear and consistent trend in the NCF metric (i.e., the number of clone functions): NCF(Type 3-2) > NCF(Type 3-2c) > NCF(Type 3-1), and NCF(Type 2) > NCF(Type 2c) > NCF(Type 1). This suggests that the blind renaming method consistently identifies the most clone fragments, with the consistent renaming method coming in second. This finding exactly aligns with the theoretical expectations outlined in NiCad [15], where blind renaming replaces all identifiers with a generic placeholder "X_n", while consistent renaming assigns sequential identifiers "X_n"(n is a sequence number). The rationale is straightforward: if a code fragment is recognized as similar with identifiers renamed as "X_n", it will also be recognized as similar when all identifiers are replaced with "X". Additionally, we observe that NCF(Type 3-x) > NCF(Type x) for x = 1, 2, 2c. This outcome is intuitive since a fragment meeting a 0% dissimilarity threshold will inherently meet a 30% threshold. Based on these observations, Type 3-2 emerges as the most effective detection method, with its results encompassing those of other types. In terms of the NCC metric, there appears to be no clear correlation between the detection types and this metric. As a result, we select Type 3-2 as the standard for code clone detection in the following analyses.

We further observe that the NCF values for all twenty projects exceed 100, suggesting that code cloning is a common issue in VR project development. Additionally, the highest number of functions ($|F|$) tend to exhibit the largest volume of code clones (NCF). Specifically, the top three projects with the most functions are "BlenderXR", "gpac", and "SoundSpace", which also show the highest occurrence of code clones. This observation suggests that, generally, the prevalence of code cloning increases as the scale of a project expands, echoing similar trends found in conventional software development.

To expand our study on cloning in serialized asset files within VR software, we add 10 more VR applications, covering diverse subcategories like games, education, simulators, and AR, ensuring a comprehensive analysis. Development tools are excluded from the analysis of file cloning because they are typically source-code-centric, with asset files serving configuration and descriptive purposes. In contrast, asset files dominate in application projects, with source code primarily existing as scripts to manipulate these assets. Table 7 provides the clone detection results for the 20 selected VR software applications.

Table 6: The quantity of clones detected in the source code.

PID	Project Name	Domain	Language	Type1		Type2		Type2c		Type3-1		Type3-2		Type3-2c		$\|F\|$
				NCF	NCC	NCF	NCC	NCF	NCC	NCF	NCC	NCF	NCC	NCF	NCC	
Development Tools																
1	BlenderXR	/	C,Python	358	175	1578	601	1486	567	1731	716	4448	1226	3112	889	27822
2	gpac	/	C	6	3	1421	259	1268	220	1195	312	2699	326	2089	278	10170
3	The-Seed-Link-Future	/	C#	44	19	231	84	223	85	201	86	462	166	295	112	8475
4	open-brush	/	C#	38	17	132	60	127	58	183	81	409	163	218	89	7548
5	UnityOculusAndroidVRBrowser	/	C#	36	15	156	53	150	55	107	47	255	86	177	65	3716
6	UnityPlugin	/	C#	12	6	53	21	52	21	60	24	113	43	85	33	2799
7	lovr	/	C	2	1	84	33	77	30	77	31	232	64	148	53	2173
8	UnityGameTemplate	/	C#	20	10	60	25	59	25	53	22	142	47	98	35	1981
9	ViveInputUtility-Unity	/	C#	20	10	62	27	62	27	69	32	125	49	84	36	1863
10	com.xrtk.core	/	C#	3	1	49	18	49	18	42	17	148	51	82	32	1607
Applications																
11	SoundSpace	simulator	C#	2149	1063	2190	1011	2185	1009	2175	985	2225	965	2201	1001	9117
12	RhythmAttack-VR	gaming	C#	37	17	162	60	149	62	118	48	330	119	221	89	6751
13	Dungeon-VR	gaming	C#	20	10	100	38	91	35	103	41	254	98	154	61	6470
14	Situated-Empathy-in-VR	education	C#	23	11	140	59	136	58	136	59	388	140	262	100	4788
15	mineRVa	education	C#	18	9	88	35	82	34	53	24	197	71	140	58	4722
16	VR-Escape-Room	gaming	C#	12	6	53	19	44	16	31	13	120	46	77	30	4067
17	Group6_ProjectNurture	healthcare	C#	27	10	148	49	142	51	102	44	248	82	171	62	3980
18	vrtist	education	C#	51	17	114	42	112	41	141	48	299	103	187	69	3322
19	Terminal	gaming	C#	19	8	96	27	88	29	71	27	160	49	121	41	2929
20	elite-vr-cockpit	gaming	C#	0	0	52	18	49	17	30	14	109	39	68	26	2725

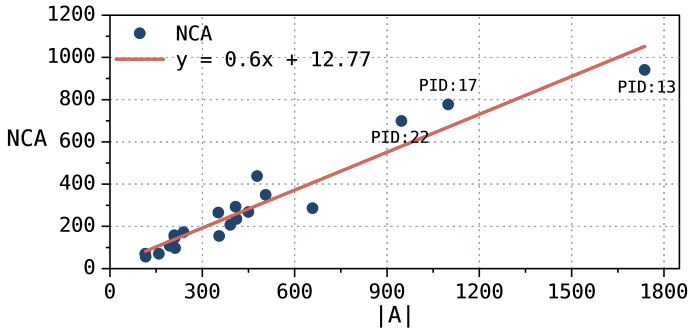


Fig. 2: The relationship between $|A|$ and NCA.

We observe that the three projects with the highest NCA (i.e., the number of file clones)—"Dungeon-VR", "Group6_ProjectNurture" and "OpendagVR2"—are also the ones with the largest $|A|$ (i.e., the number of asset files), following a pattern similar to source code cloning. For better clarity, Figure 2 visually depicts the relationship between project characteristics and clone metrics, specifically the correlation between the number of asset files ($|A|$) and file clones (NCA), with a near-linear relationship observed, suggesting that larger projects typically exhibit more file clones.

Insight 1. *Both source code cloning and serialized file cloning in VR software show a consistent pattern, with larger projects generally exhibiting more clones.*

When examining the domains of the top 20 most clone-prone software systems, gaming and education emerge as the most impacted. According to Table 6 and Table 7, code cloning occurs in 5 gaming and 3 education systems, while file cloning is present in 12 gaming and 5 education systems. This recurring pattern highlights the need for special attention to cloning control in these areas, especially in gaming.

Table 7: The quantity of clones detected in serialization files.

