How Far Can VLMs Go for Visual Bug Detection? Studying 19,738 Keyframes from 41 Hours of Gameplay Videos

Wentao Lu
wlu4@ualberta.ca
University of Alberta
Edmonton, Alberta, Canada

Alexander Senchenko
asenchenko@ea.com
Electronic Arts
Vancouver, British Columbia, Canada

Alan Sayle
asayle@ea.com
Electronic Arts
Guildford, Surrey, United Kingdom

Abram Hindle
abram.hindle@ualberta.ca
University of Alberta
Edmonton, Alberta, Canada

Cor-Paul Bezemer
bezemer@ualberta.ca
University of Alberta
Edmonton, Alberta, Canada

Abstract

Video-based quality assurance (QA) for long-form gameplay video is labor-intensive and error-prone, yet valuable for assessing game stability and visual correctness over extended play sessions. Vision language models (VLMs) promise general-purpose visual reasoning capabilities and thus appear attractive for detecting visual bugs directly from video frames. Recent benchmarks suggest that VLMs can achieve promising results in detecting visual glitches on curated datasets. Building on these findings, we conduct a real-world study using industrial QA gameplay videos to evaluate how well VLMs perform in practical scenarios. Our study samples keyframes from long gameplay videos and asks a VLM whether each keyframe contains a bug. Starting from a single-prompt baseline, the model achieves a precision of 0.50 and an accuracy of 0.72. We then examine two common enhancement strategies used to improve VLM performance without fine-tuning: (1) a secondary judge model that re-evaluates VLM outputs, and (2) metadata-augmented prompting through the retrieval of prior bug reports. Across **100 videos** totaling **41 hours** and **19,738 keyframes**, these strategies provide only marginal improvements over the simple baseline, while introducing additional computational cost and output variance. Our findings indicate that off-the-shelf VLMs are already capable of detecting a certain range of visual bugs in QA gameplay videos, but further progress likely requires hybrid approaches that better separate textual and visual anomaly detection.

CCS Concepts

• **Software and its engineering → Software defect analysis**.

Keywords

visual bug detection, gameplay videos, vision–language models, software testing, quality assurance

ACM Reference Format:
Wentao Lu, Alexander Senchenko, Alan Sayle, Abram Hindle, and Cor-Paul Bezemer. 2026. How Far Can VLMs Go for Visual Bug Detection? Studying 19,738 Keyframes from 41 Hours of Gameplay Videos. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26), July 05–09, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3803437.3805208

Figure 1: Example of a detected visual bug frame in a gameplay video using gpt-4.1-mini.

FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/2026/07
https://doi.org/10.1145/3803437.3805208

1 Introduction

Large vision–language models (VLMs) have rapidly improved in open-ended visual understanding tasks [16, 23, 27]. Their flexibility and low setup costs open up new opportunities for software quality assurance (QA), such as automatically analyzing long gameplay videos to identify visual bugs during automated QA tests.

A typical use case involves hundreds of hours of automated runs executed overnight to monitor for runtime exceptions or client instability (soak testing) [2]. In practice, however, most of the footage is never reviewed end-to-end unless a critical error is suspected. As a result, these videos form a large but underused source of evidence about overall game stability and visual quality. In this report, we present our experience with re-purposing this existing video corpus for broader QA tasks, specifically, for detecting visible in-game anomalies using VLMs. We explore whether VLM analysis can capture visual bugs from these recordings without any additional model training. Although training data can be available, our goal is to evaluate the out-of-the-box effectiveness of general-purpose VLMs as plug-and-play components in an industrial QA workflow. Meanwhile, avoiding fine-tuning also reflects practical deployment needs where game builds change frequently, and maintaining up-to-date VLM knowledge through continuous retraining would be costly and operationally unsustainable in the long run, especially when newer model generations will inevitably replace older ones.

We therefore ask a simple question: **How effectively can VLMs analyze large volumes of gameplay video to detect visual bugs, and to what extent do Retrieval-Augmented Generation (RAG) or judge-based strategies improve their performance?** We study this empirically through a case study of 100 gameplay videos (41 hours with a total of of 4,336,132 frames).

Our study demonstrates that modern VLMs can turn previously underused gameplay recordings into a practical source of QA evidence and show that a simple prompting setup enables automated triage of long-form QA videos. This approach reduces the amount of video requiring manual inspection from millions of raw frames to only a few thousand candidate frames selected by VLMs.

