Searching bug instances in gameplay video repositories

Mohammad Reza Taesiri, Finlay Macklon, Sarra Habchi, and Cor-Paul Bezemer

Abstract—Gameplay videos offer valuable insights into player interactions and game responses, particularly data about game bugs. Despite the abundance of gameplay videos online, extracting useful information remains a challenge. This paper introduces a method for searching and extracting relevant videos from extensive video repositories using English text queries. Our approach requires no external information, like video metadata; it solely depends on video content. Leveraging the zero-shot transfer capabilities of the Contrastive Language-Image Pre-Training (CLIP) model, our approach does not require any data labeling or training. To evaluate our approach, we present the `GamePhysics` dataset, comprising 26,954 videos from 1,873 games that were collected from the GamePhysics section on the Reddit website. Our approach shows promising results in our extensive analysis of simple and compound queries, indicating that our method is useful for detecting objects and events in gameplay videos. Moreover, we assess the effectiveness of our method by analyzing a carefully annotated dataset of 220 gameplay videos. The results of our study demonstrate the potential of our approach for applications such as the creation of a video search tool tailored to identifying video game bugs, which could greatly benefit Quality Assurance (QA) teams in finding and reproducing bugs. The code and data used in this paper can be found at https://zenodo.org/records/10211390

Index Terms—Software testing and debugging – video mining – bug reports – video games – video retrieval

1 Introduction

Video game development is a highly complex process. There are many unique challenges when applying general software engineering practices to video game development [33], [41], [45], [48], [54], including challenges in game testing. Manual testing is a widely accepted approach to game testing [44], [47], [59], however, this manual process is slow and error-prone, and most importantly, expensive. On the other hand, it is challenging to automate game testing [32], [44], [52] due to the unpredictable outputs of video games. Despite progress towards automated game testing methods [12], [36], [58], [59] and tools [5], [24], [46], [68], new approaches to game testing must be researched.

The difficulty of game testing due to the unique nature of games calls for unique testing methodologies as well. For example, we could leverage the visual aspect of games in the testing process. Having a gameplay video is very helpful when trying to reproduce a bug in the development environment for further analysis, as bug reports often contain incomplete information [7]. The ability to search a large repository of gameplay videos with a natural language query would be useful to help reproduce such bug reports. For example, in the game development domain, a bug report might state "a horse is flying in the air" (Figure 1) without a screenshot or video to show what is actually happening. A gameplay video search would allow game developers to find example instances of a specific bug in the pile of gameplay videos from their playtesting sessions or the internet (e.g., YouTube, Twitch).

Fig. 1: Video identified by our approach with the bug query *'A horse in the air'* for Red Dead Redemption 2.

Despite containing rich information, the challenges related to video parsing and understanding mean that gameplay videos are difficult to utilize. Manually identifying bug instances is time-consuming, and there is limited prior research on automatic methods for mining large repositories of gameplay videos [34], [38].

In this paper, we address the challenges of extracting useful information from large repositories of gameplay videos. We propose an approach for mining gameplay videos using natural language queries by leveraging the Contrastive Language-Image Pre-Training (CLIP) model [50] to identify similar text-image pairs without any additional training (i.e., zero-shot prediction). We leverage CLIP for videos by pre-processing the video frame embeddings and use Faiss [23] to perform a fast similarity search for the pairs of text queries and video frames. In

• *Mohammad Reza, Finlay, and Cor-Paul are with the Analytics of Software, GAmes and Repository Data (ASGAARD) lab, University of Alberta, Edmonton, Canada*
Sarra is with Ubisoft Montreal, Montreal, Canada E-mail: see https://github.com/asgaardlab/CLIPxGamePhysics

Manuscript received April 19, 2005; revised August 26, 2015.

our approach, we present two methods to aggregate the similarity scores of each text-image pair to identify relevant videos. Figure 1 shows an example of a video that was identified by our approach when searching videos from the Red Dead Redemption 2 game using the bug query *'A horse in the air'*. The primary application of our approach is as a gameplay video search engine to aid in reproducing game bugs.

To evaluate our approach, we collected and prepared the `GamePhysics` dataset, consisting of 26,954 gameplay videos that contain outstanding game physics examples (including many bugs). We first did a preliminary evaluation of our approach with sets of simple queries, and compound queries, to show that our approach can identify objects and (bug-related) events in large repositories of gameplay videos. Our approach and the preliminary evaluation were published in our original paper at the Mining Software Repositories conference [56]. In this paper, we extend our original work with an in-depth evaluation of our approach on a commercial game from our industry partner.

The main contributions of our paper are as follows:

- We propose an approach to search for objects and events in gameplay videos using a natural language text query. (Sec. 4)
- We present the `GamePhysics` dataset [1], consisting of 26,954 gameplay videos from 1,873 games. (Sec. 5)
- We demonstrate the promising performance of our approach in identifying game physics bugs through 2 experiments with simple and compound queries in Sec. 6 and present the results in Sec. 7.
- We extend our original paper [56] by conducting an extensive evaluation of our method on a game from our industry partner in which we compare several variations of CLIP. (Sec. 8)

The remainder of our paper is structured as follows. Section 2 motivates our study by providing relevant background information. Section 3 discusses related work. Section 4 presents our approach to mining large repositories of gameplay videos. Section 5 discusses collecting and preprocessing the `GamePhysics` dataset. Section 6 details the experimental setup of our preliminary evaluation, and Section 7 presents the results. Section 8 gives a detailed analysis of our proposed method in a controlled (industrial) environment and compares it with several alternative approaches. Section 9 provides discussion and insights on limitations and the performance of our approach. Section 10 addresses threats to validity, and Section 11 concludes the paper.

2 Motivation and Background

2.1 Video game (physics) bugs

In this paper, we are interested in a specific category of bugs in video games that we call 'game physics' bugs. Game physics bugs are not necessarily related to an inaccurate physics simulation. Many of these bugs are related to the faulty representation of game objects due to an error in the internal state of that object. A few sample instances of game physics bugs can be seen in Figure 2. In Figure 2a, a bug from Grand Theft Auto V related to object collisions is shown. Figure 2b shows a bug from The Elder Scrolls V: Skyrim, related to object clipping. In Figure 2c, a bug from Red Dead Redemption 2 related to ragdoll poses can be seen. Figure 2d shows a bug from Cyberpunk 2077, related to object collisions. Identifying game physics bugs is challenging because we need to be able to extract specific, high-level (abstract) events from the gameplay videos, that are often similar to correct behavior.

1. https://huggingface.co/datasets/taesiri/GamePhysics

2.2 Challenges in mining gameplay videos

Until now, it has been challenging to extract valuable information from large repositories of gameplay videos. Identifying bug instances by manually checking the contents of gameplay videos is time-consuming [34]. Therefore, automatic methods for mining gameplay videos are required. The only existing approach for automatically extracting events from gameplay videos requires manual data labeling (and the training of new models) [38], which itself is time-consuming. Therefore, an effective method for extracting valuable information from gameplay videos should be able to automatically analyze the video contents without requiring manual data labeling.

2.3 Contrastive learning and zero-shot transfer

While there are many approaches toward zero-shot learning, we are interested in assessing the zero-shot performance of pre-trained contrastive models. Contrastive learning is a machine learning technique in which the goal is to learn a representation of inputs such that similar items stay close to each other in the learned space, while the dissimilar items are far away [4], [9]. In recent years, contrastive learning has been one of the key drivers in the success of self-supervised learning methods and has been used for zero-shot transfer learning [11], [18], [28], [50]. Zero-shot learning tackles a family of problems in machine learning by letting an algorithm solve a task without having a training set for that specific task [30], [31]. To illustrate this idea, suppose that a person has never seen a zebra before. If we give them a detailed description of a zebra (e.g., an animal similar to a horse, but with black-and-white stripes all over their bodies), that person can identify a zebra when they see one.