PID	Project Name	Domain	NCA	NCG				$\|A\|$	rca	rcg	CI
				2	**[3, 10]**	**> 10**	**sum**				
12	RhythmAttack-VR	gaming	286	20	19	5	44	658	43.47%	6.69%	8.17%
13	Dungeon-VR	gaming	941	31	20	15	66	1737	54.17%	3.80%	16.32%
16	VR-Escape-Room	gaming	154	10	6	4	20	355	43.38%	5.63%	9.69%
19	Terminal	gaming	268	11	13	4	28	450	59.56%	6.22%	23.18%
20	elite-vr-cockpit	gaming	97	5	8	1	14	212	45.75%	6.60%	12.35%
21	VRHamsterBall	gaming	108	5	10	1	16	194	55.67%	8.25%	16.37%
23	epicslash	gaming	145	7	8	6	21	210	69.05%	10.00%	17.46%
26	AirAttack	gaming	265	8	9	5	22	352	75.28%	6.25%	27.28%
27	GolfVR	gaming	56	4	3	2	9	116	48.28%	7.76%	13.84%
28	XR-Keyboard	gaming	70	5	2	2	9	159	44.03%	5.66%	15.18%
29	HorrorGame	gaming	349	12	27	7	46	506	68.97%	9.09%	13.93%
30	Pokemon-Themed-Kiosk-VR	gaming	172	10	4	5	19	239	71.97%	7.95%	22.96%
14	Situated-Empathy-in-VR	education	158	1	3	2	6	209	75.60%	2.87%	57.42%
15	mineRVa	education	235	8	8	3	19	411	57.18%	4.62%	18.67%
18	vrtist	education	293	21	17	5	43	408	71.81%	10.54%	14.89%
22	OpendagVR2	education	699	21	26	10	57	947	73.81%	6.02%	24.05%
25	Procrastination-VR	education	438	5	5	2	12	478	91.63%	2.51%	69.29%
17	Group6_ProjectNurture	healthcare	777	24	15	12	51	1099	70.70%	4.64%	23.28%
11	SoundSpace	simulator	207	10	15	3	28	391	52.94%	7.16%	11.24%
24	CityMatrixAR	simulator	70	3	5	2	10	115	60.87%	8.70%	25.13%

Insight 2. *Code and file cloning issues are most prevalent in gaming and education software, with gaming being the most significantly impacted, highlighting the need for targeted attention in this domain.*

4.1.2 RQ2: What are the main differences in code cloning between VR and non-VR software?

To answer this RQ, we start by reviewing prior work on code cloning in traditional software. Among the most relevant studies, Kamiya et al. [13] reported a cloned function ratio of 12% to 16% in medium and large software. Kapser et al. [72] found a similar ratio, close to 20%, in C-based projects. Roy et al. [71] observed that Java and C++ projects exhibited a clone function ratio ranging from 10% to 15%. These findings suggest that code cloning is a widespread and consistent issue across traditional software development, typically affecting 10% to 20% of functions.

To investigate code cloning in VR software, we propose exploring the relationship between the ratio of clone functions (rcf) and the quantity of asset files ($|A|$), as the use of such assets constitutes a key difference between VR and traditional software. As shown in Table 8, we observe that the development tools Project-1 ("BlenderXR") and Project-2 ("gpac"), along with the VR applications Project-11 ("SoundSpace") and Project-24 ("CityMatrixAR") exhibit significantly higher levels of source code cloning than other projects, with rcf values ranging from 16% to 27%, whereas most of the remaining 26 projects fall below 10%.

We further examine the four outliers with unusually high levels of code cloning. Project-1 and Project-2 lack any serialized asset files. Project-11 and Project-24, though containing some serialized files, have relatively few when compared to the overall volume of source code. Specifically, Project-24, an AR application, constructs models from real-world inputs rather than serialized data. Similarly, Project-11, which

Table 8: The correlation between the ratio of clone functions (rcf) and the number of asset files ($|A|$) across VR projects.

PID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
rcf	16.00%	26.54%	5.45%	5.42%	6.86%	4.04%	10.68%	7.17%	6.71%	9.21%
$\|A\|$	0	0	303	1037	321	222	0	63	93	8
PID	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
rcf	24.41%	4.89%	3.93%	8.10%	4.17%	2.95%	6.23%	9.00%	5.46%	4.00%
$\|A\|$	391	658	1737	209	411	355	1099	408	450	212
PID	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
rcf	3.65%	2.52%	4.11%	18.20%	7.45%	9.88%	7.11%	2.65%	3.87%	0.79%
$\|A\|$	194	947	210	115	478	352	116	159	506	239

focuses on acoustic simulation and visualization, does not require extensive use of serialized assets. A shared trait among these projects is the limited presence of serialized files, which aligns them more closely with traditional software patterns. As a result, they exhibit cloning behavior similar to that observed in traditional software, with clone function ratios exceeding 10%.

By comparison, VR software projects that utilize extensive serialized asset files demonstrate markedly reduced code cloning, as evidenced by the remaining 26 projects' average clone function ratio of 5.02%. This may result from the limited capability of clone detection tools (e.g., NiCad) to recognize duplicates in serialized files, which differ structurally from standard source code. It is important to note that Project-7 ("lovr"), although written in C, also contains many model class files (`/modules`), which contributes to its relatively lower levels of code cloning. This observation does not contradict our conclusions. In summary, asset files significantly influence code cloning patterns, presenting unique detection challenges that current clone analysis tools must address for effective VR software maintenance.

Insight 3. *Unlike traditional software, most VR projects exhibit significantly lower source code cloning due to their heavy reliance on serialized asset files.*

4.1.3 RQ3: How to measure and analyze code cloning in VR software development?

To answer this RQ, we sequentially analyze the applicability of the metrics defined in Section 3.2.3.

NCF and NCC. The physical interpretation of NCF is relatively intuitive; higher values directly indicate a greater cloning volume. However, the implication of NCC is less straightforward. Using Type3-2 in Table 6 as an example, we observe a notable difference between NCF and NCC. For instance, comparing Project-2 to Project-1, the NCF ratio is approximately $2699/4448 \approx 60.7\%$, whereas the NCC ratio is considerably lower at $326/1226 \approx 26.6\%$. In contrast, when comparing Project-4 to Project-3, the NCF ratio is $409/462 \approx 88.5\%$, but the NCC ratio is even higher at $163/166 \approx 98.2\%$. To uncover the underlying cause of these differences, we analyze the correlation between the number of clone classes and their respective sizes. As shown in Table

Table 9: The distribution of the sizes of clone classes.

Project Name \ NCC Interval	2	[3,10]	[11, 100]	> 100	Sum
BlenderXR (PID:1)	898(73.2%)	304(24.8%)	23(1.9%)	1(0.1%)	1226
gpac (PID:2)	207(63.5%)	100(30.7%)	13(4.0%)	6(1.8%)	326
The-Seed-Link-Future (PID:3)	122(73.5%)	39(23.5%)	5(3.0%)	0	166
open-brush (PID:4)	123(75.5%)	37(22.7%)	3(1.8%)	0	163

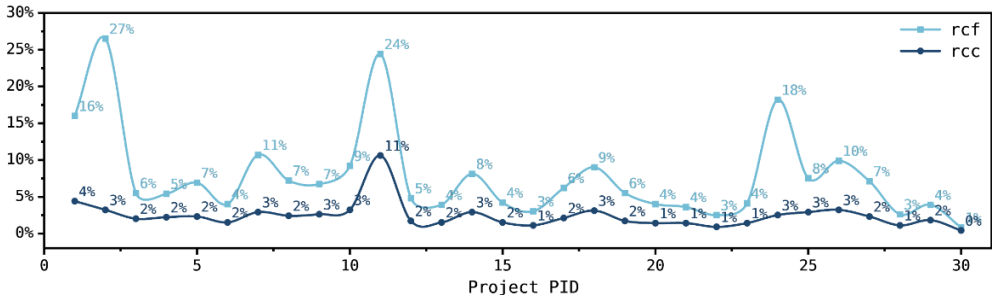


Fig. 3: The changes of rcf and rcc across diverse projects.

