Steam User Review Sentiment Analysis

Christopher Finn

A. Research Question and Context for the Project

Game development has become an increasingly competitive marketplace, especially on the PC platform, where the Steam marketplace hosts the majority of games in the industry. On average, roughly 40 new games are released on the platform every day (Team, 2025). Many games go completely unnoticed or follow unpopular trends, leading to negative reviews and limited visibility with a larger gaming audience.

Sentiment analysis of user reviews across different genres and competitors' games can help indie developers understand what design choices and features resonate positively with players. By identifying common pitfalls and points of negative feedback from players, developers can make informed decisions during development that could increase the likelihood that their game will stand out in an increasingly saturated market.

The goal of the project is to determine whether a sentiment analysis model can achieve statistically significant results, with an accuracy of 90% or higher, in classifying the polarity of Steam user reviews, while also producing actionable insights into common design pitfalls and feature-related issues in indie games.

The hypotheses of this research are as follows:

Null Hypothesis - The sentiment analysis model will not achieve statistically significant accuracy ($\geq$ 90%) in classifying the polarity of Steam user reviews.

Alternate Hypothesis - The sentiment analysis model will achieve statistically significant accuracy ($\geq$ 90%) in classifying the polarity of Steam user reviews.

By examining this research question, the project aims to demonstrate how NLP techniques can support data-driven decision-making in the game development process. The insights derived from this analysis could help indie developers make informed decisions to improve player satisfaction and increase the overall success of future titles.

B. Data Collection

For this research project, the Steam Games Metadata and Player Reviews (2020-2024) dataset will be used (Abdelqader, 2025). This dataset is publicly available and can be downloaded from the Mendeley Data website. The dataset will be used in its entirety, with filtering applied later during data engineering.

Using a dataset like this has several advantages and disadvantages:

Advantages

- The dataset is extremely large and comprehensive. It covers all user reviews for games released between 2020 and 2024. This allows the sentiment analysis model to be trained on a wealth of text data.
- As the dataset spans such a wide range, it captures real-world player views across diverse genres. This can yield valuable insights into which games are performing well in the market.

Disadvantages

- The data does not cover any newly released titles from 2025 or anything older than 2020. The majority of players (roughly 90%) appear to be playing games that are at least 6 years old (Porter, 2024). This is a significant section of the playerbase and is unaccounted for in this dataset.

The dataset does present several challenges. First, the dataset is large, over 10 gigabytes once uncompressed. This will cause memory issues on most free neural network platforms, such as Google Colab. This was addressed by sampling subsets of data to maintain a manageable computational cost while preserving sentiment diversity. Another challenge of the dataset was its multilingual nature. The primary model used in this research is a DistilBERT model, a pre-trained sentiment analysis network trained on English text only. To address this, the CSV files were run through the Python library langid for English detection, which proved robust and retained the majority of English-language reviews.

C. Data Extraction and Preparation

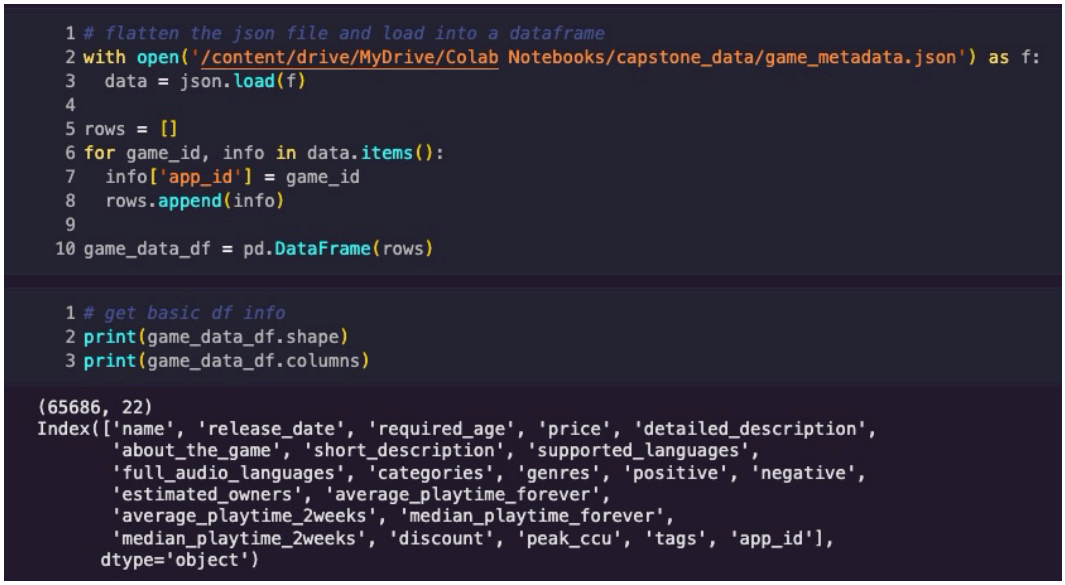
After collecting the dataset, several preparation and extraction steps were utilized to transform the raw dataset into a structured format suitable for model training. Each step of the process (data extraction, cleaning, tokenization, and splitting) is described below, along with screenshots and justifications for the tools and techniques used.