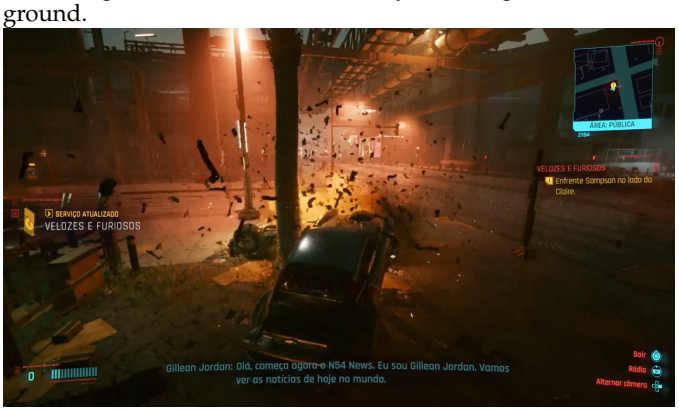
2.4 The Contrastive Language-Image Pre-Training (CLIP) model

One contrastive model that has proven zero-shot transfer capabilities is the Contrastive Language-Image Pre-Training (CLIP) model [50], which can leverage both text and image inputs together. We decided to use CLIP in our original paper because of its multimodal capabilities and the size of its training dataset. CLIP consists of two parts: a text encoder, and an image encoder. These two parts work individually, and they can accept any English text and image as input. When an encoder of this model receives an input, it will transform it into an embedding vector. These embedding vectors are high-level features that are extracted by the network, representing the input. More specifically, these embedding vectors are how the neural network represents, distinguishes, and reasons about different inputs.

(a) Bug in Grand Theft Auto V. Car stuck in a tree after colliding.

(b) Bug in The Elder Scrolls V: Skyrim. Dragon stuck in the ground.

(c) Bug in Red Dead Redemption 2. Incorrect sitting animation.



(d) Bug in Cyberpunk 2077. Cars stuck together after colliding.

Fig. 2: Sample instances of game physics bugs.

Both encoders of this model will produce vectors of the same dimension for image and text inputs. Not only do these vectors have the same dimension, but they are also in the same high-dimensional feature space and are therefore compatible with each other. For example, the embedding vector of the text 'an apple' and the embedding vector of an image of an apple are very close to each other in this learned space. CLIP was pre-trained on over 400 million pairs of images and text descriptions that were scraped from the internet and has several different backbone architectures: `RN50`, `RN101`, `RN50x16`, `ViT-B/32`, `ViT-B/16`, etc. The models with 'RN' in their name are ResNet-based [21] models using traditional convolutional layers, while the 'ViT' models are based on vision transformers [13].

3 Related Work

Event extraction from video content is of special importance for various data mining tasks [40], [49]. Only two prior studies have explicitly explored automatic approaches for mining gameplay videos, with varying success. Lin et al. showed that using metadata (such as keywords) to identify YouTube videos that contain video game bugs is feasible [34], but our approach looks at the video contents, which Lin et al. do not take into account. Our approach is more useful for game developers, as we can identify objects and (bug-related) events within gameplay videos and do not rely on metadata. Luo et al. proposed an approach for automatic event retrieval in e-sport gameplay videos that requires manual data labeling, a fixed set of classes (events), and the training of new models [38]. Our approach is more robust and easier to set up, as we can search gameplay videos with any English text query to identify specific objects and events without performing manual data labeling. Zhang et al. [67] investigated the retrieval of specific moments in narrative-driven games using natural language queries that semantically match both the auditory and visual content of scenes. In contrast, our approach focuses on video game bugs and does not rely on audio information. Furthermore, our approach works with a broader range of games.

Although there is limited prior work on mining large repositories of gameplay videos, there are several studies that propose approaches to automatically detect graphics defects in video games. One of the earliest approaches for automated detection of graphics defects was published in 2008, in which a semi-automated framework was proposed to detect shadow glitches in a video game using traditional computer vision techniques [42]. Recent studies have utilized convolutional neural networks in their approach to automatically detect a range of graphics defects [10], [12], [36], [55]. Instead of detecting graphics defects, our work is concerned with the automatic identification of game physics bugs in gameplay videos.

Tuovenen et al. leveraged the visual aspect of games through an image-matching approach to create a record-and-replay tool for mobile game testing [58]. Our approach

leverages the visual aspect of games in a different way; instead of recording tests through gameplay, we automatically identify bugs in gameplay videos.

Some studies have proposed approaches for the automated detection of video game bugs through static or dynamic analysis of source code. Varvaressos et al. proposed an approach for runtime monitoring of video games, in which they instrument the source code of games to extract game events and detect injected bugs [59]. Borrelli et al. proposed an approach to detect several types of video game-specific bad smells, which they formalize into a tool for code linting [8]. Our approach differs as we do not require access to the source code of games; instead, we identify video game bugs based solely on the contents of gameplay videos.

In addition to related work on automatic bug detection for video games, there exists a wide range of work that leverages recent advancements in deep learning to provide new tools and techniques that address problems faced by game developers. Several studies have sought to make AI methods accessible in the video game development and testing cycle, either through the game's internal state, raw pixels or through a high-level neural network-based representation [27], [39], [57]. Some studies have proposed approaches to accompany a game designer through the creation process of a game by providing suggestions and explanations to the designer [19], [20], [26]. Other studies have incorporated reinforcement learning and evolutionary methods to create AI agents that can automatically play games [6], [25], [63]. These AI agents can be further employed to perform automated game testing sessions [15], [17], [51], [68]. Our work is different from those listed above, as we focus on assisting game developers by providing an approach to efficiently search large repositories of gameplay videos to find bug instances.

4 Our Approach

To assist with the detection and analysis of game bugs, we propose an approach that quickly and effectively searches a large repository of gameplay videos to find a specific object or event in a particular game. For creating such a powerful search system, one could utilize a traditional supervised classification technique. However, any supervised classification method needs a training dataset, a test dataset, and a fixed number of classes. Maintaining these datasets and labeling each sample is demanding and labor-intensive. Conversely, CLIP provides zero-shot transfer learning capabilities that allow us to develop an approach to automatically mine gameplay videos while avoiding the aforementioned issues. Figure 3 shows an overview of our approach.

4.1 Encoding video frames and the text query

Our approach accepts a set of videos and any English text query as inputs. We first extract all frames from each video and then use CLIP to transform our input text query and input video frames into the embedding vector representations described in Section 2.4. We selected CLIP because it is flexible enough to accept any arbitrary English text as a query and compare it with a video frame, without any additional training.

4.2 Calculating the similarity of embeddings

As well as avoiding manual data labeling, our approach avoids depending upon any extra information, such as metadata, to search gameplay videos. Instead, we are able to calculate similarity scores solely based on the contents of the video frames and the text query. The similarity score in our problem is the distance between an embedding vector representing a text query and another embedding vector representing a video frame. To calculate similarity scores for the pairs of embedding vectors, we opted for cosine similarity, a widely-used similarity metric [14], [61], [62], [66]. We require an exhaustive search to calculate the similarity score of the text query with each individual frame in each input video. The performance of an exhaustive search will decrease inversely with an increasing number of videos in a repository. To combat this, we use Faiss [23] to conduct an efficient similarity search.

4.3 Aggregating frame scores per video

Although CLIP is designed to accept text and images as inputs, we can leverage CLIP for videos by treating each video as a collection of video frames (i.e. a collection of images). To identify specific events that could occur at any moment in a gameplay video, we cannot subsample the video frames as suggested in the original CLIP, because due to the richness of events in a single gameplay video, skipping any part of the video may lead to information loss and inaccurate results. Therefore, we perform a similarity search on all frames of all videos by comparing each individual video frame with the target query text, and we subsequently aggregate the similarity scores across each video. Below we detail the design of two different methods for aggregating the video frame similarity scores for each gameplay video. Our approach supports the two aggregation methods without the need to re-calculate the similarity scores.

Aggregating frame scores using the maximum score

Our first score aggregation method ranks videos based on the maximum similarity score across all frames belonging to each video. This method is highly sensitive, as a single frame with high similarity can lead to an entire video being identified as relevant to the query.

Aggregating frame scores using the similar frame count

In the second score aggregation method, we begin by ranking all frames of the input videos based on their similarity scores with the text query. Then, we select a predefined number (the *pool size* hyperparameter) of the highest-ranked frames across all videos. Finally, we count the number of frames per video within this pool of highest-ranked frames. This method is less sensitive than our first aggregation method, as identified videos must have multiple frames that are among the most similar to the input text query. We selected 1,000 as the default pool size value in our study.