9, Project-2 exhibits a significantly higher proportion of large-size (>100) and mid-size ([11, 100]) classes compared to Project-1, leading to a notable reduction in NCC. However, Project-3 and Project-4 have similar clone class distributions across different intervals, which results in a close NCC. The observations imply that NCC acts as a valuable indicator of clone class size, with lower values typically reflecting the presence of larger clone classes.

rcf and rcc. To explore the implications of rcf(i.e., the proportion of clone functions, representing the density of function-level code clones) and rcc(i.e., the proportion of unique clone classes, representing the diversity of code clone patterns), we analyze their variations across different projects, as illustrated in Figure 3. The results reveal that rcf exhibits significant fluctuations, while rcc remains relatively consistent across projects except for Project-11. The anomaly in the project "SoundSpace" arises from two highly similar directories, "SoundSpace" and "rv_SoundSpace", with 902 of 965 clone classes being pairwise clones. These predominantly size-2 clones lead to an unusually high rcc. Additionally, no clear correlation is observed between rcf and rcc, suggesting that the density of function-level code clones and the diversity of clone patterns are largely independent of each other. This decoupling highlights the complexity of code clone characteristics and underscores the need for separate consideration of density and diversity in clone analysis.

NCA and NCG. The meanings of NCA and NCG are analogous to NCF and NCC in source code cloning. A higher NCA indicates a greater quantity of cloned assets within the project, while the impact of NCG is reflected in the concentration of clones. Given a fixed NCA value, a lower NCG suggests that the number of members in individual clone groups is higher, indicating more concentrated cloning within the project.

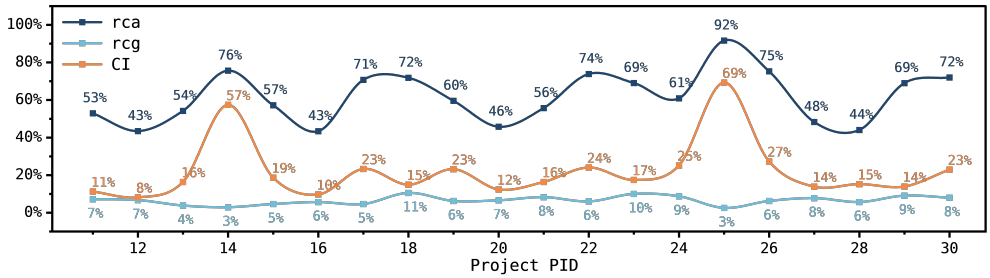


Fig. 4: The changes of rca, rcg and CI across diverse projects.

rca, rcg and CI. To explore the meanings of these three metrics, we plot their respective values for the 20 selected projects, as shown in Figure 4. The results indicate that rca and CI exhibit a certain degree of correlation, while rcg remains relatively stable across the 20 projects. We observe that Project-14 ("Situated-Empathy-in-VR") and Project-25 ("Procrastination-VR") show exceptionally high CI, standing out significantly from other projects. Upon further investigation, we find that these two projects have the highest rca values (75.60% and 91.63%) among all projects, while their rcg values are the lowest (2.87% and 2.51%), meaning that they possess the most cloned files but the fewest clone groups. This corresponds to the case where a few files are heavily cloned, leading to the emergence of some exceptionally large clone groups. Thus, we can deduce that CI is jointly influenced by rca and rcg, where rca shows a positive correlation and rcg a negative correlation.

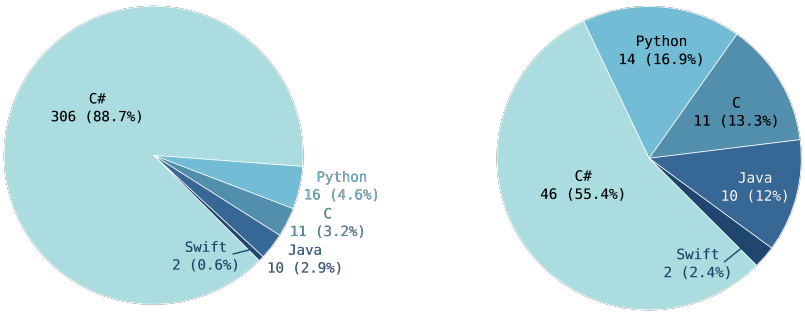
Insight 4. *The proposed metrics measure cloning from distinct perspectives: the number or ratio of clone functions (files) measures the quantity of clones, while the number or ratio of clone classes (groups) evaluates clone concentration. By combining these metrics, a comprehensive clone evaluation can be achieved.*

4.2 Source Code Cloning Analysis

4.2.1 RQ4: Which programming languages dominate VR software development and are more susceptible to code cloning?

To answer this RQ, we visualize the programming languages used by these software projects in Figure 5. The left pie chart shows the programming language distribution for all VR-345 projects, while the right highlights filtered high-star projects with star >= 200. In this context, the programming language mentioned for a project refers to the primary language used within the project.

We observe that C# dominates general VR projects, while it accounts for slightly over half in high-star projects. This is due to the fact that high-star projects not only feature engine-based applications but also include a substantial number of development tool projects, which are typically developed in various programming languages and serve as support for other projects. The advantages of C# language are most likely attributed to its close integration with the Unity game engine, which has emerged as one of the leading tools in VR development due to its ease of use and cross-platform



(a) All VR-345 projects.

(b) 83 high-star projects.

Fig. 5: Distribution of programming languages in VR projects.

features. Given that C# serves as the primary programming language for Unity, it is unsurprising that it has become the most widely adopted language for VR projects. Other languages employed in VR development include Python, C, Java, and Swift. In contrast, languages such as PHP, Ruby, WSDL, and ATL are absent. This is reasonable because web-oriented languages like PHP, Ruby, and WSDL are not well-suited for graphics-intensive applications due to limitations in execution efficiency and performance, while ATL is primarily designed for Windows-specific programming. It is also worth noting that JavaScript plays a significant role in VR project development. However, its absence in this analysis stems from a limitation in NiCad, which currently does not support clone detection for JavaScript.

To further examine which programming languages are more prone to code cloning, we identify projects with 10 or more clone classes (NCC$>=$ 10) in terms of Clone Type 3-2 within high-star VR projects and categorize these projects by their primary programming languages. Table 10 presents the distribution of projects across various languages at different cloning levels. The results reveal that C# accounts for the highest number of projects (18) with over 10 clone classes, followed by C with 5 projects. This finding aligns with the prominent role of C# as the most commonly used language in VR software development. However, when considering the cloning ratio, C ($5/11 \approx 45.5\%$) surpasses C# ($18/46 \approx 39.1\%$), indicating that C may have a higher likelihood of triggering cloning issues. This may stem from the fact that C, being process-oriented, does not support the code reuse mechanisms inherent to object-oriented languages, so developers might inadvertently copy blocks of code instead of abstracting them.

Insight 5. *While C# is the most commonly used programming language in open-source VR projects, C-based projects exhibit a higher proportion of code cloning.*

Table 10: The distribution of projects across different cloning intervals with regard to NCC using various languages.

Language \ NCC Interval	[10, 30]	[31, 50]	[51, 100]	> 100	Sum
Java	1	0	0	0	1 (out of 10)
Python	3	0	0	1	4 (out of 14)
C	1	1	1	2	5 (out of 11)
C#	7	7	2	2	18 (out of 46)

Table 11: The list of analyzed third-party libraries.