Data Extraction

The dataset was downloaded from the Mendeley Data website as a set of CSV files containing the reviews and a metadata JSON file containing summary information (e.g., number of reviews, game title, game description) for the specific games. Using the Pandas and JSON libraries, the JSON file was flattened and loaded into a DataFrame.

Figure 1

Flattening and loading of the JSON Metadata File



```
# flatten the json file and load into a dataframe
with open('/content/drive/MyDrive/Colab Notebooks/capstone_data/game_metadata.json') as f:
  data = json.load(f)

rows = []
for game_id, info in data.items():
  info['app_id'] = game_id
  rows.append(info)

game_data_df = pd.DataFrame(rows)
```

```
# get basic df info
print(game_data_df.shape)
print(game_data_df.columns)
```

```
(65686, 22)
Index(['name', 'release_date', 'required_age', 'price', 'detailed_description',
       'about_the_game', 'short_description', 'supported_languages',
       'full_audio_languages', 'categories', 'genres', 'positive', 'negative',
       'estimated_owners', 'average_playtime_forever',
       'average_playtime_2weeks', 'median_playtime_forever',
       'median_playtime_2weeks', 'discount', 'peak_ccu', 'tags', 'app_id'],
      dtype='object')
```

Once the JSON was loaded, the data was further filtered to create a list of app IDs containing only apps with reviews. This new filtered dataframe, called df_with_reviews, would serve as a filter for the 24000 CSV files in the dataset to speed up processing.

Figure 2

Creation of the df_with_reviews dataframe

```
# create data frames to seperate out games that have no reviews
df_zero_reviews = game_data_df[(game_data_df['positive'] + game_data_df['negative']) == 0]
df_with_reviews = game_data_df[(game_data_df['positive'] + game_data_df['negative']) > 0]
```

```
# create a reviews dataframe with greater than 100 reviews
df_with_reviews = df_with_reviews[df_with_reviews['positive'] + df_with_reviews['negative'] > 100]
print(df_with_reviews.shape)
```

```
(9083, 23)
```

The CSV files were prepared by loading the file names into a list to compare them with the df_with_reviews dataframe and ensure only matching files moved on to the further cleaning steps. Using Pandas for both of these tasks was a significant **advantage**, as the library works well with both large tabular datasets and JSON files. The one considerable **disadvantage** of using Pandas for this task, however, was the size of the data. This project involved multiple gigabytes of CSV files, so trying to load all of them into a single dataframe at once would easily crash Google Colab.

Data Cleaning

The review text was standardized using Pandas string operations and regular expressions. Checks were put in place to remove missing and duplicate information; however, the dataset did not contain either of these problems.

Figure 3

Main CSV processing loop

```
# --- main loop ---
for file in tqdm(file_list, desc='Processing CSVs'):
    app_id = file.split('_')[0]
    if app_id not in valid_app_ids:
        continue
    if app_id in processed_app_ids:
        continue
    if app_id in skipped_app_ids:
        continue

    # safe read
    file_path = os.path.join(review_path, file)
    try:
        df = pd.read_csv(file_path, encoding='utf-8')
    except UnicodeDecodeError:
        df = pd.read_csv(file_path, encoding='latin-1')
    except pd.errors.ParserError:
        print(f"Skipping {file}: ParserError")
        skip_log.append({"app_id": app_id, "reason": "ParserError"})
        save_log(skip_log, skip_log_path)
        continue

    # column chek
    cols_to_keep = ['post_date','review','recommend','playtime','early_access_review']
    if not set(cols_to_keep).issubset(df.columns):
        print(f"Skipping {file}: missing columns")
        skip_log.append({"app_id": app_id, "reason": "missing_columns"})
        save_log(skip_log, skip_log_path)
        continue
    df = df[cols_to_keep]

    # sample language check
    non_null_reviews = df['review'].dropna()
    if len(non_null_reviews) == 0:
        log_skip(file, 0)
        skip_log.append({"app_id": app_id, "reason": "no_non_null_reviews"})
        save_log(skip_log, skip_log_path)
        continue
```

Additionally, since the dataset was multi-lingual, the Langid Python library was used to detect language and retain only English reviews for compatibility with the DistilBERT model used later for training. If the review files did not meet a threshold for English reviews, they were skipped because they would not contribute enough to the model's training to justify the computational cost.

Figure 4

Creation of the detect_language function

```
def detect_language(text):
  # convert input to string to handle non-string types
  text = str(text)
  if len(text) < 15:
    return "und" # undetermined
  try:
    lang, confidence = langid.classify(text)
    return lang
  except:
    return 'error'
```

Figure 5

Language detection and English review ratios

```
# sample language check
non_null_reviews = df['review'].dropna()
if len(non_null_reviews) == 0:
    log_skip(file, 0)
    skip_log.append({"app_id": app_id, "reason": "no_non_null_reviews"})
    save_log(skip_log, skip_log_path)
    continue

sample = non_null_reviews.sample(n=min(sample_size, len(non_null_reviews)), random_state=23)
lang_results = sample.apply(detect_language)
english_ratio = (lang_results == 'en').mean()

if english_ratio < english_threshold:
    log_skip(file, english_ratio)
    skip_log.append({"app_id": app_id, "reason": "low_english_ratio"})
    save_log(skip_log, skip_log_path)
    continue

if len(df) < 10:
    print(f"Skipping {file}: low volume dataframe")
    skip_log.append({"app_id": app_id, "reason": "low_volume_of_reviews"})
    save_log(skip_log, skip_log_path)
    continue
```

Lastly, the main cleaning loop of the process would filter for English reviews, then normalize the data by applying string operations such as lowercasing all text, ensuring numeric columns were numeric data types, and converting date columns to datetime data types.

