OSSGameBench: A Large-Scale Dataset of Contributor Activities in Open-Source Video Games

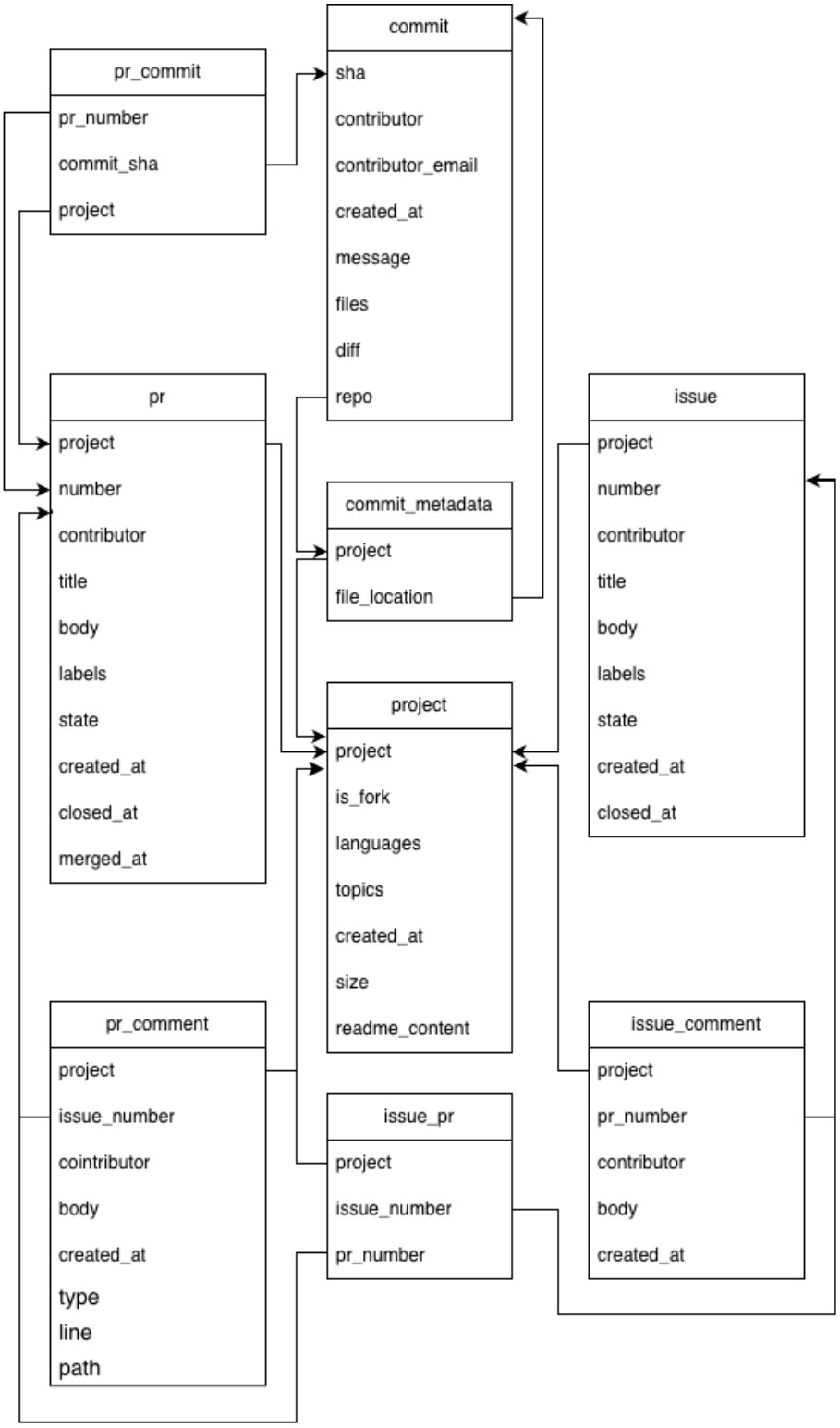
Online Appendix

1. Abstract

Video games are a distinct type of software that engage users through deeply immersive and interactive experiences. Yet, unlike other types of software where standardized benchmark datasets have accelerated research on downstream tasks such as defect localization and repair, the game development domain lacks comparable resources that reflect its distinctive characteristics. For instance, defects in games often arise from the complex interplay between source code (e.g., C++ files) and non-code assets (e.g., graphics files), whereas in other software, such complications are less prevalent. Therefore, specialized datasets are required to capture these multidisciplinary demands unique to game development. Unavailability of such specialized datasets limits the analysis, development, and evaluation of data-driven solutions tailored to game development. To fill this void, we introduce OSSGameBench, a curated benchmark dataset derived from 950 open-source repositories in GitHub, encompassing both playable video games and essential game development tools (e.g., engines, frameworks). The dataset contains issues, comments, commits, pull requests, and review comments, with explicit mappings among these entities to enable future research aimed at improving the full spectrum of game development.