Library	Type	Key Files	Exist	Clone
SteamVR	VR Hardware SDK	`Assets/SteamVR/Input/SteamVR_Input.cs`	✓	✓
Oculus	VR Platform SDK	`Assets/Oculus/VR/Scripts/OVRManager.cs`	✓	✓
OpenVR	VR Runtime	`Assets/Plugins/openvr/api.cs`	✓	×
Vive	Hardware Interaction SDK	`Assets/SteamVR/InteractionSystem/Core/Interfaces/Hand.cs`	✓	×
Google VR	Mobile VR SDK	`Assets/GoogleVR/Legacy/Scripts/GvrViewer.cs`	✓	×
VRTK	VR Framework	`Assets/VRTK/SDK/Base/Scripts/VRTK_SDKManager.cs`	✓	✓
Unity XR	Cross-Platform SDK	`Assets/XR/Input/XRController.cs`	×	
OpenXR	Cross-Vendor Runtime	`Assets/XR/OpenXR/Settings/OpenXRSettings.asset`	×	
OSVR	VR Platform	`Assets/OSVR/ClientKit/OSVR_ClientContext.cs`	×	
Pico SDK	VR SDK	`Assets/PicoMobileSDK/Scripts/PicoVR_Manager.cs`	×	
Wave SDK	VR Platform	`Assets/WaveVR/Scripts/WaveVR_Reticle.cs`	×	
Varjo Base	Enterprise VR SDK	`Assets/Varjo/Scripts/VarjoManager.cs`	×	
Ultraleap	Gesture SDK	`Assets/Ultraleap/Hands/LeapHandController.cs`	×	
WebXR	Browser-Based VR	`Assets/WebXR/Plugins/WebXRInterface.cs`	×	

4.2.2 RQ5: How do third-party libraries impact code cloning in VR software?

To answer this RQ, we examine 20 application projects to identify third-party libraries with code clones. We initially identify 14 widely used libraries, 6 of which are found in our dataset, with 3—SteamVR, Oculus, and VRTK—showing cloning issues. The specific procedure is as follows: (i) we identify 14 widely used third-party libraries in VR development for study, as shown in Table 11; (ii) we verify whether each of these 14 libraries is integrated in any of the 20 projects by checking for the existence of their distinctive key files, resulting in 6 libraries being present; (iii) we then detect whether the remaining libraries have code clones, identifying 3 libraries—SteamVR, Oculus, and VRTK—with cloning issues; and (iv) we review the cloning results to ensure all libraries with clones are properly examined.

Upon reviewing our results, we find that SteamVR appears in 10 projects, while Oculus and VRTK each appear in 4 projects. However, when it comes to introducing source code clones, SteamVR has the lowest contribution, Oculus ranks in the middle, and VRTK contributes the most. This is reasonable because VRTK, as an open-source toolkit, supports numerous VR devices and platforms, providing high-level abstractions that simplify development and lead to significant code redundancy. On the other hand, SteamVR and Oculus are development kits designed for specific hardware, prioritizing compatibility and low-level optimization, which results in less cross-platform or high-level redundant code.

Table 12: The evaluation results of third-party libraries in the projects.

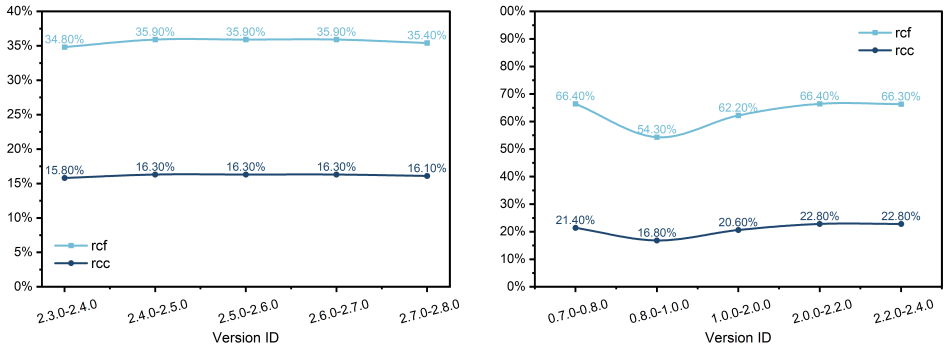
PID	Project			Library				
	Name	NCF	NCC	Name	NCF	Per	NCC	Per
11	SoundSpace	2225	965	SteamVR	70	3.17%	26	2.69%
12	RhythmAttack-VR	330	119	SteamVR	29	73.64%	8	68.07%
				Oculus	54		13	
				VRTK	160		60	
13	Dungeon-VR	254	98	SteamVR	29	57.87%	8	42.86%
				VRTK	81		34	
15	mineRVa	197	71	SteamVR	33	97.46%	8	94.37%
				VRTK	158		59	
16	VR-Escape-Room	120	46	SteamVR	29	91.67%	8	91.30%
				VRTK	81		34	
17	Group6_ProjectNurture	248	82	Oculus	222	89.52%	69	84.15%
19	Terminal	160	49	SteamVR	22	56.88%	6	46.94%
				Oculus	69		17	
20	elite-vr-cockpit	109	39	SteamVR	83	76.15%	28	71.79%
21	VRHamsterBall	74	28	SteamVR	66	89.19%	24	85.71%
22	OpendagVR2	45	16	SteamVR	29	64.44%	8	50.00%
23	epicslash	54	19	SteamVR	26	48.15%	7	36.84%
26	AirAttack	94	30	Oculus	54	57.45%	13	43.33%

These findings confirm the presence of code clones caused by third-party libraries in VR software. Such clones are not introduced by the developers but are intrinsic to the third-party libraries themselves. For instance, in the three third-party libraries identified in our analysis, different projects display the same cloning results, such as (NCF:29, NCC:8) of SteamVR in Project-12, 13, 16, 22, (NCF:54, NCC:13) of Oculus in Project-12, 26, and (NCF:81, NCC:34) of VRTK in Project-13, 16. These clones are triggered by the most commonly used code in these libraries, with additional clones being introduced by project-specific features. Although developers cannot modify clones in third-party tools, they can minimize them by using dependency injection, interface-based techniques, or opting for lightweight libraries when cross-platform support is unnecessary.

Insight 6. *Third-party libraries introduced in VR applications often contribute to source code cloning issues, sometimes constituting the majority of the cloning results. The more functionalities the third-party libraries provide, the higher the clone evaluation metrics tend to be.*

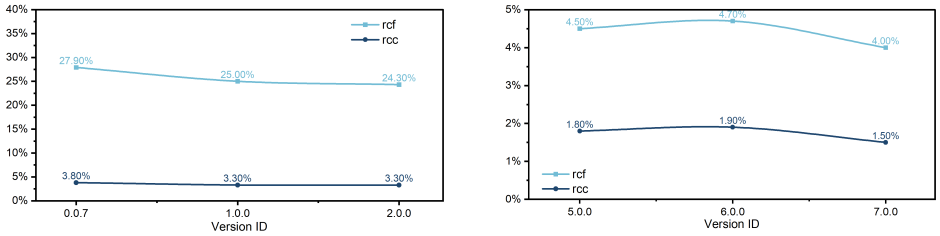
4.2.3 RQ6: How do intra-version and inter-version code cloning evolve across different versions of a VR project?

To answer this RQ, we begin by examining whether code clones exist between different versions of the same project. To achieve this, we select two projects with a high number of versions and a wide range between versions for analysis. Specifically, We execute code clone detection on six sequential versions of each project, with the detection carried out between each pair of adjacent versions, yielding five data sets per project. The experimental results are illustrated in Figure 6.



(a) open-brush (b) gpac

Fig. 6: Detection results of inter-version code clones on five adjacent version pairs of two projects.

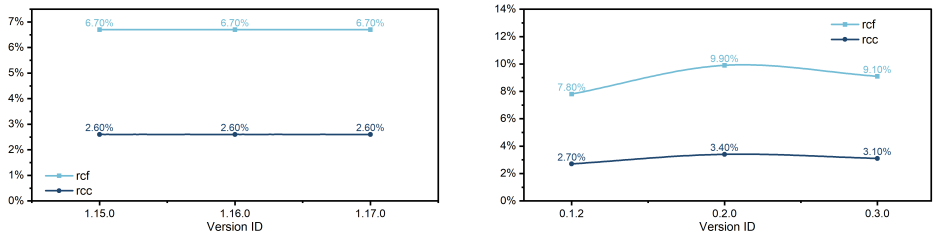


(a) gpac (b) UnityPlugin

Fig. 7: Detection results of intra-version clones across multiple versions when major version number changes.