Figure 6

Normalization and English filtering

```
# clean and filter for english
df = df[df['review'].notna() & df['review'].str.strip().ne("")]
df['lang'] = df['review'].apply(detect_language)
df = df[df['lang'] == 'en']

# normalize
df['recommend'] = df['recommend'].fillna('').astype(str).str.lower().str.startswith('rec').astype(int)
df['playtime'] = pd.to_numeric(df['playtime'], errors='coerce')
df['post_date'] = pd.to_datetime(df['post_date'], errors='coerce')
df['early_access_review'] = df['early_access_review'].fillna(False).astype(bool)
df['app_id'] = app_id
```

Due to connection timeouts, regular checkpoints in the form of parquet files and logs were created after a specific number of CSV files had been processed by the cleaning function. This safeguarded against session timeouts in Google Colab and created an easy-to-use checkpoint system for the project.

Figure 7

Checkpoint system

```
# path to parquet files
batch_files = glob.glob('/content/drive/MyDrive/Colab Notebooks/capstone_data/review_batches/*.parquet')
print(f"Found {len(batch_files)} batch files.")

# combine files into one dataframe
df_review = pd.concat([pd.read_parquet(file) for file in batch_files], ignore_index=True)

# structure chek
print(f"Total reviews: {len(df_review)}")
print(f"Unique games: {df_review['app_id'].nunique()}")
print(df_review.head())
```

```
Found 23 batch files.
Total reviews: 10152931
Unique games: 6763
   post_date                                          review  recommend  \
0 2021-07-10                                We need Chinese          1
1 2020-04-11  While Cook, Serve, Delicious! was already one ...          1
2 2021-12-25      WE NEED CHINESE !!!!!!!!!!!!!!!!!!!!!!!!!!!          1
3 2022-07-04  It's not "fun" in the traditional sense, espec...          1
4 2020-08-03  CSD2 is really good, but it doesn't hold a can...          1

   playtime  early_access_review lang   app_id
0       5.1                False   en  1000030
1     154.3                False   en  1000030
2       0.7                False   en  1000030
3      31.6                False   en  1000030
4      39.4                 True   en  1000030
```

Overall, this process worked well enough, but took an extremely long time to finish. Over twenty-four hours in fact. One **advantage** of this process was the use of the Langid library. It did a good job of classifying English reviews and made filtering the multi-lingual dataset much

easier. However, a significant **disadvantage** of this process was the computational overhead when applied to the dataset's nearly ten million lines of review text.

Further Cleaning, Normalization, and Label Preparation

After filtering for English reviews, additional preprocessing steps were performed to normalize the dataset and get it ready for model training. These included cleaning and encoding review fields, removing low-quality or duplicate text, and generating balanced sample sets of positive and negative reviews.

To reduce overall noise in the data, reviews shorter than 20 characters were removed, and any duplicate entries were dropped using Pandas drop_duplicates(). This prevented identical or small snippets of a review from skewing the model.

Figure 8

Downsizing and trimming duplicates

```
# downsizing and trimming duplicates
df_review = df_review[df_review['review'].str.len() > 20]
df_review = df_review.drop_duplicates(subset=['review'])
print(f"Total reviews: {len(df_review)}")
print(f"Unique games: {df_review['app_id'].nunique()}")
```

```
Total reviews: 9429899
Unique games: 6763
```

An **advantage** of this approach was that these actions would improve model generalization by ensuring high-quality input. However, removing short reviews risks losing meaningful small reviews and reducing data diversity.