5 Preparing the GamePhysics dataset

5.1 Collecting the `GamePhysics` dataset

Developing and testing a new machine learning system requires a dataset. Unfortunately, there is no such dataset for

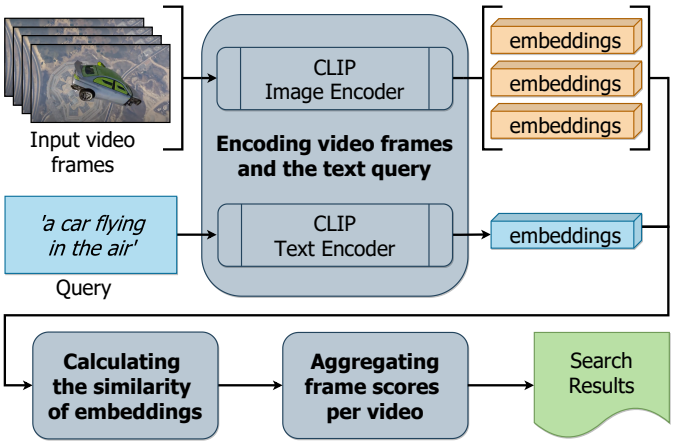


Fig. 3: Overview of our gameplay video search approach.

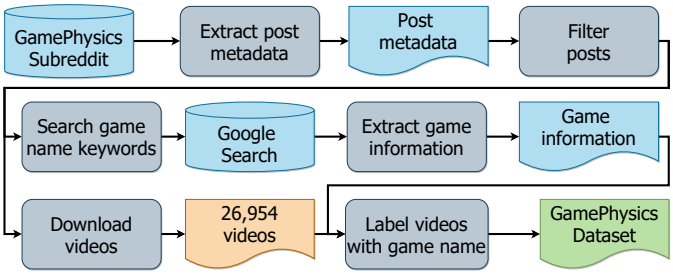


Fig. 4: Overview of our data collection process.

gameplay bugs. To this end, we present the `GamePhysics` dataset, which consists of **26,954** gameplay videos collected from the GamePhysics subreddit. An overview of our data collection process can be seen in Figure 4.

Extracting post metadata and downloading videos

To collect the data, we created a custom crawler that uses both the official Reddit and the `PushShift.io` [3] APIs. We use the `PushShift.io` API to get high-level information about each submission in the GamePhysics subreddit. After obtaining high-level data, we use Reddit's official API to update the scores and other metadata of each submission. For downloading video files, we combine `youtube-dl` and `aria2c` to extract links and download them.

Filtering posts

We applied several filters to our dataset during the data-collecting process to remove spam posts, low-quality content, and outliers. There are several spam posts in the GamePhysics subreddit, and these posts are marked explicitly as spam by the subreddit's moderators. Furthermore, we treat post scores as a quality signal as this score captures up/down votes from Reddit users, and consider any post with a score of less than one as low-quality content. The lengths of the video files vary from a few seconds to multiple hours. We avoid long videos in our dataset because they can contain multiple events of different kinds and are very hard to process. We only keep videos that are longer than 2 seconds and shorter than 60 seconds. After applying our filters, our final dataset contains **26,954** video files from **1,873** different games.

Labelling videos with the game name

In order to simulate the realistic scenario in which a game developer would search a repository of gameplay videos for a specific game, we extract the game name for each gameplay video from the title of its respective post. Detecting the game's name from a GamePhysics submission is not straightforward. While there is a community guideline that suggests including the name of the game in the submission's title, people often forget to include the game name or use several aliases for the game name, meaning the task of detecting the game name can be hard. For example, 'GTA V' is a widely-used alias that refers to the 'Grand Theft Auto V' game. To address this issue, we created a second custom crawler to search game name keywords in Google and subsequently map them to the full game name. Google search results provide a specific section called the Knowledge Panel that contains the game name, as well as other relevant game information such as initial release date, genre, development studio(s), and publisher.

5.2 Pre-processing the videos

As discussed in Section 4.2, our approach can search a large repository of gameplay videos more efficiently by pre-processing the embedding vectors of every frame for each video in the repository before inputting any text queries. Therefore, for our dataset to be suitable for our approach, we pre-process all videos in the `GamePhysics` dataset before proceeding with any experiments. We pre-processed all 26,954 videos using a machine with two NVIDIA Titan RTX graphics cards, but it is certainly possible to perform this step with less powerful graphics cards too. It is worth noting that this is by far the most computationally expensive step in our approach.

6 Preliminary Evaluation Setup

In this section, we describe the preliminary evaluation of our approach on the `GamePhysics` dataset through a diverse set of experiments. To evaluate our video search method, we conducted multiple experiments with varying difficulty levels. The main obstacle to evaluating our search system is the lack of a benchmark dataset. To this end, we designed two experiments with corresponding sets of queries to shed light on the capabilities of our proposed method.

6.1 Experiment overview

In two experiments, we evaluate the accuracy of our approach when retrieving videos with particular objects in them. The results for this step indicate the generalization capability of the model.

6.2 Selecting CLIP architectures

To understand the relative performance of the available ResNet-based and vision transformer-based CLIP models, we tried the `RN101` and `ViT-B/32` backbone architectures. We chose these backbones as fair baseline comparisons because they are the largest backbone architectures in their respective families, assuming we stipulate equivalent input image sizes (224×224). For comparison, the `ViT-B/32`

backbone architecture has 151 million total parameters, while the `RN101` backbone architecture contains 119 million parameters. We selected the largest architectures as we only perform inference and no training.

6.3 Selecting video games

Our dataset contains videos from 1,873 different video games, and the differences in their high-level characteristics, such as genre, visual style, game mechanics, and camera view, can be vast. Therefore, we performed a comprehensive evaluation in both experiments with 8 popular video games that differ in their high-level characteristics. The only uniting characteristic of our selected games is that they have open-world mechanics because developers of open-world games would find particular benefits from an effective video search for bug reproduction. Open-world games allow a player to freely explore the game world, providing a very large set of potential interactions between the player and the game environment. Open-world games are therefore more likely to suffer from game physics bugs that are difficult to reproduce. Table 1 shows the selected games, as well as some game characteristics and the reason for inclusion. In total, 23% of videos in the `GamePhysics` dataset are from these 8 video games (6,192 videos).

6.4 Query formulation

To come up with a set of relevant search queries in the experiments, we randomly picked 10 videos from each of the 8 selected games. The first author manually reviewed each of the 80 samples to understand what was happening and how we could describe events in gameplay videos that contain bugs. This sampling process helped us pick relevant objects and events to use in our queries.

6.5 Preliminary Experiment 1: Simple Queries

In this experiment, we searched for specific objects in videos, e.g. a car. Our main objective in this experiment is to demonstrate the capability of our system for effective zero-shot object identification. As a reminder, we never trained or fine-tuned our neural network model for any of these experiments or any video game. We created 22 distinct queries for Experiment 1, including transportation vehicles, animals, and special words describing the weather or environment. For this experiment, we wanted our approach to operate with very high detectability and to detect smaller variations in the video, and so we selected our first aggregation method, i.e. using maximum frame score per video (Section 4.3).

6.6 Preliminary Experiment 2: Compound Queries

Continuing our evaluation, we search for compound queries, i.e. queries in which an object is paired with some descriptor. Similar to Experiment 1, we only use compound queries that are relevant to each video game. For example, in the previous experiment, we searched for videos in the Grand Theft Auto V game that contained a car, but in this experiment, we evaluate the performance of our approach when searching for objects with a specific condition, like a car with a particular color. For this second experiment, we created a set of 22 compound queries and again selected our first aggregation method (using maximum frame score per video).

6.7 Evaluating the experiments

Evaluating preliminary experiments.

In the first and second experiments, we assess the detectability of our approach by measuring Top-1 and Top-5 accuracy. This is because for our approach to be useful to a game developer, the search system should be able to reliably identify objects specified in the text queries. Top-k accuracy is a binary measure; if there is a correct result in the Top-k results, the accuracy is 100%, otherwise, the accuracy is 0% – there are no possible values in between.