We find that in the "open-brush" project, the rcc and rcf values fall within the ranges of [0.158, 0.163] and [0.348, 0.359], respectively. In the "gpac" project, these values span wider ranges, with rcc between [0.168, 0.228] and rcf between [0.543, 0.664]. These observations suggest that while the degree of inter-version code cloning varies across projects, it remains consistently high, underscoring the prevalence of inter-version code clones in VR software development. This trend highlights that while code reuse improves development efficiency, it also poses the risk of propagating vulnerabilities, which may compromise software stability and security in future versions.

To examine intra-version code clones, we divide the analysis into three sub-RQs focusing on their evolution with changes in (i) major version numbers (e.g., 0.x.x, 1.x.x, 2.x.x), (ii) secondary version numbers (e.g., 0.1.x, 0.2.x, 0.3.x), and (iii) ending version numbers (e.g., 0.1.1, 0.1.2, 0.1.3). For each sub-RQ, we select two distinct projects and analyze their latest three eligible versions. These projects are chosen for their frequent release schedules, which provide a robust dataset for analysis.



(a) ViveInputUtility-Unity (b) com.xrtk.core

Fig. 8: Detection results of intra-version clones across multiple versions when secondary version number changes.

For *major version changes*, Figure 7a and Figure 7b illustrate how rcf and rcc evolve with major version changes in the VR projects "gpac" and "UnityPlugin", respectively. In the project "gpac", rcf fluctuates by up to 0.036 and rcc by up to 0.005 across three major versions. Similarly, in the project "UnityPlugin", rcf shows a variation of up to 0.007, and rcc up to 0.004. These observations suggest that the extent of code clones, as measured by rcf and rcc, remains relatively stable during major version updates for VR projects.

For *secondary version changes*, Figure 8a and Figure 8b show the evolution of rcf and rcc as the secondary version numbers change in the projects "ViveInputUtility-Unity" and "com.xrtk.core", respectively. We observe that in the project "ViveInputUtility-Unity", rcf and rcc remain constant across the three analyzed versions. In the project "com.xrtk.core", rcf fluctuates by up to 0.021, and rcc by up to 0.007. This indicates that the level of code clones generally remains stable, or even unchanged, during secondary version updates.

For *ending version changes*, we do not present figures because the variations in rcf and rcc across such updates are negligible. The minimal differences suggest that code clone patterns exhibit almost no significant evolution at this level of versioning.

Insight 7. *Inter-version code cloning is more common than intra-version code cloning, introducing significant security risks due to the potential propagation of vulnerabilities across multiple versions of the software.*

4.3 Asset File Cloning Analysis

4.3.1 RQ7: How do cloning practices vary across different types of asset files in VR software?

To answer this RQ, we select four representative projects from distinct application categories and conduct clone detection on the serialized files within each project, with results illustrated in Figure 9.

We observe a considerable disparity between scene files and material files, with scene files typically exhibiting the lowest clone level, averaging CI $\approx$ 0.1125, while material files demonstrate the highest, with an average of CI $\approx$ 1.05. This is possibly

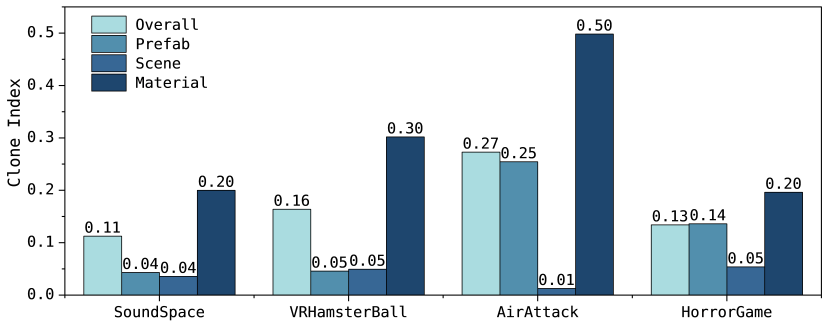


Fig. 9: Analysis of cloning behavior of various asset files from four typical projects.

due to the fact that VR scenes generally contain unique content and structures that are difficult to replicate or share. Conversely, material files are extensively reused across various objects and scenes for optimization and performance reasons.

Additionally, we find that the clone level of prefab files is intermediate, with an average of $\mathsf{CI} \approx 0.375$, yet it shows substantial fluctuations. This may be attributed to variations in system design and resource reuse strategies across different projects, which are heavily influenced by the developer's style. Some projects may require complex, customized prefabs, while others rely on a large number of standardized prefabs that are heavily reused. The use of techniques like prefab variants and runtime instantiation in complex prefabs, analogous to object-oriented programming, can lead to a reduced CI. In contrast, extensive reuse of standardized prefabs results in a higher CI, facilitating development but introducing redundancy, as changes to attributes need to be applied to every file within the clone group.

Insight 8. *In VR software, the cloning behavior of asset files varies with complexity. Complex files like scenes have lower clone levels, simpler ones like materials have higher, and prefab files fluctuate with development style.*

5 Discussion

5.1 Recommendations

5.1.1 For Researchers

This study yields several actionable implications for researchers studying software cloning in VR development. First, the strong correlation between project size and cloning suggests that scalable clone detection tools are necessary for large VR projects, particularly in the gaming and education domains where cloning is most prevalent. Second, the distinct reliance on serialized asset files rather than source code highlights the importance of developing specialized detection methods tailored to asset-based cloning. Third, security-focused research should particularly investigate inter-version cloning patterns that facilitate vulnerability propagation. Finally, to advance clone

analysis in VR software, researchers should focus on creating context-aware, multi-faceted metrics that capture the unique structural and behavioral features of VR systems. This effort should be supported by a comprehensive evaluation framework combining empirical validation, contextual experimentation, and cross-scenario applicability testing to ensure both precision and practical impact.

5.1.2 For VR Software Developers and Practitioners

This study also yields several actionable implications for VR software developers and practitioners. First, employ robust design patterns and modular architectures to facilitate code reuse and minimize reliance on copy-paste practices, particularly in large-scale VR projects where code and asset cloning is more prevalent.

Second, the integration of clone detection tools into development workflows is essential for the early identification and refactoring of duplicated content, thereby mitigating long-term maintenance burdens in increasingly complex systems.

Third, asset management should be treated with the same rigor as source code maintenance. Rather than generating slight variations of materials or textures, developers are encouraged to establish a centralized asset repository and adopt reusable instances or prefabs to ensure consistency and reduce redundancy across scenes.

Fourth, development practices should align with the characteristics of the chosen programming language. In C#, for instance, developers can leverage ScriptableObjects and inheritance to encapsulate reusable logic, whereas in C or C++, repetitive tasks should be delegated to optimized libraries or engine-level modules instead of being implemented repeatedly.

Fifth, evaluate third-party libraries carefully and consider alternatives with lower cloning levels to minimize clone. Developers should consider trimming or replacing libraries with high clone rates, especially when only a small subset of features is required. When robust frameworks are indispensable (e.g., Oculus SDK or SteamVR), their integration should be abstracted via dependency injection or interface wrappers to avoid deeply embedded code dependencies and facilitate easier maintenance.

Finally, to address the risks of inter-version cloning, teams should establish cross-version code management workflows. This includes centralizing reusable logic in shared core modules and maintaining traceability of duplicated segments to ensure timely propagation of fixes and enhancements across all project versions.