Figure 9

Downsampling datasets

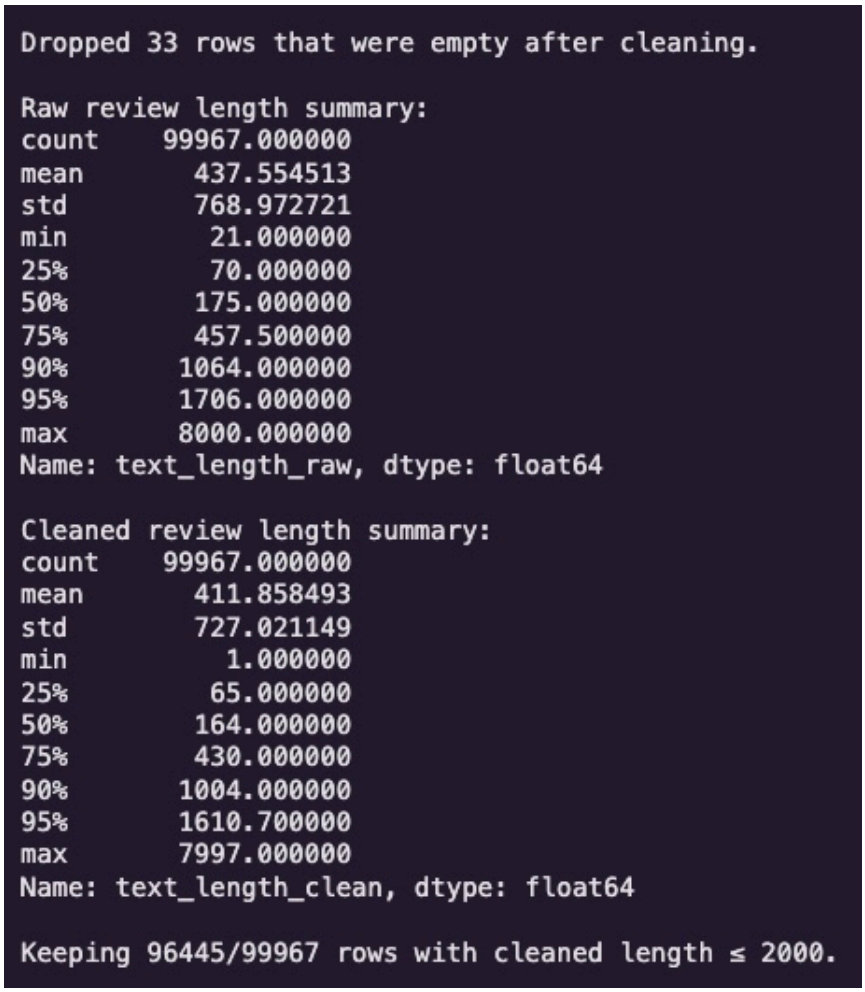
```
# sampling datset to keep roughly 100k records 50% positive/50% negative
pos = df_review[df_review['recommend'] == 1].sample(n=50000, random_state=23)
neg = df_review[df_review['recommend'] == 0].sample(n=50000, random_state=23)
df_samples = pd.concat([pos, neg]).sample(frac=1, random_state=23).reset_index(drop=True)
```

This balanced data would improve the classification accuracy and the reliability of the metrics produced by the model. The downside of this approach is the omission of potentially informative examples during downsampling.

Next, another round of cleaning on the final text reviews was performed before tokenization. A custom clean_text() function was created that removed many common issues in real-world text, such as URLs and special characters, collapsing repeated characters, removing single-letter tokens, and trimming whitespace. The NLTK word corpus was imported as well to maintain a reference list of valid English words for later checks. Outliers in review length were also capped at under 2000 characters to reduce training time and memory use. This still retained 99% of the data and kept the reviews to a more manageable length.

Figure 10

Cleaning results



Both Regex and NLTK offer tools for precise cleaning, and both work well in this context. Regex offers precise rule control, while NLTK adds linguistic resources for more robust cleaning. The downside to this is the computational cost. Regex operations are slower on very large text data, which this dataset falls into.

Tokenization and Dataset Preparation

After data cleaning and balancing, the next stage involved converting the review text into numerical representations for use as input to the DistilBERT sentiment analysis model. This required tokenizing the text, splitting the dataset into training, validation, and testing sets, and creating PyTorch dataset wrappers for model ingestion.

The cleaned dataset was first partitioned into three subsets using the scikit-learn library's train-test-split function. 80% of the data was used for training, with the remaining 20% split evenly between the validation and test datasets. The samples were stratified to preserve equal proportions of positive and negative reviews across each subset.

Figure 11

Train, Validation, Test split

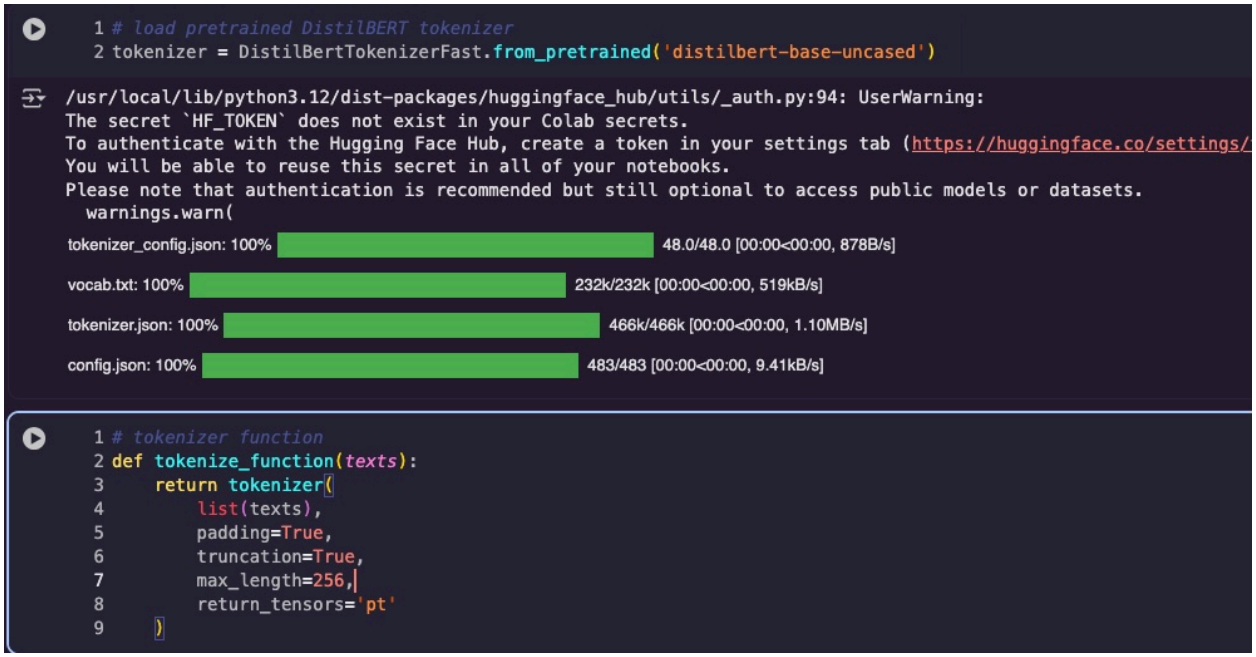
```
1 # split into training (80%) and temporary (20%) sets (10% val 10% test)
2 df_train, df_temp = train_test_split(df_final, test_size=0.2, random_state=23, stratify=df_final['recommend'])
3 df_val, df_test = train_test_split(df_temp, test_size=0.5, random_state=23, stratify=df_temp['recommend'])
4
5 print(f"Train: {len(df_train):,}  Val: {len(df_val):,}  Test: {len(df_test):,}")

Train: 77,156  Val: 9,644  Test: 9,645
```