Our main contributions are:

- An experience-based case study demonstrating how gameplay recordings originally collected for stability testing can be re-purposed for large-scale visual QA through automated VLM analysis.
- A practical assessment of two common enhancement strategies, RAG and judge-based, applied to industrial video game QA data.

2 Related Work

Traditional bug detection. Early studies in game quality assurance concentrated on bug detection through hand-crafted rules, visual and text features, or specialized machine learning models. Lin et al. [13] conducted one of the earliest studies, examining whether gameplay videos containing bugs could be automatically identified using publicly available metadata from online platforms. Building on this foundation, Guglielmi et al. introduced *GELID* [10, 11], a method that extracts bug moments from gameplay videos by leveraging video caption texts. Similarly, Truelove et al. proposed classifying video segments as buggy or non-buggy based on video captions [29]. In subsequent research, Guglielmi et al. focused on detecting specific stuttering bug events and developed *HASTE* [9], which utilizes image similarity to identify stuttering occurrences.

In addition to caption-based methods, deep learning techniques have been employed to detect visual glitches directly from rendered frames. Ling et al. proposed a neural network approach to identify four types of rendered glitches in video games [14], by training convolutional neural networks on labeled image data. Recent work by Li et al. [31] explores visual bug detection using supervised machine learning with a hybrid co-finetuning approach that incorporates both labeled and unlabeled data. These approaches generally depend on domain-specific training data and predefined bug categories, which restricts their ability to generalize across diverse games or to detect previously unseen visual anomalies.

VLM-based bug detection. Recent advancements in large-scale vision–language models (VLMs) have opened new directions for visual bug detection. Compared to traditional approaches, VLM-based methods reduce the need for explicit ML training and manual annotation, making them suitable for a variety of visual bug detection tasks. Models such as GPT-4o demonstrate strong zero-shot understanding of complex scenes and can identify visual inconsistencies directly from raw images or video frames [23]. Early work by Taesiri et al. [28] demonstrated the effectiveness of using the Contrastive Language–Image Pre-Training (CLIP) model to identify game videos containing bugs through text queries, and language models are good as a zero-shot bug detector. They later introduced *VideoGameBunny* [25], a model fine-tuned on extensive video game image datasets and associated instruction pairs, demonstrating an improved understanding of the game context and being more accurate in-game responses. Their subsequent studies introduced GlitchBench [26] and VideoGameQA-Bench [27], which established benchmarks for evaluating VLMs on a range of game quality assurance tasks. These benchmarks demonstrated strong performance in identifying visual glitches, particularly in needle-in-a-haystack tasks that require locating bug instances within gameplay videos. Building on this line of research, our previous work [16] utilized VLMs to detect potential bug frames in gameplay videos by integrating bug descriptions from reports with keyframes extracted from videos.

Unlike prior studies that primarily rely on public or synthetically generated datasets, **our work builds upon industrial, real-world QA gameplay footage**, a continuation from our previous study [16]. We adapt the used keyframe-based strategy for extended long gameplay video by evaluating VLM performance on production-scale, long-duration gameplay recordings that are automatically collected over hundreds of hours, and by exploring methods to enhance detection accuracy without fine-tuning. Specifically, two complementary techniques are investigated: RAG and VLM as a judge model.

3 Methodology

This section outlines our approach for detecting visual bugs using VLMs and keyframes extracted from long gameplay videos. Figure 2 illustrates the steps of our approach. For each long gameplay recording, we (1) extract keyframes, (2) present each to a VLM, and (3) collect the model's decision on whether a bug is present.