5.2 Limitation and Future Work

The present study has several limitations that need to be addressed in future research. First, the scope of our dataset presents certain constraints that may affect the generalizability of our findings. (i) Currently, the dataset primarily consists of C# projects, aligning with the current landscape of open-source VR development. Although the exclusion of non-NiCad-supported languages such as JavaScript affects only a small subset of auxiliary tools, extending language support is essential for enhancing generalizability—particularly in multilingual and proprietary VR contexts. (ii) Another key limitation of our study lies in the exclusion of Unreal Engine-based projects, despite their prominence in VR development. This exclusion stems from two primary factors:

most Unreal Engine-based VR projects are proprietary and not publicly accessible as open-source code; and such projects are predominantly developed in C++, which is not supported by our clone detection tool, NiCad. These constraints collectively prevent the inclusion of Unreal Engine projects in our dataset and may limit the generalizability of our findings to closed-source VR ecosystems. We suggest that future research explore alternative analysis methods or tools capable of supporting C++ and binary formats, thereby broadening the scope of VR project analysis.

Second, the employment of GPT-4o for file clone detection presents several limitations that warrant discussion. (i) Different versions or configurations of the GPT-4o can sometimes produce varying similarity values, highlighting the need for consistent model configurations and fine-tuning to reduce discrepancies in similarity tasks. (ii) Although our study illustrates the effectiveness of GPT-4o in asset file clone detection for open-source VR software, its scalability and real-time performance in large-scale VR systems have not yet been evaluated, and current LLMs often underperform in such settings. In addition, while fine-tuned LLMs could improve detection precision for serialized assets, the absence of domain-specific datasets currently hinders such efforts. To address these issues, our future work will focus on building dedicated asset file corpora and training customized models. (iii) The application of LLMs in code and file clone detection within VR software introduces ethical concerns, particularly regarding data privacy and inference bias. While our reliance on open-source datasets mitigates these concerns in the present study, future research targeting proprietary VR projects or user-contributed content should adopt more robust privacy-preserving and fairness-aware practices.

Another limitation of this study lies in its absence of VR-specific clone metrics. The adopted metrics are derived from traditional software research and remain overly straightforward. Given the complexity and unique characteristics of VR environments, there exists a pressing need to establish more sophisticated, VR context-aware metrics that correspond to varied analytical objectives, coupled with systematic evaluations encompassing strengths, limitations, verification, and scenario-specific applicability.

Finally, our current evaluation methodology examines source code and serialized asset cloning in isolation. An important research direction would involve developing an integrated assessment framework that simultaneously analyzes both artifact types. This may be achieved through the design of a specialized LLM-based analysis agent capable of cross-artifact pattern recognition, unified clone behavior modeling, and combined metric computation—thereby enabling truly holistic VR software clone analysis.

6 Related Work

6.1 Empirical Study of VR Software

The empirical study of VR software has gained increased attention in recent years, and the scope includes educational [73], marketing communication [74], user operational performance [75], automated testing [76], vulnerabilities [65, 77, 78] and code cloning [17]. Among the studies, Our prior work [17] provides the most relevant research on

code cloning in open-source VR software. However, the study does not address the code cloning of serialized files, which represents an important aspect of VR software.

6.2 Code Clone Detection

Traditional methods for detecting code clones include text-based, token-based, and AST-based techniques. Tools like Simian [26], CCfinder [13, 79], and NiCad [15, 30] focus on comparing code at the syntactic or token level, while others like CloneDR [33, 34] and Duplication Finder [80, 81] leverage the Abstract Syntax Tree for deeper analysis of code structure.

Recent advancements have introduced the use of Large Language Models (LLMs), such as Codex [82, 83] and CodeBERT [41, 42], to detect clones with semantic awareness. These models understand code functionality, allowing them to identify clones that may not be syntactically identical but are semantically equivalent. By leveraging LLMs, clone detection can move beyond superficial matches, improving accuracy in detecting complex clones and offering greater flexibility across different programming languages. Recent research [45] shows that NiCad performs best in non-LLMs-based detection, while GPT-4 performs best in LLMs-based detection. Both of them achieve the highest precision and recall for each clone type in the same type of comparison.

6.3 File Clone Detection

Asset files in VR software, representing the differences from traditional software, can be considered specialized text files. There are several methods to detect their similarity, with some of the more representative ones being content-based methods, fingerprint-based methods, and semantic (deep learning) methods.

Content-based detection methods [48–51] focus on directly comparing the file contents using traditional techniques such as string matching and text block comparison. However, these methods tend to be less efficient when dealing with large-scale systems. Fingerprint-based detection methods [53, 54] improve efficiency by generating fingerprints for the files and using efficient lookup algorithms, making them particularly suitable for large-scale source code repositories. However, they cannot detect clones caused by syntax or structural changes. Deep learning-based detection methods [56–59] rely on neural networks to model the semantics of the code or text. These methods can identify clones that differ in structure but are semantically similar, offering strong adaptability and high accuracy. However, they come with greater implementation complexity and computational overhead. Nevertheless, with the recent development of LLM technology and architecture, the disadvantages of deep learning-based detection methods have been alleviated to a certain extent, making it an advantageous method for similarity detection of serialized text files.

7 Conclusion

In this paper, we conduct a quantitative empirical study on code cloning in open-source VR projects. We refine the VR-345 dataset from previous studies for this research, ensuring a balance between the influence and range of the projects. We then propose a

set of metrics to measure code cloning from different perspectives. Finally, we perform empirical experiments on seven carefully designed research questions at three distinct levels, in order to capture an overview of code cloning in VR software.

Our study reveals the key differences between VR software and traditional software in terms of code cloning. Due to these differences, certain files cannot be detected by traditional detection tools, which is why we employ large language model technology to assist in our detection. We evaluate and analyze the experimental results from various aspects such as clone distribution, clone concentration, programming languages, third-party libraries, and the impact of different types of assets. Through this, we answer the seven proposed research questions, deepening our understanding of code cloning in VR software.

References

[1] Anthes, C., García-Hernández, R.J., Wiedemann, M., Kranzlmüller, D.: State of the art of virtual reality technology. In: 2016 IEEE Aerospace Conference, pp. 1–19 (2016). IEEE

[2] Berg, L.P., Vance, J.M.: Industry use of virtual reality in product design and manufacturing: a survey. Virtual reality **21**, 1–17 (2017)

[3] Kamińska, D., Sapiński, T., Wiak, S., Tikk, T., Haamer, R.E., Avots, E., Helmi, A., Ozcinar, C., Anbarjafari, G.: Virtual reality and its applications in education: Survey. Information **10**(10), 318 (2019)

[4] Virtual Reality Statistics: The Ultimate List in 2024. https://academyofanimatedart.com/virtual-reality-statistics/. (Accessed on 08/10/2024)

[5] Juergens, E., Deissenboeck, F., Hummel, B., Wagner, S.: Do code clones matter? In: 2009 IEEE 31st International Conference on Software Engineering, pp. 485–495 (2009). IEEE

[6] Chatterji, D., Carver, J.C., Kraft, N.A., Harder, J.: Effects of cloned code on software maintainability: A replicated developer study. In: 2013 20th Working Conference on Reverse Engineering (WCRE), pp. 112–121 (2013). IEEE

[7] Lozano, A., Wermelinger, M.: Assessing the effect of clones on changeability. In: 2008 IEEE International Conference on Software Maintenance, pp. 227–236 (2008). IEEE

[8] Duala-Ekoko, E., Robillard, M.P.: Clonetracker: tool support for code clone management. In: Proceedings of the 30th International Conference on Software Engineering, pp. 843–846 (2008)

[9] Rajakumari, K.E.: Towards a novel conceptual framework for analyzing code clones to assist in software development and software reuse. In: 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS), pp.