Sci-kit learn made this process very simple and ensured the datasets would be reproducible and balanced across multiple runs of the code. However, it does require the entire dataset to reside in memory, which can be problematic for large text datasets. This was a problem for this project, but since prior checkpoints were created as Parquet files, refreshing the session allowed the process to complete correctly.

For tokenization, the DistilBERT tokenizer from the Hugging Face Transformers library was used to convert the cleaned review text into token IDs and attention masks (Wolf et al., 2020). A custom tokenize_function() wrapped the tokenizer to handle padding, truncation to a max length of 256 tokens, and batch tensor conversion for PyTorch compatibility.

Figure 12

Loading the DistilBERT tokenizer and defining a tokenize function



Tokenizing the data preserves contextual meaning by using subword embeddings consistent with the pretrained DistilBERT model's vocabulary. A **disadvantage** of this approach is that it is technically computationally intensive, but it was not a problem for the Google Colab platform.

Next, label encoding and tensor conversion were applied to the datasets. The sentiment labels (0 for negative and 1 for positive) were converted to PyTorch tensors for model training.

Figure 13

Encoding and converting to Tensors

This is a necessary step and allows for integration with PyTorch's training loops.

The final step before jumping into the actual training was creating a custom SteamReviewsDataset class wrapper utilizing the torch.utils.data.Dataset library. This wrapper helps to align the encoded text and label tensors and supports indexing and batching during the training loop. A separate dataset object was created for each of the training, validation, and testing sets.

Figure 14

Creating the dataset class and running the encodings through it

```python
# dataset wrapper
class SteamReviewsDataset(Dataset):
  def __init__(self, encodings, labels):
    self.encodings = encodings
    self.labels = labels

  def __len__(self):
    return len(self.labels)

  def __getitem__(self, idx):
    item = {key: val[idx] for key, val in self.encodings.items()}
    item['labels'] = self.labels[idx]
    return item
```

```python
# dataset creation
train_dataset = SteamReviewsDataset(train_encodings, y_train)
val_dataset = SteamReviewsDataset(val_encodings, y_val)
test_dataset = SteamReviewsDataset(test_encodings, y_test)
```

An **advantage** of this approach is that it is the simplest data handling during the training loop and ensures a consistent pairing of inputs and labels. The downside is the increased implementation time and the possibility of debugging if the dataset lengths become misaligned (Paszke et al., 2019)

Once all of these steps were complete, the data was fully pre-processed, balanced, and formatted for neural network training. The next stage of the implementation is fine-tuning the DistilBERT model and evaluating its performance.

D. Analysis

The analytical phase focused on fine-tuning the DistilBERT sentiment classification model on the cleaned Steam reviews dataset. The goal was to train the model to classify reviews as either Recommended (positive) or Not Recommended (negative) and to evaluate the model's accuracy, precision, recall, and F1 score against the 90% threshold defined in the hypothesis.

Model Initialization and Configuration

- Model - DistilBERT-base-uncased pre-trained model from Hugging Face Transformers, configured for two-label classification.
- Hardware - Model executed on GPU (Google Colab Tesla T4)
- Key Parameters
    - Learning Rate = 2 x 10^{-5}
    - Epochs = 3
    - Training batch Size = 16
    - Validation batch size = 32
    - Weight decay = 0.01
    - Evaluation strategy = epoch
    - Metric for best model = F1 score

Figure 15

Training Arguments

```
# training config
training_args = TrainingArguments(
    output_dir='/content/drive/MyDrive/Colab Notebooks/capstone_data/results',
    eval_strategy='epoch',
    save_strategy='epoch',
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir='/content/drive/MyDrive/Colab Notebooks/capstone_data/logs',
    load_best_model_at_end=True,
    metric_for_best_model='f1',
    report_to='none',
    logging_steps=100,
    save_total_limit=2
)
```

The training parameters were all set up in the Hugging Face Trainer API, which has the **advantage** of simplifying the fine-tuning and logging processes while working well with PyTorch. As a **disadvantage,** however, it lacks the low-level optimization that creating a new model would have, and could require a significant amount of VRAM on larger datasets.