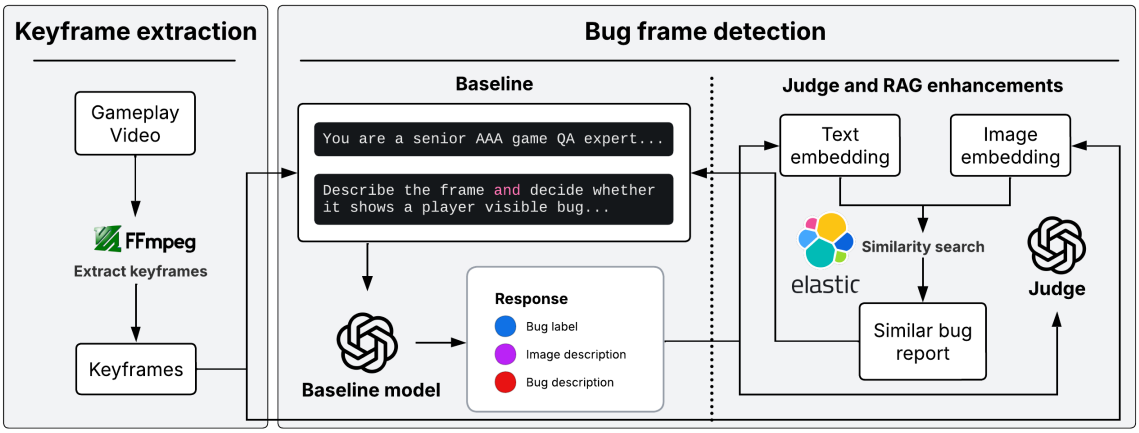

Figure 2: Overview diagram of our methodology

Table 1: Precision and accuracy across different variants and judge models

	Precision			Accuracy		
4.1-mini (Baseline)	0.50			0.72		
5-mini	0.35			0.57		
Enhancements	**4.1-mini**	**4o-mini**	**5-mini**	**4.1-mini**	**4o-mini**	**5-mini**
+ Judge	0.47	0.48	0.55	0.72	0.71	0.74
+ Image RAG	0.37	0.35	0.46	0.67	0.67	0.71
+ Text RAG (Bug description)	0.47	0.47	0.57	0.71	0.71	0.75
+ Text RAG (Image description)	0.45	0.45	0.54	0.70	0.70	0.74

Table 2: Distribution of primary bug categories (randomly sampled 1,000 results from baseline performance)

Category	Percentage (%)
Log message(s)	46.80
Other visual anomalies	53.20
- Rendering & Texture	18.60
- UI & HUD	18.10
- Animation & VFX	6.10
- Lighting & Shadow	5.90
- Physics & Collision	3.10
- Game Crash & Logic	0.70
- Performance	0.60
- Other	0.10

3.1 Keyframe extraction

Keyframes capture major visual changes in the video, enabling analysis to focus on the most representative moments of gameplay. Long QA test recordings often span tens of minutes or hours, resulting in substantial redundancy across adjacent frames. Analyzing every frame is computationally intensive and unnecessary for detecting visible anomalies. Building on our previous work [16], keyframes are extracted using FFmpeg [1] to significantly reduce the number of frames processed while preserving those most likely to contain visual anomalies. This sampling strategy achieved 98.79% bug coverage in prior evaluations, and in this study, we evaluated the same First Person Shooter (FPS) game.

3.2 Bug frame detection

The bug frame detection task is formulated as a single binary decision accompanied by a brief justification: given a keyframe, the model must classify it as buggy or normal and provide a rationalization in one or two sentences. This task instruction remains consistent across all experiments. Subsequent techniques only (1) re-evaluate the model's answer or (2) modify the information provided to the model. To enhance VLM performance in this task, two complementary strategies are incorporated, motivated by recent advances in model evaluation and context enrichment:

Judge models: We introduce secondary VLMs to re-evaluate the primary output. Previous 'LLM-as-a-Judge' studies have demonstrated that models can serve as reliable evaluators, exhibiting high agreement with human judgments [5, 6, 12, 30]. In this study, the judge model receives the frame, the decision label, and its rationale, and then either confirms or overturns the decision. This verification is primarily designed to reduce false positives in bug detection.

Metadata-augmented prompting using RAG: To expand the model's knowledge base without fine-tuning, prior bug reports most similar in visual or textual content are retrieved and appended

to the prompt context [4, 7, 15, 18]. Providing metadata and textual descriptions of expected behavior has been shown to improve VLM reasoning regarding anomalies [16, 17]. In practice, for each keyframe, a similar bug report is retrieved from the database and provided alongside the task instruction. The model then issues the same binary decision with a brief justification.

3.2.1 Baseline. The baseline serves as the simplest reference pipeline. Each extracted keyframe is directly presented to the VLM (gpt-4.1-mini [22]) using the above task instruction, producing:

- **Bug label (True/False):** Whether the frame contains a visible bug.
- **Image description:** A detailed description of the keyframe.
- **Glitch description:** A short rationale explaining the observed bug, if labeled True.

This baseline reflects the model's out-of-the-box ability to generalize to the game QA domain.