7 Results for Preliminary Experiments

In this section, we present experimental results to examine our proposed search system's ability.

Results for simple queries (Preliminary Experiment 1)

In the first experiment, we measured the Top-1 and Top-5 accuracy of our system with simple queries. The average accuracy for experiment 1 per game can be seen in Table 2, and per query in Table 4. The overall average Top-1 accuracy and average Top-5 accuracy of `ViT-B/32` is 60% and 76% respectively, and for `RN101` we have 64% and 86% respectively. These results show that our system can identify a majority of objects without fine-tuning or re-training.

Results for compound queries (Preliminary Experiment 2)

In the second experiment, we measure the Top-1 and Top-5 accuracy of our approach with compound queries. The average accuracy in Experiment 2 per game can be seen in Table 3. For the second experiment, we find that our approach shows particularly high performance for all of our selected games, except for The Witcher 3: Wild Hunt. Our approach achieves an overall average Top-5 accuracy of **78%** using `ViT-B/32` and **82%** using the `RN101` model. These results show that our approach is flexible enough to effectively search gameplay videos with compound queries.

8 In-depth Evaluation

Our preliminary findings demonstrate the effectiveness of two CLIP architectures in retrieving relevant gameplay video clips using simple queries. We expand our assessment by focusing on finding relevant videos using descriptions of bugs written in natural language. Each bug can be described using simple phrases that explain the issue. We use such descriptions as queries to search videos to find instances of that particular bug. The in-depth evaluation reported in this section was done in collaboration with Ubisoft La Forge.

8.1 Experiment setup

In this experiment, we search videos with bug queries, i.e., phrases that describe an event in the game that is related to a bug. We create a new dataset in which each video is associated with a list of text descriptions detailing the

TABLE 1: Selected games for the preliminary evaluation of our approach. All selected games are open-world.

Game	Key	Genre	Visual style	Reason for inclusion	Videos
Grand Theft Auto V	GTA	Action-adventure	Realism	Variety of vehicles	2,230
Red Dead Redemption 2	RDR	Action-adventure	Realism	Historical style	754
Just Cause 3	JC3	Action-adventure	Realism	Physical interactions	680
Fallout 4	F4	Action role-playing-game	Fantasy realism (Retro-futuristic)	Unique look and feel	614
Far Cry 5	FC5	First-person shooter	Realism	First-person camera	527
Cyberpunk 2077	C77	Action-adventure	Fantasy realism (Futuristic)	High-quality lighting	511
The Elder Scrolls V: Skyrim	ESV	Action role-playing-game	Fantasy realism	Magical effects	489
The Witcher 3: Wild Hunt	W3	Action role-playing-game	Fantasy realism	Mythical beasts	387

TABLE 2: Average Top-k accuracy (%) per game for simple queries (Preliminary Experiment 1).

		GTA	RDR	JC3	F4	FC5	C77	ESV	W3
ViT-B/32	**Top-**1	74	71	61	65	50	55	54	54
	Top-5	89	86	67	71	88	73	62	62
RN101	**Top-**1	84	50	61	59	59	43	62	62
	Top-5	89	79	83	82	94	71	92	85

TABLE 3: Average Top-k accuracy (%) per game for compound queries (Preliminary Experiment 2).

		GTA	RDR	JC3	F4	FC5	C77	ESV	W3
ViT-B/32	**Top-**1	68	88	56	43	31	50	56	56
	Top-5	100	100	81	64	69	75	89	67
RN101	**Top-**1	84	88	31	36	56	67	33	44
	Top-5	95	100	75	79	94	83	78	56

TABLE 4: Average Top-k accuracy (%) per query for simple queries (Preliminary Experiment 1). N is the number of games searched.

		ViT-B/32		RN101	
Query	N	**Top-**1	**Top-**5	**Top-**1	**Top-**5
Airplane	4	75	100	100	100
Bear	5	80	100	60	100
Bike	4	50	75	50	100
Bridge	8	88	88	50	100
Car	5	80	100	80	100
Carriage	4	50	50	75	100
Cat	6	33	50	33	67
Cow	8	63	75	25	75
Deer	7	57	71	75	100
Dog	8	25	38	38	63
Fire	8	88	100	100	100
Helicopter	5	60	60	60	100
Horse	3	67	100	100	100
Mountain	7	100	100	100	100
Parachute	2	0	67	67	100
Ship	8	50	63	38	75
Snow	6	67	83	33	50
Tank	3	67	67	100	100
Traffic Light	5	40	40	20	20
Train	5	80	100	17	67
Truck	4	75	100	100	100
Wolf	6	17	50	86	86
Average	**5.5**	**60**	**76**	**64**	**86**

bugs occurring within the video. It is important to note that these text descriptions are not utilized for retrieving videos, but rather serve as a means to evaluate the retrieval performance. For this particular experiment, we choose a game from the Assassin's Creed[2] franchise.

We rank videos based on their similarity to a given bug query and return a list of videos sorted by rank. We report the Top-k accuracy for various values of k ranging from 1 up to 50. The accuracy @25 and @50 is particularly relevant, as reviewing each result takes roughly one second. By presenting up to 50 results on a single page, users can efficiently review all the results in under a minute[3]. In order to enhance the efficiency of the review process, we provide a segment of the video that closely corresponds to the query and generate a GIF animation, which allows for a quick review of the results. This approach is especially advantageous for important bugs, as it allows for greater effort to be invested in identifying a relevant video.

To offer a comprehensive understanding of model performance and resource requirements, we present GPU memory usage for each model, in addition to the performance metrics. By examining the accuracy and computational costs, we provide a thorough analysis of the factors to consider when selecting a model for real-world applications.

8.2 Dataset and labeling videos

We selected all 278 videos from an Assassin's Creed game from our GamePhysics dataset. We labeled all the videos manually and annotated them with natural language descriptions. We allowed multiple labels for each video since multiple bugs can occur in one video. We used an iterative process to label videos and in each iteration, we ensured that different occurrences of the same bug have been labeled with a unique bug description. That is, if multiple videos exhibit the same bug but in different places, we assign an identical label to all videos. After filtering out low-quality videos (those with low resolution and significant degradation due to video encoding) and eliminating those without any apparent bug, our refined dataset ultimately contains 220 videos. Some sample bug descriptions can be seen in Table 5.

8.3 Models

In total, we evaluated five architectures for CLIP, and two for OpenCLIP [22], an open-source replication of CLIP. In addition, we evaluated four video-text contrastive models (XCLIP [43], BridgeFormer [16], CLIP4Clip [37] and Frozen-In-Time [2]), which work on more than one frame in contrast to CLIP, which operates at the individual frame level.

2. https://www.ubisoft.com/en-ca/game/assassins-creed
3. A sample search result is available at this URL.

TABLE 5: Sample queries

#	Description	# Videos
1	An axe is floating in the air.	1
2	A horse is walking in the air.	1
3	A whale is floating in the air.	1
4	A person is shaking very fast.	2
5	A sword is flying in the air.	2
6	A floating boat is rotating very fast.	4
7	A person swimming in the air.	4
8	A boat is floating in the air.	6
9	A person is standing in the air.	7
10	A horse is running on its two legs.	8
11	A person is falling from the sky.	12

Some gameplay bugs manifest themselves temporally, such as when an object shakes rapidly in the game. Video-text models can incorporate temporal information and, in theory, have the ability to detect such bugs.

8.3.1 CLIP

We studied five architectures for the original CLIP model, with different backbone architectures (ResNet and ViT). In particular, we include `ViT-B/16`, `ViT-B/32`, `ViT-L/14`, `RN50x64`, and `ViT-L/14@336px` in our study, which differ in architecture and parameter sizes. The input dimension of all the models is 224×224, with the exception of `ViT-L/14@336px` and `RN50x64` which require an input image with the resolution of 336×336 pixels and 448×448 pixels, respectively.