Evaluation Metrics and Computation

A custom compute_metrics() function was created to calculate the model's accuracy, precision, recall, and F1 score at each epoch using scikit-learn's precision_recall_fscore_support and accuracy_score utilities.

Figure 16

Metric computation function

```python
# metric function
def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='binary')
    acc = accuracy_score(labels, preds)
    return {
        "accuracy": acc,
        "f1": f1,
        "precision": precision,
        "recall": recall
    }
```

Training Process and Results

The model was fine-tuned for three epochs with automatic evaluation and checkpoint saving. Loss and metric trends were plotted across epochs to monitor for overfitting.

Figure 17

Epoch training results

[14469/14469 1:22:47, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.267100	0.229447	0.912277	0.913638	0.912148	0.915133
2	0.175100	0.265493	0.914869	0.917529	0.901678	0.933947
3	0.132600	0.336831	0.914247	0.915500	0.914846	0.916155

As shown above, the model achieved high performance early on in the training, surpassing the 90% accuracy goal required by the hypothesis. However, while training loss decreased steadily, the validation loss began to increase in further epochs, indicating the model was overfitting the training data. This shows that the model started to memorize training examples rather than improve generalization.

Both the validation accuracy (≈0.91) and F1 score (≈0.91) remained stable across epochs, confirming the model maintained strong predictive capabilities on unseen data.

Figure 18

Training and validation loss graph

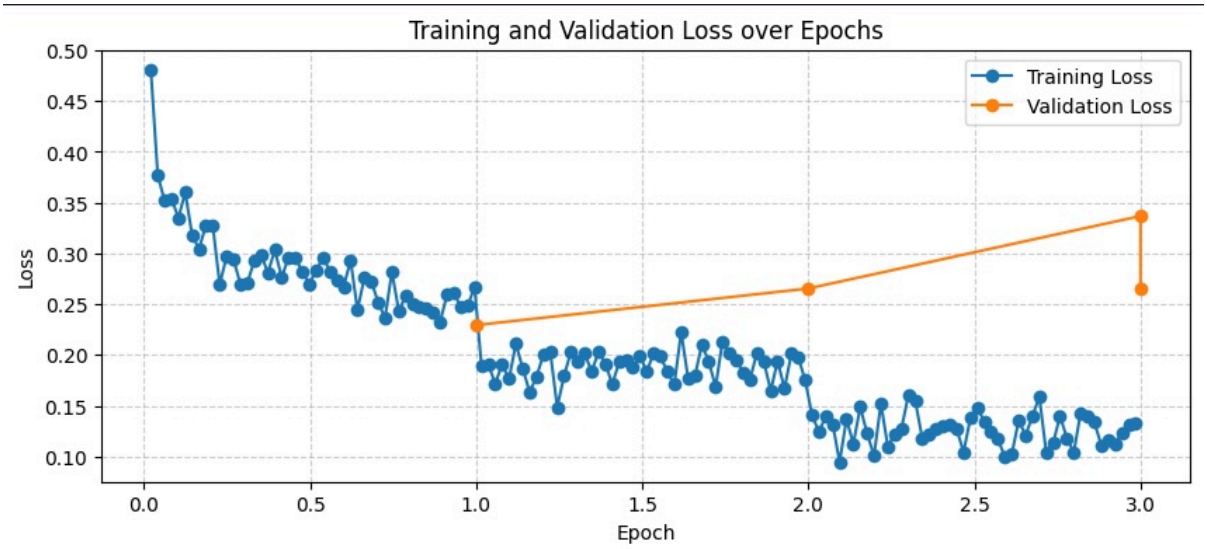
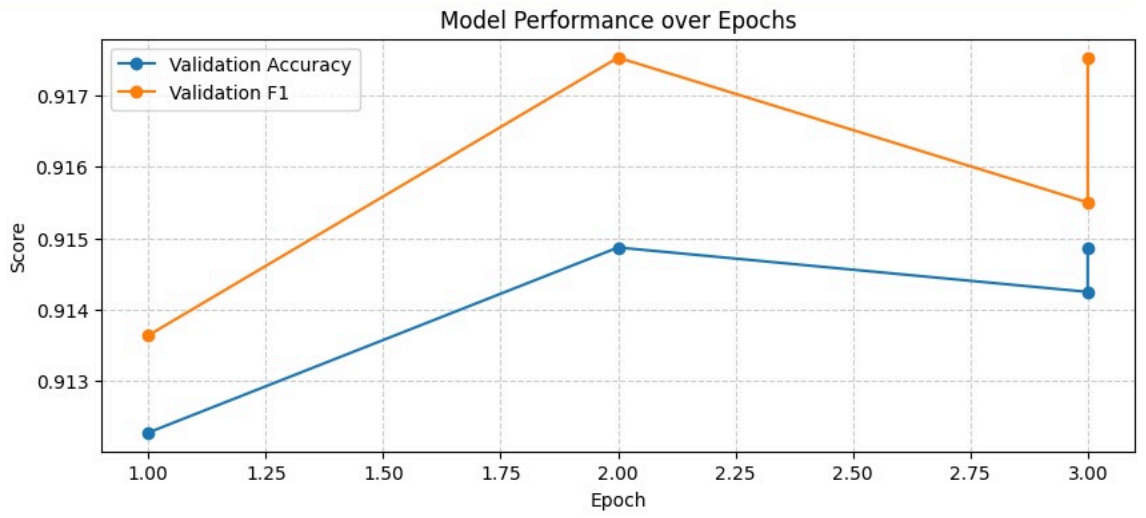