3.2.2 Judge and RAG enhancements. With the goal of improving the baseline performance, we first apply the judge step directly over the baseline results. Subsequently, we augmented the baseline approach with an additional retrieval-augmented prompting step. Three complementary retrieval strategies are used to identify the most relevant prior bug reports for each keyframe:

(1) **Image similarity:** The keyframe is embedded using a CLIP image encoder [21] and compared (based on cosine similarity between two vectors) against stored image embeddings of historical images attached to bug reports.
(2) **Image description similarity:** The description of the keyframe produced by the baseline model is embedded using a text embedder and matched against historical bug report descriptions to capture semantic similarity. Unlike the next strategy, this gives all keyframes a chance to retrieve similar reports regardless of any potential bugs.
(3) **Bug description similarity:** When the baseline model identifies a bug (label is True), its generated bug description is embedded and compared to historical report texts to retrieve examples with similar bugs.

Finally, we re-run the task using an updated prompt that includes (1) the keyframe image, (2) the baseline model's initial decision and rationale, and (3) the retrieved bug report.

4 Experimental setup

Gameplay video dataset: Our evaluation is conducted on a corpus of 100 long gameplay test videos drawn at random from daily automated runs within an industrial QA pipeline. These videos comprise a total of 41 hours of footage, encompassing 4,336,132 video frames, with a mean video length of 24 minutes and 5 seconds. Segmentation using FFmpeg [1], consistent with our prior work [16], yields 19,738 keyframes. To establish a ground truth, all extracted keyframes were manually reviewed and labelled by one of the authors. Each frame is annotated with a binary label indicating the presence or absence of a player-visible bug.

Bug categories: Unlike regular gameplay recordings, automated QA test runs constantly display on-screen log messages generated by backend logging systems. When these messages indicate critical errors or failures, they are considered bugs. Consequently, we introduced an additional category to our study beyond the seven categories used in our previous work [16], shown in Table 2.

Bug reports: In addition to this labeled gameplay video corpus, we utilize a database of historical bug reports collected from the internal JIRA system, a widely adopted platform for bug reports [3]. Each report includes a bug description, optional screenshot artifacts, and associated developer metadata. The database is pre-vectorized for internal services: text is embedded using *gte-qwen2-1.5b-instruct-embed-f16* [20], and images are embedded using *clip-vit-base-patch32* [21]. Embeddings are stored in an ElasticSearch (ES) index [8]. This vectorized database enables similarity search queries based on cosine similarity and underpins the RAG prompting strategies evaluated in Section 5. Due to legal constraints, we cannot publish our data or provide a more detailed description of the embedding strategy.

Used models: All judge and baseline models used in this study are sourced from OpenAI [24], accessed through API version "2025-04-01-preview" (the latest version at the time) [19], which provides industry-leading, production-grade VLMs. This ensures that model behavior and evaluation results are based on standardized, widely recognized architectures. We specifically select the *mini* variants to balance performance and cost, reflecting practical considerations for deployment in large-scale, production-ready QA environments. Due to internal access restrictions, we are limited to using OpenAI models.

Performance metrics: All metrics are reported as per-video means to ensure that longer recordings with more keyframes do not disproportionately influence the overall results. We focus on two key metrics: **precision** (fraction of detected bugs that are actual bugs) and **accuracy** (fraction of correct detection overall).

5 Results

5.1 Baseline performance

Motivation: Prior to evaluating enhancement strategies, the performance of a VLM on the bug detection task using only visual input is evaluated. This baseline establishes the minimum performance achievable without external guidance.

Approach: Each extracted keyframe is analyzed by the VLM using the prompt design detailed in Section 3. The gpt-4.1-mini model is selected as the baseline due to its high ranking (#3) in the previous *VideoGameQA Bench* study [27] and its cost efficiency. For comparison, we also evaluate the same baseline configuration using 5-mini, but omit 4o-mini due to its significantly higher cost (approximately 20x more tokens than 4.1-mini, as a known issue to OpenAI).

As shown in Table 1, the baseline achieves a precision of 0.50 and an accuracy of 0.72 across 100 gameplay videos. These results indicate that **off-the-shelf VLMs generalize effectively to game QA videos**. With a precision of 0.50, half of the model-flagged frames correspond to true bugs. Reviewing only the flagged frames reduces human inspection from 4,336,132 total frames to 2,002 (≈0.05%), thereby converting recordings into a manageable triage queue with candidate bug descriptions generated by the model.