105–111 (2020). IEEE

[10] Mondal, S., Mondal, M., Debnath, R.: Granularity-based comparison of the bug-proneness of code clones. In: 2023 IEEE 17th International Workshop on Software Clones (IWSC), pp. 8–14 (2023). IEEE

[11] Islam, M.R., Zibran, M.F., Nagpal, A.: Security vulnerabilities in categories of clones and non-cloned code: An empirical study. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 20–29 (2017). IEEE

[12] Solanki, K., Kumari, S.: Comparative study of software clone detection techniques. In: 2016 Management and Innovation Technology International Conference (MITicon), p. 152 (2016). IEEE

[13] Kamiya, T., Kusumoto, S., Inoue, K.: Ccfinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE transactions on software engineering **28**(7), 654–670 (2002)

[14] White, M., Tufano, M., Vendome, C., Poshyvanyk, D.: Deep learning code fragments for code clone detection. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pp. 87–98 (2016)

[15] Cordy, J.R., Roy, C.K.: The nicad clone detector. In: 2011 IEEE 19th International Conference on Program Comprehension, pp. 219–220 (2011). IEEE

[16] Feng, C., Wang, T., Liu, J., Zhang, Y., Xu, K., Wang, Y.: Nicad+: Speeding the detecting process of nicad. In: 2020 IEEE International Conference on Service Oriented Systems Engineering (SOSE), pp. 103–110 (2020). IEEE

[17] Huang, W., Chen, J., Chen, H., Qi, Z., Yang, X., Peng, K., He, S.: A study of code clone on open source vr software. In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering Workshops, pp. 239–244 (2024)

[18] Technologies, U.: Unity User Manual. (2024). https://docs.unity3d.com/Manual/index.html

[19] Goldstone, W.: Unity 2021 Game Development Essentials. Packt Publishing, Birmingham, UK (2021)

[20] Linowes, J.: Unity Virtual Reality Projects. Packt Publishing, Birmingham, UK (2021)

[21] Roy, C.K., Cordy, J.R., Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Science of computer programming **74**(7), 470–495 (2009)

[22] Rattan, D., Bhatia, R., Singh, M.: Software clone detection: A systematic review. Information and Software Technology **55**(7), 1165–1199 (2013)

[23] Ain, Q.U., Butt, W.H., Anwar, M.W., Azam, F., Maqbool, B.: A systematic review on code clone detection. IEEE access **7**, 86121–86144 (2019)

[24] Sajnani, H., Saini, V., Svajlenko, J., Roy, C.K., Lopes, C.V.: Sourcerercc: Scaling code clone detection to big-code. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 1157–1168 (2016). https://doi.org/10.1145/2884781.2884877

[25] Banda, B., Bahadur, S.: Code clone detection and analysis using software metrics and neural network-a literature review. (2015). https://api.semanticscholar.org/CorpusID:1993733

[26] Consulting, R.H.: Simian: A tool for detecting duplicate code fragments. http://www.redhillconsulting.com.au/products/simian. Accessed: 2024-12-04

[27] Chapman, P.: Duplo - Duplicate Code Detection Tool. https://github.com/pmachapman/duplo. Accessed: 2024-12-12 (2005)

[28] Sheneamer, A.M., Kalita, J.K.: A survey of software clone detection techniques. International Journal of Computer Applications **137**, 1–21 (2016)

[29] Agrawal, A., Yadav, S.K.: A hybrid-token and textual based approach to find similar code segments. In: 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), pp. 1–4 (2013). https://doi.org/10.1109/ICCCNT.2013.6726700

[30] NiCad Clone Detector. http://www.txl.ca/txl-nicaddownload.html. (Accessed on 08/10/2024)

[31] Ain, Q.U., Butt, W.H., Anwar, M.W., Azam, F., Maqbool, B.: A systematic review on code clone detection. IEEE Access **7**, 86121–86144 (2019) https://doi.org/10.1109/ACCESS.2019.2918202

[32] Zakeri-Nasrabadi, M., Parsa, S., Ramezani, M., Roy, C., Ekhtiarzadeh, M.: A systematic literature review on source code similarity measurement and clone detection: techniques, applications, and challenges (2023). https://arxiv.org/abs/2306.16171

[33] O'Callahan, R., Zeldman, S.: CloneDR: A tool for detecting source code clones in the software maintenance process. https://www.rstcorp.com/products/clonedr/. (Accessed on 08/10/2024)

[34] O'Callahan, R., Zeldman, S.: Clonedr: A tool for detecting source code clones

in the software maintenance process. In: Proceedings of the 5th European Conference on Software Maintenance and Reengineering (CSMR 2003), pp. 190–199. IEEE, New York, USA (2003). https://doi.org/10.1109/CSMR.2003.1193494 . https://ieeexplore.ieee.org/document/1193494

[35] Jiang, L., Misherghi, G., Su, Z., Glondu, S.: Deckard: Scalable and accurate tree-based detection of code clones. In: Proceedings of the 29th International Conference on Software Engineering (ICSE), pp. 96–105. IEEE, New York, USA (2007). https://doi.org/10.1109/ICSE.2007.30

[36] Sheneamer, A., Roy, S., Kalita, J.: A detection framework for semantic code clones and obfuscated code. Expert Systems with Applications **97**, 405–420 (2018) https://doi.org/10.1016/j.eswa.2017.12.040

[37] Zhang, W., Guo, S., Zhang, H., Sui, Y., Xue, Y., Xu, Y.: Challenging machine learning-based clone detectors via semantic-preserving code transformations. IEEE Transactions on Software Engineering **49**(5), 3052–3070 (2023) https://doi.org/10.1109/tse.2023.3240118

[38] Jiang, L., Gabel, M., Su, Z.: Scalable detection of semantic clones. In: Proceedings of the 30th International Conference on Software Engineering (ICSE), pp. 321–330. ACM, New York, NY, USA (2008). https://doi.org/10.1145/1368088.1368132 . https://dl.acm.org/doi/10.1145/1368088.1368132

[39] Walker, A., Cerny, T., Song, E.: Open-source tools and benchmarks for code-clone detection: past, present, and future trends. SIGAPP Appl. Comput. Rev. **19**(4), 28–39 (2020) https://doi.org/10.1145/3381307.3381310

[40] Majdinasab, V., Nikanjam, A., Khomh, F.: Trained without my consent: Detecting code inclusion in language models trained on code. arXiv preprint arXiv:2402.09299 (2024)

[41] Feng, Y., Xie, T.: CodeBERT: A Pre-Trained Model for Programming and Natural Languages. https://arxiv.org/abs/2002.08155. (Accessed on 08/10/2024) (2020)

[42] Feng, Y., Xie, T.: Codebert: A pre-trained model for programming and natural languages. In: Proceedings of the 43rd International ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020), pp. 1–15. ACM, New York, USA (2020). https://doi.org/10.1145/3385412.3386001 . https://dl.acm.org/doi/10.1145/3385412.3386001

[43] Svajlenko, J., Roy, C.K.: Evaluating clone detection tools with bigclonebench. In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 131–140. IEEE, Piscataway, NJ, USA (2015). https://doi.org/10.1109/ICSM.2015.7332459