Figure 19

Model performance over epochs graph



An **advantage** of using a pre-trained model, like DistilBert, rather than creating a new model is that it enables high performance and accuracy with few epochs, reducing computational overhead. A **disadvantage**, however, is that extended training beyond the optimal epoch leads to overfitting and reduced validation performance.

Test Set Evaluation

After training, the best-performing model, based on the F1 score, was evaluated on the remaining 10% of the test dataset. The model achieved consistent results on this unseen data.

Figure 20

Testing set model evaluation

	precision	recall	f1-score	support
Not Recommended	0.93	0.89	0.91	4754
Recommended	0.90	0.93	0.91	4891
accuracy			0.91	9645
macro avg	0.91	0.91	0.91	9645
weighted avg	0.91	0.91	0.91	9645

A confusion matrix confirmed balanced classification performance across both sentiment classes, and the ROC curve (AUC ≈ 0.97) and Precision-Recall curve (AP ≈ 0.96) demonstrated strong discriminative power.

Figure 21

Confusion Matrix

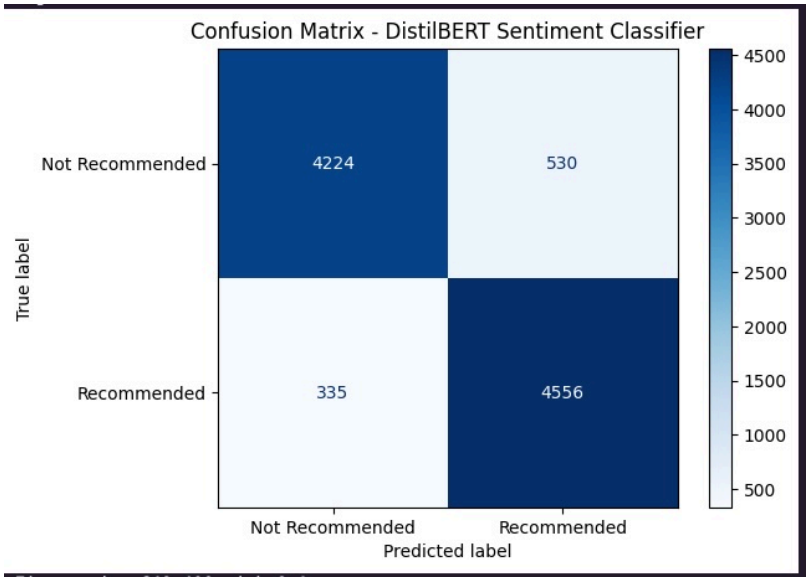


Figure 22

ROC Curve

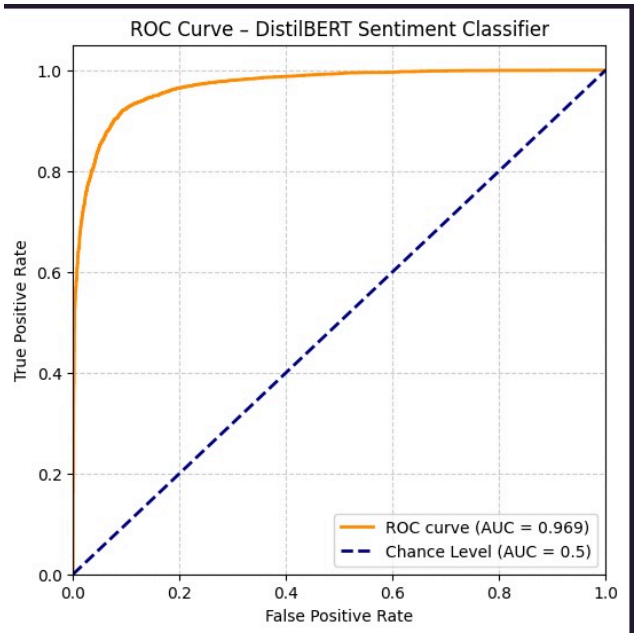
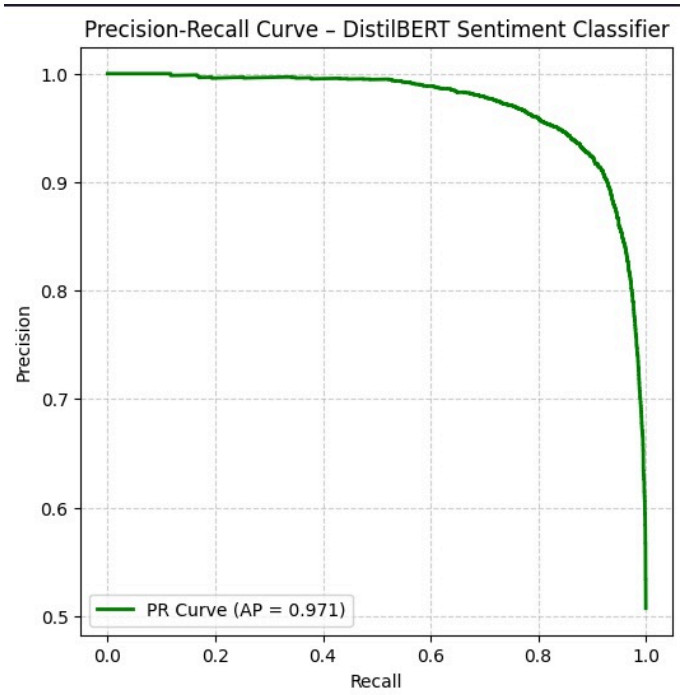