In this document, we provide extra information about our dataset for anyone who is interested in using it.

2. Dataset Schema



Projects

Entity name

project

Attributes

Attribute name	Description
project	The full name of the project repository in the format owner/repo (e.g., AlmasB/FXGL).
is_fork	Indicates whether the repository is a fork of another project (`true` or `false`).
languages	A dictionary mapping programming languages to the number of bytes of code written in each language (e.g., `{"Java": 515464, "Shell": 1702, "Rich Text Format": 928}`).
topics	A list of topics or tags associated with the repository on GitHub (e.g., `["game", "java", "2d-games"]`).
created_at	The timestamp representing when the repository was created.
readme_content	Content of the `README` file, if available

Relative path to the data

`OSSGameBench/project.parquet`

Open the data in Python

```
import pandas as pd
projects = pd.read_parquet("OSSGameBench/project.parquet")
```

Issue reports

Entity name

issue

Attributes

Attribute name	Description
project	The full name of the project repository in the format owner/repo (e.g., AlmasB/FXGL).
number	Issue number (int)
contributor	Issue author’s login/display name.
title	Issue title.
body	Issue description text (may be empty).
labels	List of label names (strings).
state	open / closed
created_at	Creation timestamp
closed_at	Close timestamp (nullable)

Note that one repo can contain multiple issues. Issue entity can be linked to project entity by <repo> attribute.

Relative path to the data

```
OSSGameBench/issue.parquet
```

Open the data in Python

```
import pandas as pd
issues = pd.read_parquet(“OSSGameBench/issue.parquet")
```

Issue comments

Entity name

issue_comment

Attributes

Attribute name	Description
project	owner/name
issue_number	Issue number (int).
contributor	Comment author.
body	Comment text.
created_at	Comment timestamp

Note that one issue can have multiple comments, and comments can be ordered by the created_at time. Issue_comment entity can be linked to issue entity by <repo,issue_number>. attributes.

Relative path to the data

```
OSSGameBench/issue_comment.parquet
```

Open the data in Python

```
import pandas as pd
issue_comment_mapping =
pd.read_parquet("OSSGameBench/issue_comment.parquet")
```

Pull request

Entity name

pr

Attributes

Attribute name	Description
project	The full name of the project repository in the format owner/repo (e.g., AlmasB/FXGL).
number	PR number (int)
contributor	PR author’s login/display name.
title	PR title.
body	PR description text (may be empty).
labels	List of label names (strings).
state	open / closed
created_at	Creation timestamp
closed_at	Close timestamp (nullable)
merged_at	PR merge timestamp (nullable)

Note that one repo can contain multiple PRs. pr entity can be linked to the project entity by <repo> attribute.

Relative path to the data

`OSSGameBench/pr.parquet`

Open the data in Python

```
import pandas as pd
prs = pd.read_parquet("OSSGameBench/pr.parquet")
```

PR comments

Entity name

pr_comment

Attributes

Attribute name	Description
project	owner/name
pr_number	PR number (int).
contributor	Comment author.
body	Comment text.
created_at	Comment timestamp
type	“discussion” or “review”
path	If type==“review”: The path contains the file path of the file to which the review comment is applicable.
line	If type==“review”: The line contains the line of the file to which the review comment is applicable.

Note that one PR can have multiple review comments, and comments can be ordered by the created_at time. pr_comment entity can be linked to pr entity by <repo,pr_number>. attributes.

Relative path to the data

```
OSSGameBench/pr_comment.parquet
```

Open the data in Python

```
import pandas as pd
pr_review_comments =
pd.read_parquet("OSSGameBench/pr_comment.parquet")
```

Issue-PR mapping

Entity name

issue_pr

Attributes

Attribute name	Description
project	owner/name
issue_number	Issue number (int).
pr_number	PR number (int).

Note that issues to PRs mapping is many-to-many. One issue can have multiple PRs and vise versa.