8.3.2 OpenCLIP

OpenCLIP is an open-source replication of CLIP that contains a diverse array of models with different sizes and architectures. We selected two of the largest and most capable OpenCLIP models; `ViT-g/14` and `ViT-H/14`, to compare with the original CLIP. OpenCLIP models were trained on the LAION [53] dataset, consisting of 5.85 billion CLIP-filtered image-text pairs. Both of the selected models have an input spatial dimension of 224×224 pixels.

8.3.3 XCLIP

XCLIP consists of a transformer [60] model trained on features extracted by the original CLIP model to incorporate and fuse temporal signals and information. Essentially, this model receives a sequence of 32 frames with a resolution of 224×224, as its input and then compares the similarity of this sequence to a given text.

8.3.4 CLIP4Clip

CLIP4Clip is a straightforward expansion of the original CLIP model for learning representations of video-text data. CLIP4Clip first extracts CLIP embeddings for each frame and then combines them to create an embedding for the entire video. Three types of aggregation methods can be used with CLIP4Clip: 'parameter-free', 'sequential type', and 'tight type'. The first method is a basic averaging technique that does not require any training. We opt for this aggregation method because it is not limited to a particular domain, and more importantly, it delivers similar and competitive results across various benchmarks [37]. The CLIP4Clip model employs a `ViT-B/32` model with an input resolution of 224×224 pixels. Note that CLIP4Clip is very similar to the approach used in our preliminary evaluation (the approaches were developed and published in parallel).

8.3.5 Frozen-In-Time

Frozen-In-Time is a transformer-based model that effectively learns a joint representation of video and text. The model supports single and multi-frame inputs, allowing it to leverage the strengths of both image and video datasets. This model treats image inputs as a special case of videos that are "frozen" in time. This model accepts frames and images with a 224×224 pixels resolution.

8.3.6 BridgeFormer

BridgeFormer is a video-text pre-training model based on a Multiple Choice Questions (MCQ) formulation. This model is trained to answer questions constructed by the text features, based on video features. This formulation enables the model to capture additional regional content and temporal dynamics in videos, in addition to semantic associations between local video-text features. BridgeFormer works on the input resolution of 224×224 pixels.

8.3.7 Random baseline

Our labeled dataset comprises only 220 videos, which may result in inflated accuracy rates across all models. We include a `Random Baseline` to evaluate the efficacy of our proposed methods compared to randomly reviewing videos. This model assigns random similarity scores to query and video pairs, returning a random order of videos for queries. Therefore, this random retriever provides a way to determine the actual benefit of the proposed models.

8.3.8 Implementation details

We used official libraries for the CLIP[4] and OpenCLIP[5] models. Since CLIP and OpenCLIP work at the individual frame level, we used two aggregation methods for all models of this kind, *max* and *mean*. For a given query and a video, we calculate the similarity score between all embeddings of frames and the embedding of the given query. For the *max* aggregator, we choose the maximum score among all frames with the query and use it as the final score, and for the *mean* aggregator, we use the average of the top 5 similar frame count scores (as discussed in Section 4).

We used the XCLIP implementation from the Hugging Face library [64]. Since XCLIP expects an array of 32 frames as input, we fed the model a rolling window of 32 frames and stored multiple embeddings per video. For a given query and a video, we calculated the text embedding of the video with all window embeddings and used the *max* aggregator to calculate the final similarity score.

We employed the Towhee framework[6] to implement the CLIP4Clip, Frozen-In-Time, and BridgeFormer models. For each input video, we uniformly sampled frames and fed

4. Github - OpenAI CLIP
5. Github - mlfoundations OpenCLIP
6. https://towhee.io/

them to the respective model. Specifically, we utilized the *uniform temporal subsampling*[7] technique to extract frames from each video.

8.4 Experimental Results

Table 6 shows the results of our experiment, and Figure 5 illustrates samples of successful retrieval. The considerable gap between the results of the `Random Baseline` and the other models demonstrates that the selected models are genuinely detecting relevant information and are not simply benefiting from chance.

`ViT-L/14` and `ViT-L/14@336px` perform best, with `ViT-L/14@336px` marginally better. Our results show the `ViT-L/14@336px` model consistently outperforms all other tested models. It achieves a Top-10 accuracy of 49.44% and its Top-50 accuracy reaches up to 82.58%, suggesting it can successfully identify relevant videos for most queries.

Both input resolution and architecture are important factors in bug video identification performance. Among all tested models, some have different input resolutions, e.g., `RN50x64` has an input size of 448×448 pixels, and `ViT-L/14@336px` has 336×336 pixels. Higher-resolution inputs provide more information for the model to process, paving the way for better performance at the expense of increased computational resources (discussed in Section 8.5). Comparing performance among architectures based on ViT (`ViT-L/14` vs `ViT-L/14@336px`), shows indeed that higher resolution inputs lead to better performance. However, the ViT architecture performs better than the ResNet architecture (`RN50x64` vs `ViT-L/14@336px`), despite the fact that ResNet uses a higher input resolution. This suggests that both model size and architecture play roles in the final performance of a model.

Video-text models consistently perform worse than image-text models for gameplay videos. Among all tested video-text models, XCLIP achieves better Top-1 and Top-5 accuracies. However, as we gradually increase the accuracy threshold, the advantage of XCLIP seems to diminish, and all video-text models perform similarly and consistently worse than image-text models. This low performance at first is counter-intuitive since video-text models have the ability to exploit temporal information in videos and should perform better than image-text models. However, this low performance happens because these models have been fine-tuned on real-world videos, which are very different from gameplay videos, i.e., gameplay videos are considered out-of-distribution for these models. This finding is aligned with the observations reported by other previous studies [29], [65].

Image-text models cannot exploit the temporal dimension but are able to use other information to find relevant videos. A subset of bugs in our dataset manifest themselves over time, and one can not detect them from a single frame. For instance, when an object is unintentionally *shaking*, individual frames do not reflect this behavior and we need to incorporate the temporal aspects of the video. Even though both CLIP and OpenCLIP are image-text models and lack an understanding of the temporal aspects of a video, they consistently performed better than video-text models. After carefully analyzing some queries, we found that image-text models oftentimes utilize a portion of the query to retrieve the most relevant videos that represent the query as closely as possible. For instance, for the query "*A person is being teleported into the sky*", CLIP retrieves videos showing aerial views of the game world, which turn out to be the correct videos for the given query.

8.5 GPU memory usage and search speed analysis

The `ViT-L/14@336px` model outperforms its counterparts at every accuracy threshold; however, this performance comes with the drawback of higher GPU memory usage, which might hinder its usage compared to other models. In this section, we present the GPU memory requirements for each model. We used the `gpustat`[8] Python package to determine GPU memory usage for each model. We created a sample script that loads and passes input to the model and measures the maximum memory utilization. For both CLIP and OpenCLIP families, we use a batch size of 256 images as input. This batch size is roughly equivalent to the length of all frames in a short video (e.g., a 10-second video with 24 frames per second). For all video-text models, we use an input that matches the network input dimensions. These measurements provide an approximate estimate of the GPU memory required to calculate the similarity of a gameplay video with a text query. Table 7 shows the GPU memory usage measurements. As we can see, the OpenCLIP `ViT-g/14` model requires the most memory, but according to our results in Table 6, its performance falls behind the CLIP `ViT-L/14@336px` model. Also, the ResNet model requires considerably more GPU memory than the better-performing ViT models. Hence, model performance is not necessarily correlated with GPU memory requirements.

An additional factor that impacts the practical application of our method is the search speed, which refers to how fast a video can be retrieved. Our method requires a one-time pre-processing of videos into embeddings, which can conveniently be performed overnight or as part of a company's continuous build process. This enables subsequent search queries to be efficiently addressed using the generated embedding files. By employing the Faiss library [23], the search process can be executed in a matter of milliseconds, even when handling a dataset comprising millions of frames. This efficiency makes the search both practical and realistic in real-world circumstances.