[44] Xu, X., Ni, C., Guo, X., Liu, S., Wang, X., Liu, K.: Distinguishing llm-generated from human-written code by contrastive learning. arXiv preprint arXiv:2411.04704 (2024)

[45] Dou, S., Shan, J., Jia, H., Deng, W., Xi, Z., He, W., Wu, Y., Gui, T., Liu, Y., Huang, X.: Towards Understanding the Capability of Large Language Models on Code Clone Detection: A Survey (2023). https://arxiv.org/abs/2308.01191

[46] Kabore, A.K., Klein, J.: Cross-lingual code clone detection: When llms fail short against embedding-based classifier. Proceedings of the ACM International Symposium on Software Engineering (ICSE) (2024)

[47] Khajezade, M., Wu, J., Fard, F.H.: Investigating the efficacy of large language models for code clone detection. Proceedings of the ACM Symposium on Software Engineering and Analysis (2024)

[48] Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill, New York, NY, USA (1983)

[49] Jones, K.S.: A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation **28**(1), 11–21 (1972)

[50] Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady **10**(8), 707–710 (1966)

[51] Jaccard, P.: Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines. Bulletin of the Society of Neuchâtel Sciences Naturelles **25**, 213–220 (1901)

[52] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. https://scikit-learn.org/. Accessed: 2024-12-13 (2011)

[53] Huang, Q., Feng, J., Zhang, Y., Fang, Q., Ng, W.: Query-aware locality-sensitive hashing for approximate nearest neighbor search. Proc. VLDB Endow. **9**(1), 1–12 (2015) https://doi.org/10.14778/2850469.2850470

[54] Rivest, R.L.: The MD5 Message-Digest Algorithm. RFC 1321 (1992). https://www.rfc-editor.org/rfc/rfc1321

[55] Duplicate File Finder. https://www.duplicatefilefinder.com/. Accessed: 2024-12-13 (2024)

[56] Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Conference on Empirical Methods in Natural Language Processing (2014). https://api.semanticscholar.org/CorpusID:1957433

[57] Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching Word Vectors with Subword Information (2017). https://arxiv.org/abs/1607.04606

[58] Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations (2018). https://arxiv.org/abs/1802.05365

[59] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2019). https://arxiv.org/abs/1810.04805

[60] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Brew, J.: Hugging Face Transformers. https://huggingface.co/transformers. Accessed: 2024-12-13 (2020)

[61] Liu, Y., Zhang, Z., Li, W., Xu, X.: Enhancing text similarity detection with gpt-4: A comparative study. arXiv preprint arXiv:2305.05656 (2023)

[62] Zhang, W., Li, J., Liu, Z.: Gpt-4 vs. t5 in text similarity tasks: A comparative study. arXiv preprint arXiv:2301.01947 (2023)

[63] Rzig, D.E., Iqbal, N., Attisano, I., Qin, X., Hassan, F.: Virtual Reality (VR) Automated Testing in the Wild: A Case Study on Unity-Based VR Applications. In: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 1269–1281. ACM, Seattle WA USA (2023). https://doi.org/10.1145/3597926.3598134

[64] Nusrat, F., Hassan, F., Zhong, H., Wang, X.: How developers optimize virtual reality applications: A study of optimization commits in open source unity projects. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 473–485 (2021). IEEE

[65] Rodriguez, I., Wang, X.: An empirical study of open source virtual reality software projects. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 474–475 (2017). IEEE

[66] Nusrat, F., Hassan, F., Zhong, H., Wang, X.: How Developers Optimize Virtual Reality Applications: A Study of Optimization Commits in Open Source Unity Projects. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 473–485. https://doi.org/10.1109/ICSE43902.2021.00052 . https://ieeexplore.ieee.org/document/9402052/?arnumber=9402052 Accessed 2024-11-01

[67] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. In: Proceedings of the 36th International Conference on Neural Information Processing Systems. NIPS '22. Curran Associates Inc., Red Hook, NY, USA

(2024)

[68] Fu, Y., Peng, H., Sabharwal, A., Clark, P., Khot, T.: Complexity-Based Prompting for Multi-Step Reasoning (2023). https://arxiv.org/abs/2210.00720

[69] Li, Y., Lin, Z., Zhang, S., Fu, Q., Chen, B., Lou, J.-G., Chen, W.: Making Large Language Models Better Reasoners with Step-Aware Verifier (2023). https://arxiv.org/abs/2206.02336

[70] Zhang, Z., Zhang, A., Li, M., Zhao, H., Karypis, G., Smola, A.: Multimodal Chain-of-Thought Reasoning in Language Models (2024). https://arxiv.org/abs/2302.00923

[71] Roy, C.K., Cordy, J.R.: A survey on software clone detection research. Computer Science Review **2**(3), 137–158 (2008)

[72] Kapser, C., Godfrey, M.W.: "cloning considered harmful" considered harmful. In: IEEE International Conference on Software Maintenance, pp. 267–276 (2006)

[73] Oyelere, S.S., Bouali, N., Kaliisa, R., Obaido, G., Yunusa, A.A., Jimoh, E.R.: Exploring the trends of educational virtual reality games: A systematic review of empirical studies. Smart Learning Environments **7**(1), 31 (2020) https://doi.org/10.1186/s40561-020-00142-7 . Accessed 2024-11-26

[74] Filip, G., Marcin, A., Grzegorz, M., Katarzyna, P.: Virtual Reality in Marketing Communication – the Impact on the Message, Technology and Offer Perception – Empirical Study. Economics and Business Review **4**(3), 36–50 (2018) https://doi.org/10.18559/ebr.2018.3.4

[75] Epp, R., Lin, D., Bezemer, C.-P.: An empirical study of trends of popular virtual reality games and their complaints **13**(3), 275–286. Accessed 2024-11-26

[76] Rzig, D.E., Iqbal, N., Attisano, I., Qin, X., Hassan, F.: Virtual reality (vr) automated testing in the wild: A case study on unity-based vr applications. In: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2023, pp. 1269–1281. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3597926.3598134 . https://doi.org/10.1145/3597926.3598134

[77] Dastgerdy, S.: Virtual Reality and Augmented Reality Security: A Reconnaissance and Vulnerability Assessment Approach. arXiv (2024). https://doi.org/10.48550/arXiv.2407.15984

[78] Guo, H., Dai, H.-N., Luo, X., Zheng, Z., Xu, G., He, F.: An empirical study on oculus virtual reality applications: Security and privacy perspectives. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, pp. 1–13 (2024)

[79] Kamiya, T., Kusumoto, S., Inoue, K.: Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. IEEE Transactions on Software Engineering **28**(7), 654–670 (2002) https://doi.org/10.1109/TSE.2002.1019480

[80] Garcia, L.M., Roussel, G.: Duplication Finder: A tool for finding duplication in large-scale software systems. https://www.ruby-lang.org/en/. (Accessed on 08/10/2024)

[81] Garcia, L.M., Roussel, G.: Duplication finder: A tool for finding duplication in large-scale software systems. In: Proceedings of the 10th Working Conference on Reverse Engineering (WCRE 2009), pp. 181–190. IEEE, New York, USA (2009). https://doi.org/10.1109/WCRE.2009.55 . https://ieeexplore.ieee.org/document/5361435

[82] Chen, M., Tworek, J., Jun, H.e.a.: Evaluating Large Language Models Trained on Code. https://openai.com/blog/evaluating-large-language-models-trained-on-code. (Accessed on 08/10/2024) (2021)

[83] Chen, M., Tworek, J., Jun, H., *et al.*: Evaluating large language models trained on code. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP 2021) **2021**, 1–15 (2021) https://doi.org/10.18653/v1/2021.emnlp-main.183