Relative path to the data

`OSSGameBench/issue_pr.parquet`

Open the data in Python

```
import pandas as pd
issue_pr_mapping= pd.read_parquet("OSSGameBench/issue_pr.parquet")
```

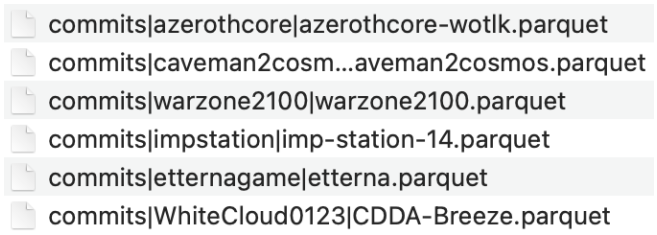
Commits metadata

Entity name

commit_metadata

Attributes

Note that the commit dataset is over 300 GB, and thus, for easy loading purposes, we store the commits of each project in a separate file. Below are some example files. Note that each file is a `parquet` file. We use `parquet` files for compression purposes. The link between these files are repos are included in this entity.



Each file contains a list of commits for that project. Below we show the list of attributes per commit.

Attribute name	**Description**
project	owner/name
file_location	Relative path to the repo's `parquet` file which includes all the commits for a given repository.

Relative path to the data

`OSSGameBench/commit_metadata.parquet`

Open the data in Python

```
import pandas as pd
commit_metadata =
pd.read_parquet("OSSGameBench/commit_metadata.parquet")
```

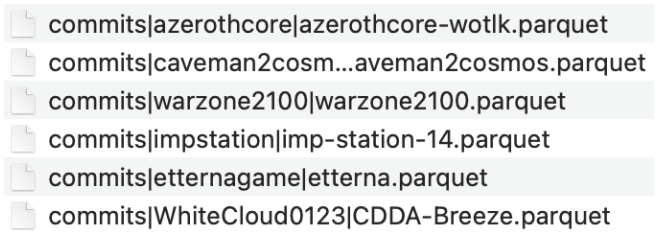
Commits

Entity name

commit

Attributes

Note that the commit dataset is over 300 GB, and thus, for easy loading purposes, we store the commits of each project in a separate file. Below are some example files. Note that each file is a `parquet` file. We use `parquet` files for compression purposes. Please follow the instructions in the next section to open these files in Python.



Each file contains a list of commits for that project. Below we show the list of attributes per commit.

Attribute name	**Description**
repo	owner/name
sha	Commit hash/identifier.
contributor	Commit author name/login
contributor_email	Author email (if available).
created_at	Comment timestamp
message	Commit message.
files	List/array of changed file paths
diff	per-file patch

Relative path to the data

`OSSGameBench/commits/commit|<repo_owner_name>|<repo_name>.parquet`

Open the data in Python

```
import pandas as pd
import json
import numpy as np
def normalize_files(x):
    if isinstance(x, str):
```

```
        try:
            return json.loads(x)  # Convert JSON string to Python list
        except json.JSONDecodeError:
            return [x]  # fallback if not valid JSON
    return x  # already a list or iterable

filename = "OSSGameBench/filename.parquet"

if filename != "OSSGameBench/commits/commits|4ian|gdevelop.parquet":
         # Case 1: normal single parquet file
         commits = pd.read_parquet(filename)
else:
         # Case 2: specific project with 4 split parquet files
         commits_1 = pd.read_parquet("OSSGameBench/commits/commits|4ian|gdevelop|1.parquet")
         commits_2 = pd.read_parquet("OSSGameBench/commits/commits|4ian|gdevelop|2.parquet")
         commits_3 = pd.read_parquet("OSSGameBench/commits/commits|4ian|gdevelop|3.parquet")
         commits_4 = pd.read_parquet("OSSGameBench/commits/commits|4ian|gdevelop|4.parquet")

         # Combine them into one DataFrame
         commits = pd.concat([commits_1, commits_2, commits_3, commits_4], ignore_index=True)

# Apply normalization safely (if columns exist)
for col in ["diff", "files"]:
         if col in commits.columns:
               commits[col] = commits[col].map(normalize_files)
```

Note that these pre-processing steps are essential before using.

PR - commit mapping

Entity name

pr_commit

Attributes

Attribute name	Description
project	owner/name
pr_number	PR number (int)
commit_sha	Commit hash/identifier.