9 Discussion & Limitations

In this section, we discuss the strengths and weaknesses of our approach, based on the results of our experiments. Figure 5 shows several example video frames from videos identified when searching gameplay videos with text queries using our approach. These examples help to illustrate the promising potential of our approach. Given that our method does not require any training in gameplay videos, our zero-shot setup for detecting objects in videos is promising. During our experiments, the first author manually analyzed each video returned by our search approach, including

7. PyTorch Video Documentation

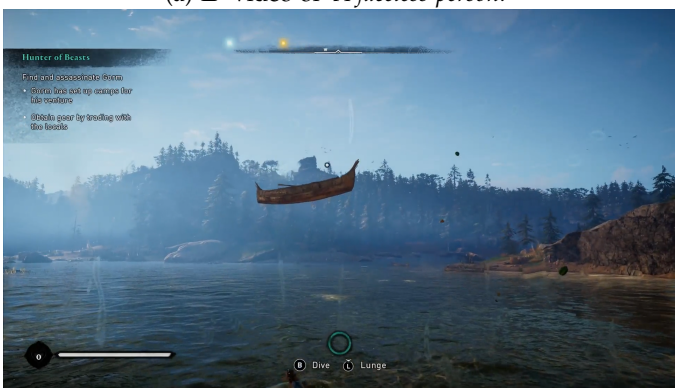
8. https://github.com/wookayin/gpustat

TABLE 6: The performance of the studied models on our labeled dataset of 220 videos – The bold numbers highlight the best-performing model. (%)

	Model Name	Modality	Aggregator	Top-1	Top-5	Top-10	Top-15	Top-20	Top-25	Top-50
	Random Baseline	video-text	-	3.93	6.74	8.99	12.92	16.29	20.22	32.02
CLIP	ViT-L/14	image-text	Max	17.42	33.15	42.70	52.25	55.62	62.36	79.21
CLIP	ViT-L/14@336px	image-text	Max	21.91	38.20	47.19	52.81	56.74	62.92	80.34
CLIP	ViT-B/16	image-text	Max	15.73	31.46	41.57	48.31	51.12	53.37	75.28
CLIP	ViT-B/32	image-text	Max	15.73	25.84	34.83	42.70	48.31	53.93	68.54
CLIP	RN50x64	image-text	Max	16.29	35.39	41.57	48.31	51.69	56.18	74.16
CLIP	ViT-L/14	image-text	Mean	19.66	34.27	44.94	55.06	58.99	62.92	82.02
CLIP	ViT-L/14@336px	image-text	Mean	**22.47**	**40.45**	**49.44**	**56.74**	**62.36**	**66.85**	**82.58**
CLIP	ViT-B/16	image-text	Mean	20.22	32.58	41.01	46.07	53.37	57.30	79.21
CLIP	ViT-B/32	image-text	Mean	15.73	32.02	41.57	46.07	53.37	55.62	73.03
CLIP	RN50x64	image-text	Mean	16.85	35.39	42.70	50.56	55.06	61.24	78.65
OpenCLIP	ViT-H/14	image-text	Max	16.85	29.21	36.52	47.75	51.69	55.62	70.79
OpenCLIP	ViT-g/14	image-text	Max	17.42	27.53	35.96	43.82	50.00	52.81	69.66
OpenCLIP	ViT-H/14	image-text	Mean	19.10	32.02	42.13	48.88	52.25	55.06	73.60
OpenCLIP	ViT-g/14	image-text	Mean	16.85	31.46	38.76	47.75	53.37	59.55	73.60
	XCLIP [43]	video-text	Max	10.11	21.91	29.78	37.64	43.26	51.69	69.66
	CLIP4Clip [37]	video-text	-	7.87	21.35	33.71	41.57	47.19	52.25	66.85
	BridgeFormer [16]	video-text	-	7.87	18.54	29.78	40.45	46.63	52.81	70.22
	Frozen-In-Time [2]	video-text	-	6.18	17.98	28.09	33.71	39.89	47.75	66.29

(a) Video of *'A faceless person.'*

(b) Video of *'A horse is running on its two legs.'*



(c) Video of *'A boat is floating in the air.'*

(d) Video of *'A person is stuck in a tree.'*

Fig. 5: Relevant gameplay videos identified using our approach with bug queries in a game from the Assassin's Creed franchise. Our method can effectively retrieve highly specific bugs using textual descriptions, without depending on specialized vocabulary or technical terms.

false positives. Below, we highlight our observations about retrieving buggy gameplay videos using our approach:

Handling technical and non-technical terms

Our observations suggest CLIP understands these technical terms, and sometimes, describing a bug in simple words

TABLE 7: GPU memory usage of the studied models.

Model	VRAM (MB)	Input Size	Embedding Size
ViT-L/14	5,861	224 × 224px	768
ViT-L/14@336px	11,803	336 × 336px	768
ViT-B/16	4,361	224 × 224px	512
ViT-B/32	2,551	224 × 224px	512
RN50x64	21,829	448 × 448px	768
ViT-H/14	15,559	224 × 224px	1024
ViT-g/14	13,275	224 × 224px	1024
XCLIP	7,979	224 × 224px	512
CLIP4Clip	2,987	224 × 224px	512
BridgeFormer	3,463	224 × 224px	512
Frozen-In-Time	3,405	224 × 224px	512

retrieves relevant videos. For instance, Figure 5(a) illustrates a game character without facial features. CLIP can retrieve this video using both a specialized term like "*a low-poly face*," and a common and non-technical description such as "*a faceless person*." It is worth noting that non-technical users and gamers could describe bugs using non-technical terms in their bug reports. For example, game reviews tend to contain implicit bug reports which are often written by non-technical users [35]. Our approach would help developers to gather more information about such bug reports.

Adversarial poses

An important category of problems is the unusual pose of familiar objects. As extensively tested and reported by Alcorn et al. [1], neural networks occasionally misclassify objects when they have different poses than what they used to have in the training set. For example, consider a neural network that can detect a "*car*" in an image. It is possible to find a particular camera angle for which the neural network can not detect the "*car*" in that image. In a dataset of natural images, there may be many examples of cars, but the camera angle and the position of the car relative to the camera usually does not vary much. A neural network trained on such data will struggle to detect a car when it sees it from a very unusual angle (e.g. when it is positioned vertically).

Embedding size

In our experiment, we used a range of models with different architectures and modalities. A key variable in these models is the embedding size. The embedding size is a fixed-length vector that summarizes the contents of an image or video. For the models we tested, the embedding size varied from 512 to 1024. A natural question arises: *Are larger embedding sizes necessarily better?* Our findings indicate that there is no direct correlation between embedding size and performance. This is evident by comparing the models like ViT-L/14@336px and ViT-g/14 which have dimension sizes of 768 and 1024 respectively, but the performance of ViT-L/14@336px is consistently better than ViT-g/14. One explanation is that the final embedding size alone does not guarantee the quality of the embedding, and this factor is largely dependent on the architecture and the computational power spent on the training rather than other factors.

Rapid camera motion causes confusion in video-text models

Among video-text models, especially XCLIP, we noticed an unexpected behavior of the model when the video contains rapid camera motion. After reviewing a subset of queries that result in retrieving unrelated videos, we observed that the top matching video often contained rapid camera movements, such as orbital movement (which is sometimes known as the "death cam"[9] in gaming communities). These movements, which are present in many gameplay videos, can be considered 'adversarial samples,' create distractions and leading to incorrect query-video matching.

Video quality

Our dataset is constructed from internet videos, which are often in low-resolution formats. However, due to lossy and heavy video compression, a significant predictive signal is lost, which can pose a challenge for any vision model. Furthermore, we have observed that sometimes, the extracted frames have very low quality. While human vision has no problem detecting objects in the video, sometimes CLIP models cannot properly detect them. It is worth mentioning that, in a controlled testing environment within a gaming company, high-quality videos can be recorded without incurring significant costs, thus avoiding any problems that may arise due to video quality.

CLIP can recognize texts

Another observation is about text patches inside gameplay videos. CLIP can *read* the text inside an image as well. This feature is not something that the model was explicitly trained for, but rather some emergent behavior of pre-training in a contrastive setting. Sometimes searching a particular text query will result in retrieving a video that ignores the meaning of the text query, but the image contains that text. For example, if any video frames include a text field containing "*a blue car*", searching for the query "*a blue car*", will retrieve that video. Obviously, depending on the use case, this can be treated as both a feature and a bug.