In addition to performance metrics, the types of visual bugs that the VLM is capable of detecting are examined. Table 2 presents the distribution of primary detected bug categories from baseline results. Nearly half of the identified bugs (**46.8%**) are on-screen error

messages generated by backend logging systems. These overlays typically indicate lower-level system or engine errors, such as missing assets, null references, or network failures, rather than visual anomalies. However, such text overlay bugs are readily detected by the model, indicating that VLMs can effectively recognize interface overlays to diagnostic and prioritized error messages. However, these cases could potentially be addressed more efficiently using traditional optical character recognition (OCR), which would allow VLMs to focus on non-textual anomalies. The remaining **53.2%** of detected bugs comprises a diverse range of visual anomalies. The VLM can identify many of these cases based solely on visual context. Notably, visual bugs within HUD & UI often persist across multiple consecutive keyframes, resulting in duplicate detections. A promising direction for future research is to cluster bug keyframes based on visual similarity or bug type, merging redundant detections and thus effectively capturing recurring or persistent visual anomalies.

An off-the-shelf VLM can achieves 50% precision while reducing manual review to a manageable subset of frames, and discover bugs that would otherwise remain unseen in unreviewed QA recordings.

5.2 Judge and RAG performance

5.2.1 Judge models. In this setting, the judge model receives the keyframe, the baseline predicted label, and the associated reasoning, and then re-evaluates the decision.

Overall, **judge models yield mixed outcomes.** Both 4.1-mini and 4o-mini perform slightly below the baseline (precision 0.47–0.48, accuracy 0.71–0.72), whereas 5-mini demonstrates a modest improvement, achieving 0.55 precision and 0.74 accuracy. However, when 5-mini is used directly as the baseline detector model instead of 4.1-mini, its performance is substantially lower (0.35 precision, 0.57 accuracy). This result suggests that 5-mini is more effective as an evaluator than as a predictor.

5.2.2 Retrieval-augmented prompting. Results indicate that **RAG often introduces noise**. Image-based retrieval decreases performance (precision = 0.38, accuracy = 0.68) because visually similar images are often retrieved, yet the corresponding bug report descriptions are unrelated. Text-based retrieval yields improved performance, particularly when retrieved using bug description similarity and reviewed by 5-mini, achieving 0.57 precision and 0.75 accuracy. This result slightly surpasses both the baseline and the judge-only technique.

Both judge and RAG steps double inference time and cost compared to the baseline, since they require a second model call and additional embedding retrieval queries. Given the marginal accuracy gains, this highlights a trade-off between performance improvement and practical deployability in QA pipelines.

Judge and RAG strategies yield only marginal, model-specific gains. A capable judge can refine baseline outputs, whereas text-based retrieval can sometimes improve results.

6 Discussion

Comparison to prior work: Taesiri et al. introduced *VideoGameQA-Bench* [27], which features the challenging Needle-in-a-Haystack (NIAH) task. In this task, a model is required to identify anomalous frames and their corresponding timestamps in a video, referred to as "needles," which are among thousands of normal frames sampled uniformly at one frame per second. Within this framework, 4.1-mini achieved an accuracy of only 10%. Our study constitutes a variant of the NIAH task. Instead of uniform sampling, our analysis focuses on extracted keyframes that capture major visual scenes and substantially reduce redundancy. Second, our industrial QA dataset differs qualitatively. A large portion of detected anomalies are on-screen log messages, which are visually easily recognized by the model. Finally, our formulation analyzes each keyframe, similar to image-based detection in *VideoGameQA-Bench* that scores 76.8% in accuracy. Bug frames remain rare within these keyframe sequences, and our result achieves 74% accuracy, substantially outperforming the NIAH baseline but comparable to image-based glitch detection. Additionally, the use of keyframes preserves timestamp metadata from the original videos, enabling direct bug localization without requiring the model to infer.

7 Threats to Validity

Internal validity: A potential threat is the ground truth labeling process. All keyframes were labeled by a single author, which risks annotation mistakes and subjective bias. Some label noise is likely present and may affect the performance evaluation.

External validity: Our dataset consists of long QA recordings from a single game within one industrial pipeline. Other game genres or earlier development stages may exhibit different bug types and visual styles, which could change VLM behaviour.