Relative path to the data

`OSSGameBench/pr_commit.parquet`

Open the data in Python

```
import pandas as pd
pr_commit_mapping = pd.read_parquet("OSSGameBench/pr_commit.parquet")
```

3. Analysis: Game genres

Introduction

A video game can belong to one or more genres, as defined in earlier research (Swacha 2025).[1] The list of genres is as follows:

Table A.1: Gameplay types

Type	Description	Subtypes
Action	Games that revolve around a fast-paced experience, often emphasizing reaction-based challenges in terms of how the player interacts with the game world.	Action–Adventure, Arcade, Block Breaking, Brawler, Fighting, Hack and Slash, Multiplayer Online Battle Arena, Music, Party, Platform, Stealth, Survival, Vehicle Combat
Puzzle	Games that emphasize solving puzzles and/or organizing pieces.	Block Fill, Hidden Object, Match Puzzles, Point and Click, Word Puzzles
Rhythm	Games that involve the player inputting commands or completing actions while synchronizing to a rhythm.	–
Role-Playing	Games related to tabletop role-playing, involving a heavy focus on the statistical advancement (like "leveling up") of characters, combined with exploration.	Japanese RPG, Massively Multiplayer Online RPG, Rogue-Like, Western RPG
Simulation	Games that simulate actions or situations based on either an existing or fictional reality.	Breeding, Construction and Management Simulation, Flight Simulator, God Game, Interactive Movie, Programming Game, Sandbox, Social Simulator, Virtual Life, Visual Novel
Sports	Games that simulate real-world sports.	Racing
Shooter	Games that are based on a shooting mechanic where players target and shoot objects or enemies to progress.	First-Person Shooter, Light-Gun Shooter
Strategy	Games that revolve around strategic or tactical planning, often involving building, resource management, and exploration components.	4X, Military Simulator, Real-Time Strategy, Tactics, Tower Defense, Turn-Based Strategy
Traditional	Games based on mechanics that exist in the real world and can be played in a physical setting.	Board Games, Card Games, Exercise, Gambling, Game Shows, Mazes, Pinball, Puzzles, Trivia Games

The genre of a game is not explicitly available in the metadata of projects (i.e., the project entity of our dataset). However, the `README`-file content can contain such information yet in unstructured textual format. To extract this information, we perform a *directed content analysis* (Humble 2009) to classify `README` files in our dataset into game genres based on their textual content. A directed content analysis is a qualitative method in which predefined categories guide the coding of textual artifacts (Humble 2009). In our analysis, the goal is to classify each `README` file into one or more predefined categories in Table A.1. Below, we discuss the approach, evaluation, and results of this classification.

[1] https://github.com/uwgamergroup/vocabulary-gameplay-genre/

Details of the coding process

In fact, two coders classify the `README` files of 100 randomly sampled game projects; one coder is a human (the second author), and the other is an LLM (OpenAI GPT-4.1).[2] The human coder reviewed Table A.1 to understand how each genre is described. Then, they investigated the text of each `README` file in our sample and categorized it accordingly. The second coder, i.e., OpenAI GPT-4.1-mini model, was provided the following prompt for each `README` file content.

> You are a video game developer.
> Determine whether the README text discusses the genre of the game
> Use ONLY the genres defined in the JSON below.
> If a subtype/example is mentioned, map it to its parent 'genre'.
> You can guess the genre, but it should be one of the genres in the JSON below.
> Return strictly one of the following:
> - A comma-separated list of genre names exactly as they appear in the 'genre' field, OR
> - 'Not found' if no genre is mentioned (or information is not sufficient).
>
> GENRE: <CONTENT OF TABLE A.1>
> README TEXT: <TEXT OF THE `README` FILE>

Evaluation of the coding agreement

After coding, we compute the inter-coder reliability using Krippendorff's α,[3] a widely used reliability coefficient that corrects for chance agreement and generalizes to multiple coders, missing data, and different levels of measurement. This metric is particularly suitable for assessing agreement in multi-label coding tasks, such as genre classification, where a single project may belong to more than one genre. Specifically, we calculate α separately for each genre and report both the macro-averaged and prevalence-weighted average coefficients to capture the overall agreement between the human and LLM coders.

Our evaluation resulted in an average coefficient value of 0.67 across all predefined genres, indicating moderate-to-substantial agreement between the coders. With this agreement, we extend the LLM script to code the README files of all the projects in our dataset. This information is now available in our dataset for validation and replication purposes.

[2] https://openai.com/index/gpt-4-1/
[3] https://repository.upenn.edu/handle/20.500.14332/2089

References

Humble, Áine M. 2009. “Technique Triangulation for Validation in Directed Content Analysis.” *International Journal of Qualitative Methods* 8 (3): 34–51.

Swacha, Jakub. 2025. “The Relative Popularity of Video Game Genres in the Scien- Tific Literature: A Bibliographic Survey.” *Multimodal Technologies and Interaction* 9.