Confusing textures and patterns in the images

The textures and patterns can pose influential distractions and confusion for the neural network model in our approach. Sometimes a part of a game environment has a texture similar to another object. For example, our model confuses a striped colored wall in the Grand Theft Auto V game with a "*parachute*." This category of problems is hard to encounter globally because each game has a diverse look and feel and creative artistic directions.

Confusions in accurately recognizing objects with CLIP

The CLIP model struggles to accurately distinguish between various objects, with vehicles and animals serving as notable examples. In our analysis, we discovered instances where the search results partially matched the textual description but mistakenly featured a different object from the intended category. Our manual evaluation showed that CLIP often make errors in identifying various vehicles, such as cars,

9. You can view a sample by watching this video.

airplanes and tanks. Likewise, the model struggles to differentiate between four-legged animals, including dogs, cats, wolves, cows, horses, etc.

10 Threats to validity

In this section, we discuss potential internal and external factors that may affect our findings.

10.1 Threats to internal validity

Size of dataset in the industry-related evaluation

The validation set used in the industry experiment comprises 220 videos. Although this dataset encompasses all videos from our selected game, it remains relatively small compared to the entire `GamePhysics` dataset. Labeling a dataset is a laborious, time-consuming, and costly task. In this study, we selected a single, representative game illustrating the challenges of an underdeveloped industry game. When adapting our method, it is important to consider that depending on video game visuals and the size of the dataset, the number of false positives can increase or decrease.

Effect of subsampling on the performance

Our straightforward extension of image-text models for video data entails initially encoding all frames using the model, followed by an aggregation step, to select one or several frames that exhibit the highest degree of alignment with the given query. Conversely, video-text models are designed to inherently accept a collection of frames as input and subsequently generate an embedding. Yet, these models do not work on all frames, and in the initial step, they subsample the input video. We employed *uniform sampling* to select specific frames for these models; however, this approach may result in the loss of frames, potentially leading to diminished performance.

10.2 Threats to external validity

Dataset generalizability and video length

As our dataset predominantly consists of gameplay videos that contain game physics bugs, our approach may not be as effective for other datasets of gameplay videos that contain other types of bugs. Non-curated datasets may contain many more videos of non-buggy gameplay, for example, if using gameplay streaming footage. Additionally, we excluded long (>60 seconds) videos, meaning our approach may not be effective for long videos. We also ignored videos with scores of zero from the GamePhysics subreddit. After manually checking a random sample of low-scored posts we observed that a score of 0 almost always indicated low quality/spam/etc. This threshold might not be applicable for other subreddits. Future research is required to evaluate our approach with long videos and non-curated datasets.

11 Conclusion

In this paper, we proposed a novel approach to mine large repositories of gameplay videos by leveraging the zero-shot transfer capabilities of CLIP to connect video frames with an English text query. Our method can find objects in a large video dataset using simple and compound queries. Additionally, our approach demonstrates promising performance in detecting specific (bug-related) events, suggesting potential applicability in automatic bug identification for video games. Without fine-tuning or re-training for the video game domain, our method shows surprising effectiveness in most studied games.

In a preliminary study, we evaluated our method on a dataset of **6,192** videos from 8 games with different visual styles and elements to assess its capability in identifying objects in a gameplay video date. Our experiment confirms that CLIP can generalize to video game data and can effectively identify objects with simple and compound queries.

In a controlled setting, we evaluated our method on a video game from the Assassin's Creed franchise to understand the benefits and limitations of our method in a real game development environment. Our findings showcase the usefulness and potential of our technique in retrieving gameplay videos using natural language descriptions of bugs. The Top-10 accuracy of our best-performing model reaches 49.44Ġiven out-of-distribution nature of video game footage and video game bugs, this result is quite promising. Furthermore, our research demonstrates that, by employing our CLIP-based search method, it is possible to retrieve relevant videos for up to **82.58%** of queries. Despite potentially needing to review up to 8 minutes of video footage per query, this approach highlights the practicality of our method in aiding developers and quality assurance teams to rapidly identify and resolve issues within their games.

Additionally, we carried out a comparative study between various video-text and image-text models to assess their performance. Initially, it was anticipated that video-text models would be at an advantage, given their ability to integrate temporal information. However, our findings demonstrate that all tested video-text models underperformed in comparison to their image-text counterparts. This discrepancy can be attributed to factors such as out-of-distribution visuals of the game as well as camera motions that are very different from those in regular video datasets.

In conclusion, our novel method for retrieving gameplay videos demonstrates a significant advancement in the field of video game quality assurance. Our method provides a rapid and efficient approach to search through large quantities of video content, enabling quality assurance teams to rapidly identify relevant data related to game bugs. Finally, our approach lays the foundation for utilizing contrastive learning models for zero-shot bug identification in video games. Future work in this line of research will provide more insights into video game bugs and will pave the way to creating a new paradigm of automated bug detection methods for video games.

References

[1] M. A. Alcorn, Q. Li, Z. Gong, C. Wang, L. Mai, W.-S. Ku, and A. Nguyen, "Strike (with) a pose: Neural networks are easily

fooled by strange poses of familiar objects," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4845–4854.

[2] M. Bain, A. Nagrani, G. Varol, and A. Zisserman, "Frozen in time: A joint video and image encoder for end-to-end retrieval," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1728–1738.

[3] J. Baumgartner, S. Zannettou, B. Keegan, M. Squire, and J. Blackburn, "The pushshift reddit dataset," in *Proceedings of the international AAAI conference on web and social media*, vol. 14, 2020, pp. 830–839.

[4] S. Becker and G. E. Hinton, "Self-organizing neural network that discovers surfaces in random-dot stereograms," *Nature*, vol. 355, no. 6356, pp. 161–163, 1992.

[5] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén, "Augmenting automated game testing with deep reinforcement learning," in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 600–603.

[6] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[7] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, pp. 308–318.

[8] A. Borrelli, V. Nardone, G. A. Di Lucca, G. Canfora, and M. Di Penta, "Detecting video game-specific bad smells in unity projects," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 198–208.

[9] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 04, pp. 669–688, 1993.

[10] K. Chen, Y. Li, Y. Chen, C. Fan, Z. Hu, and W. Yang, "Glib: towards automated test oracle for graphically-rich applications," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1093–1104.

[11] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[12] P. Davarmanesh, K. Jiang, T. Ou, A. Vysogorets, S. Ivashkevich, M. Kiehn, S. H. Joshi, and N. Malaya, "Automating artifact detection in video games," *arXiv preprint arXiv:2011.15103*, 2020.

[13] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[14] M. Fazli, A. Owfi, and M. R. Taesiri, "Under the skin of foundation nft auctions," *arXiv preprint arXiv:2109.12321*, 2021.

[15] P. García-Sánchez, A. Tonda, A. M. Mora, G. Squillero, and J. J. Merelo, "Automated playtesting in collectible card games using evolutionary algorithms: A case study in hearthstone," *Knowledge-Based Systems*, vol. 153, pp. 133–146, 2018.

[16] Y. Ge, Y. Ge, X. Liu, D. Li, Y. Shan, X. Qie, and P. Luo, "Bridging video-text retrieval with multiple choice questions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 16 167–16 176.

[17] C. Gordillo, J. Bergdahl, K. Tollmar, and L. Gisslén, "Improving playtesting coverage via curiosity driven reinforcement learning agents," in *2021 IEEE Conference on Games (CoG)*, 2021, pp. 1–8.

[18] J. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. Á. Pires, Z. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko, "Bootstrap your own latent - A new approach to self-supervised learning," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[19] M. Guzdial, N. Liao, and M. Riedl, "Co-creative level design via machine learning," in *Joint Proceedings of the AIIDE 2018 Workshops co-located with 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2018), Edmonton, Canada, November 13-14, 2018*, ser. CEUR Workshop Proceedings, vol. 2282. CEUR-WS.org, 2018.

[20] M. Guzdial and M. Riedl, "Game level generation from gameplay videos," in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[22] G. Ilharco, M. Wortsman, R. Wightman, C. Gordon, N. Carlini, R. Taori, A. Dave, V. Shankar, H. Namkoong, J. Miller, H. Hajishirzi, A. Farhadi, and L. Schmidt, "Openclip," Jul. 2021, if you use this software, please cite it as below.