Another threat is that our analysis is limited to OpenAI mini VLM variants because of internal access constraints. Other model families with larger-capacity models might exhibit different performance and different sensitivity to RAG and judge strategies. Similarly, our RAG setups depend on one specific embedding configuration and JIRA-based report structure. Different bug tracking systems or embedding strategies might yield different RAG quality.

8 Conclusion

In this study, we evaluated the capability of large vision-language models to detect visual bugs directly from industrial QA gameplay videos. Using a dataset of 100 videos, 41 hours and 19,738 keyframes, we showed that a simple baseline with gpt-4.1-mini can reduce manual inspection of entire videos to a manageable triage queue of frames.

We further examined two widely used strategies for improving large model performance: judge-based re-evaluation and RAG. Across the configurations we evaluated, image-based retrieval degraded performance, while text-based retrieval yielded only marginal gains. Similarly, a stronger judge (gpt-5-mini) could refine baseline results, but improvements were modest.

Overall, these findings highlight the practical potential of VLM-based triage for industrial QA workflows, substantially reducing the volume of video that requires human review. However, our findings also show that improving VLM performance is not trivial. Further progress will require more targeted hybrid designs, such as separating textual OCR, incorporating temporal reasoning over sequences of frames, and clustering redundant bug keyframes to better exploit the structure of QA gameplay recordings.

References

[1] 2025. *FFmpeg*. https://ffmpeg.org/

[2] Simon Amey. 2022. *Software Test Design: Write Comprehensive Test Plans to Uncover Critical Bugs in Web, Desktop, and Mobile Apps.* Packt Publishing, Birmingham, UK.

[3] Atlassian. 2025. *Jira*. https://www.atlassian.com/software/jira

[4] Mirco Bonomo and Simone Bianco. 2025. Visual RAG: Expanding MLLM visual knowledge without fine-tuning. arXiv:2501.10834 [cs.CV] https://arxiv.org/abs/2501.10834

[5] Dongping Chen, Ruoxi Chen, Shilin Zhang, Yaochen Wang, Yinuo Liu, Huichi Zhou, Qihui Zhang, Yao Wan, Pan Zhou, and Lichao Sun. 2024. MLLM-as-a-Judge: Assessing Multimodal LLM-as-a-Judge with Vision-Language Benchmark. In *Forty-first International Conference on Machine Learning*. https://openreview.net/forum?id=dbFEFHAD79

[6] Laura Dietz, Oleg Zendel, Peter Bailey, Charles L. A. Clarke, Ellese Cotterill, Jeff Dalton, Faegheh Hasibi, Mark Sanderson, and Nick Craswell. 2025. Principles and Guidelines for the Use of LLM Judges. In *Proceedings of the 2025 International ACM SIGIR Conference on Innovative Concepts and Theories in Information Retrieval (ICTIR)* (Padua, Italy) *(ICTIR '25)*. Association for Computing Machinery, New York, NY, USA, 218–229. https://doi.org/10.1145/3731120.3744588

[7] Zongcan Ding, Haodong Zhang, Peng Wu, Guansong Pang, Zhiwei Yang, Peng Wang, and Yanning Zhang. 2025. SlowFastVAD: Video Anomaly Detection via Integrating Simple Detector and RAG-Enhanced Vision-Language Model. arXiv:2504.10320 [cs.CV] https://arxiv.org/abs/2504.10320

[8] elastic. 2025. *OPEN SOURCE SEARCH, ANALYTICS, AND AI PLATFORM*. https://www.elastic.co/elasticsearch

[9] Emanuela Guglielmi, Gabriele Bavota, Rocco Oliveto, and Simone Scalabrino. 2025. Automatic Identification of Game Stuttering via Gameplay Videos Analysis. *ACM Trans. Softw. Eng. Methodol.* 34, 2, Article 38 (Jan. 2025), 29 pages. https://doi.org/10.1145/3695992

[10] Emanuela Guglielmi, Simone Scalabrino, Gabriele Bavota, and Rocco Oliveto. 2022. Towards Using Gameplay Videos for Detecting Issues in Video Games. arXiv:2204.04182 [cs.SE] https://arxiv.org/abs/2204.04182

[11] Emanuela Guglielmi, Simone Scalabrino, Gabriele Bavota, and Rocco Oliveto. 2023. Using gameplay videos for detecting issues in video games. *Empirical Softw. Engg.* 28, 6 (Oct. 2023), 32 pages. https://doi.org/10.1007/s10664-023-10365-0