[23] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2021.

[24] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar *et al.*, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018.

[25] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep learning for video game playing," *IEEE Transactions on Games*, vol. 12, no. 1, pp. 1–20, 2019.

[26] F. Khadivpour and M. Guzdial, "Explainability via responsibility," in *Proceedings of the AIIDE Workshop on Experimental AI in Games*, 2020.

[27] N. Y. Khameneh and M. Guzdial, "Entity embedding as game representation," in *Proceedings of the AIIDE Workshop on Experimental AI in Games*, 2020.

[28] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[29] A. Kumar, A. Raghunathan, R. M. Jones, T. Ma, and P. Liang, "Fine-tuning can distort pretrained features and underperform out-of-distribution," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[30] C. H. Lampert, H. Nickisch, and S. Harmeling, "Learning to detect unseen object classes by between-class attribute transfer," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 951–958.

[31] H. Larochelle, D. Erhan, and Y. Bengio, "Zero-data learning of new tasks." in *AAAI*, vol. 1, no. 2, 2008, p. 3.

[32] C. Lewis and J. Whitehead, "The whats and the whys of games and software engineering," in *Proceedings of the 1st international workshop on games and software engineering*, 2011, pp. 1–4.

[33] C. Lewis, J. Whitehead, and N. Wardrip-Fruin, "What went wrong: a taxonomy of video game bugs," in *Proceedings of the fifth international conference on the foundations of digital games*, 2010, pp. 108–115.

[34] D. Lin, C.-P. Bezemer, and A. E. Hassan, "Identifying gameplay videos that exhibit bugs in computer games," *Empirical Software Engineering*, vol. 24, no. 6, pp. 4006–4033, 2019.

[35] D. Lin, C.-P. Bezemer, Y. Zou, and A. E. Hassan, "An empirical study of game reviews on the steam platform," *Empirical Software Engineering*, vol. 24, pp. 170–207, 2019.

[36] C. Ling, K. Tollmar, and L. Gisslén, "Using deep convolutional neural networks to detect rendered glitches in video games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, 2020, pp. 66–73.

[37] H. Luo, L. Ji, M. Zhong, Y. Chen, W. Lei, N. Duan, and T. Li, "Clip4clip: An empirical study of clip for end to end video clip retrieval and captioning," *Neurocomputing*, vol. 508, pp. 293–304, 2022.

[38] Z. Luo, M. Guzdial, and M. Riedl, "Making CNNs for video parsing accessible: event extraction from dota2 gameplay video using transfer, zero-shot, and network pruning," in *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 2019, pp. 1–10.

[39] F. Macklon, M. R. Taesiri, M. Viggiato, S. Antoszko, N. Romanova, D. Paas, and C.-P. Bezemer, "Automatically detecting visual bugs in html5 canvas games," in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–11.

[40] L. MacLeod, M.-A. Storey, and A. Bergen, "Code, camera, action: How software developers document and share program knowledge using youtube," in *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, 2015, pp. 104–114.

[41] E. Murphy-Hill, T. Zimmermann, and N. Nagappan, "Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?" in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 1–11.

[42] A. Nantes, R. Brown, and F. Maire, "A framework for the semi-automatic testing of video games." in *AIIDE*, 2008.

[43] B. Ni, H. Peng, M. Chen, S. Zhang, G. Meng, J. Fu, S. Xiang, and H. Ling, "Expanding language-image pretrained models for general video recognition," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part IV*. Springer, 2022, pp. 1–18.

[44] L. Pascarella, F. Palomba, M. Di Penta, and A. Bacchelli, "How is video game development different from software development in open source?" in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 2018, pp. 392–402.

[45] F. Petrillo, M. Pimenta, F. Trindade, and C. Dietrich, "What went wrong? a survey of problems in game development," *Computers in Entertainment (CIE)*, vol. 7, no. 1, pp. 1–22, 2009.

[46] J. Pfau, A. Liapis, G. Volkmar, G. N. Yannakakis, and R. Malaka, "Dungeons & replicants: automated game balancing via deep player behavior modeling," in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 431–438.

[47] C. Politowski, F. Petrillo, and Y.-G. Guéhéneuc, "A survey of video game testing," in *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*. IEEE, 2021, pp. 90–99.

[48] C. Politowski, F. Petrillo, G. C. Ullmann, J. de Andrade Werly, and Y.-G. Guéhéneuc, "Dataset of video game development problems," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 553–557.

[49] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, M. Hasan, B. Russo, S. Haiduc, and M. Lanza, "Too long; didn't watch! extracting relevant fragments from software development video tutorials," in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 261–272.

[50] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 139. PMLR, 2021, pp. 8748–8763.

[51] S. Roohi, C. Guckelsberger, A. Relas, H. Heiskanen, J. Takatalo, and P. Hämäläinen, "Predicting game difficulty and engagement using AI players," *Proc. ACM Hum. Comput. Interact.*, vol. 5, no. CHI, pp. 1–17, 2021.

[52] R. E. Santos, C. V. Magalhães, L. F. Capretz, J. S. Correia-Neto, F. Q. da Silva, and A. Saher, "Computer games are serious business and so is their quality: particularities of software testing in game development from the perspective of practitioners," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10.

[53] C. Schuhmann, R. Beaumont, R. Vencu, C. W. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman, P. Schramowski, S. R. Kundurthy, K. Crowson, L. Schmidt, R. Kaczmarczyk, and J. Jitsev, "LAION-5b: An open large-scale dataset for training next generation image-text models," in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[54] P. Stacey and J. Nandhakumar, "A temporal perspective of the computer game development process," *Information Systems Journal*, vol. 19, no. 5, pp. 479–497, 2009.

[55] M. R. Taesiri, M. Habibi, and M. A. Fazli, "A video game testing method utilizing deep learning," *Iran Journal of Computer Science*, vol. 17, no. 2, 2020.

[56] M. R. Taesiri, F. Macklon, and C.-P. Bezemer, "Clip meets gamephysics: Towards bug identification in gameplay videos using zero-shot transfer learning," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 270–281.

[57] C. Trivedi, A. Liapis, and G. N. Yannakakis, "Contrastive learning of generalized game representations," in *2021 IEEE Conference on Games (CoG), Copenhagen, Denmark, August 17-20, 2021*. IEEE, 2021, pp. 1–8.

[58] J. Tuovenen, M. Oussalah, and P. Kostakos, "Mauto: Automatic mobile game testing tool using image-matching based approach," *The Computer Games Journal*, vol. 8, no. 3, pp. 215–239, 2019.

[59] S. Varvaressos, K. Lavoie, S. Gaboury, and S. Hallé, "Automated bug finding in video games: A case study for runtime monitoring," *Computers in Entertainment (CIE)*, vol. 15, no. 1, pp. 1–28, 2017.

[60] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[61] M. Viggiato, D. Paas, C. Buzon, and C.-P. Bezemer, "Identifying similar test cases that are specified in natural language," *IEEE Transactions on Software Engineering*, 2022.

[62] ——, "Using natural language processing techniques to improve manual test case descriptions," in *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, 2022, pp. 311–320.

[63] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[64] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[65] M. Wortsman, G. Ilharco, J. W. Kim, M. Li, S. Kornblith, R. Roelofs, R. G. Lopes, H. Hajishirzi, A. Farhadi, H. Namkoong *et al.*, "Robust fine-tuning of zero-shot models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 7959–7971.

[66] M. Zandigohar, M. Han, D. Erdoğmuş, and G. Schirner, "Towards creating a deployable grasp type probability estimator for a prosthetic hand," in *Cyber Physical Systems. Model-Based Design*, R. Chamberlain, M. Edin Grimheden, and W. Taha, Eds. Cham: Springer International Publishing, 2020, pp. 44–58.

[67] X. Zhang and A. M. Smith, "Retrieving videogame moments with natural language queries," in *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 2019, pp. 1–7.

[68] Y. Zheng, X. Xie, T. Su, L. Ma, J. Hao, Z. Meng, Y. Liu, R. Shen, Y. Chen, and C. Fan, "Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 772–784.