[12] Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. 2024. LLMs-as-Judges: A Comprehensive Survey on LLM-based Evaluation Methods. arXiv:2412.05579 [cs.CL] https://arxiv.org/abs/2412.05579

[13] Dayi Lin, Cor-Paul Bezemer, and Ahmed E. Hassan. 2019. Identifying gameplay videos that exhibit bugs in computer games. *Empirical Software Engineering* (12 2019). https://doi.org/10.1007/s10664-019-09733-6

[14] Carlos García Ling, Konrad Tollmar, and Linus Gisslén. 2020. Using deep convolutional neural networks to detect rendered glitches in video games. In *Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'20)*. AAAI Press, Article 10, 8 pages.

[15] Jingyu Liu, Jiaen Lin, and Yong Liu. 2024. How Much Can RAG Help the Reasoning of LLM? arXiv:2410.02338 [cs.CL] https://arxiv.org/abs/2410.02338

[16] Wentao Lu, Alexander Senchenko, Abram Hindle, and Cor-Paul Bezemer. 2025. Automated Bug Frame Retrieval from Gameplay Videos Using Vision-Language Models. arXiv:2508.04895 [cs.SE] https://arxiv.org/abs/2508.04895

[17] Finlay Macklon and Cor-Paul Bezemer. 2025. Exploring the Capabilities of Vision-Language Models to Detect Visual Bugs in HTML5 <canvas> Applications. arXiv:2501.09236 [cs.SE] https://arxiv.org/abs/2501.09236

[18] Aigerim Mansurova, Aiganym Mansurova, and Aliya Nugumanova. 2024. QA-RAG: Exploring LLM Reliance on External Knowledge. *Big Data and Cognitive Computing* 8, 9 (2024). https://doi.org/10.3390/bdcc8090115

[19] Microsoft. 2025. *Azure*. https://azure.microsoft.com/en-ca

[20] Ollama. 2024. *gte-Qwen2-1.5B-instruct*. https://ollama.com/rjmalagon/gte-qwen2-1.5b-instruct-embed-f16

[21] OpenAI. 2021. *clip-vit-base-patch32*. https://huggingface.co/openai/clip-vit-base-patch32

[22] OPENAI. 2024. *GPT-4.1 mini*. https://platform.openai.com/docs/models/gpt-4.1-mini

[23] OpenAI. 2024. *Hello GPT-4o*. https://openai.com/index/hello-gpt-4o

[24] OpenAI. 2025. *OpenAI*. https://openai.com/

[25] Mohammad Reza Taesiri and Cor-Paul Bezemer. 2025. VIDEOGAMEBUNNY: Towards vision assistants for video games. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (2025-03-01).

[26] Mohammad Reza Taesiri, Tianjun Feng, Cor-Paul Bezemer, and Anh Nguyen. 2024. GlitchBench: Can Large Multimodal Models Detect Video Game Glitches?. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 22444–22455.

[27] Mohammad Reza Taesiri, Abhijay Ghildyal, Saman Zadtootaghaj, Nabajeet Barman, and Cor-Paul Bezemer. 2025. VideoGameQA-Bench: Evaluating Vision-Language Models for Video Game Quality Assurance. arXiv:2505.15952 [cs.CV] https://arxiv.org/abs/2505.15952

[28] Mohammad Reza Taesiri, Finlay Macklon, and Cor-Paul Bezemer. 2022. CLIP meets GamePhysics: Towards bug identification in gameplay videos using zero-shot transfer learning. In *International Conference on Mining Software Repositories (MSR)* (2022-03-24).

[29] Andrew Truelove, Shiyue Rong, Eduardo Santana de Almeida, and Iftekhar Ahmed. 2023. Finding the Needle in a Haystack: Detecting Bug Occurrences in Gameplay Videos. arXiv:2311.10926 [cs.SE] https://arxiv.org/abs/2311.10926

[30] Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. 2024. Replacing Judges with Juries: Evaluating LLM Generations with a Panel of Diverse Models. arXiv:2404.18796 [cs.CL] https://arxiv.org/abs/2404.18796

[31] Faliu Yi, Sherif Abdelfattah, Wei Huang, and Adrian Brown. 2025. A hybrid co-finetuning approach for visual bug detection in video games. In *Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (Edmonton, Alberta, Canada) *(AIIDE '25)*. AAAI Press, Article 17, 11 pages. https://doi.org/10.1609/aiide.v21i1.36820