Figure 23

Precision-Recall curve



The ROC and PR curves have the **advantage** of providing a deeper understanding of model behavior across thresholds. They visually show what the metric table above outputs in

text, allowing technical stakeholders to get a sense of model performance quickly. However, the charts are less valuable to non-technical stakeholders and require explanation to be appropriately interpreted.

Error Inspection

A set of misclassified reviews was examined to better understand the model's limitations. The majority of false negatives involved sarcastic or humor-laden reviews that convey a positive tone using negative wording. False positives often included mixed reviews, with statements such as "The game was fun, but had a lot of performance problems". These cases show that a model can be proficient at detecting general polarity but struggles with linguistic features such as sarcasm, irony, or ambiguity.

Figure 24

Error Inspection of Testing Data

```
# checking a couple of mistakes
# convert to DataFrame for inspection
results_df = pd.DataFrame({
    "true_label": all_labels,
    "pred_label": all_preds,
    "prob_positive": all_probs,
    "review": df_test["review"].values,
    "genre": df_test["genres"].values
})

# find a few mistakes
errors = results_df[results_df.true_label != results_df.pred_label]
print(errors.sample(5)[["true_label","pred_label","prob_positive","review"]])
```

```
      true_label  pred_label  prob_positive  \
2094           0           1       0.763654
8017           0           1       0.625171
6982           1           0       0.047638
3314           0           1       0.904268
276            0           1       0.929408

                                                 review
2094  From all Survivor-Likes this is by far the one...
8017           17 HOURS AND STUCK ON EASIEST DIFFICULTY
6982  play this game without knowing anythingit is F...
3314  is a Point & Click "game" that requires you to...
276   It's a buggy mess but it's fun enough to play ...
```

Overall, the DistilBERT model achieved statistically significant accuracy and F1 performance in classifying Steam user review sentiment, meeting and slightly exceeding the 90% target threshold of the hypothesis. While some overfitting was seen after the first epoch, generalization remained strong, and the overall result demonstrated the effectiveness of transformer-based sentiment modeling on user-generated text. Due to this, the null hypothesis is rejected in favor of the alternative hypothesis, confirming that a transformer-based model can achieve ≥ 90% accuracy in classifying Steam user review sentiment.

E. Data Summary and Implications

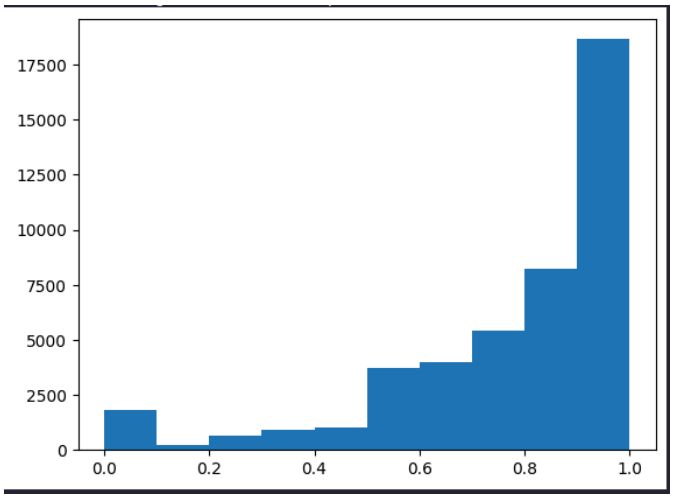
The fine-tuned DistilBERT model achieved over 90% accuracy and F1 score in classifying the polarity of Steam user reviews. With the model established and proven to deliver quality metrics, the next phase of analysis focused on exploring what the sentiment patterns reveal about player feedback and game design factors across genres.

Summary of Key Findings

In general, games from this time period on Steam were reviewed favorably. While Valve Inc.'s exact algorithm for aggregating review scores is proprietary, community-compiled data generally agrees that, for games with at least 10 reviews, ratings between 70% and 100% are considered positive (GearsRealm, 2024). This is reflected in the dataset as a whole.

Figure 25

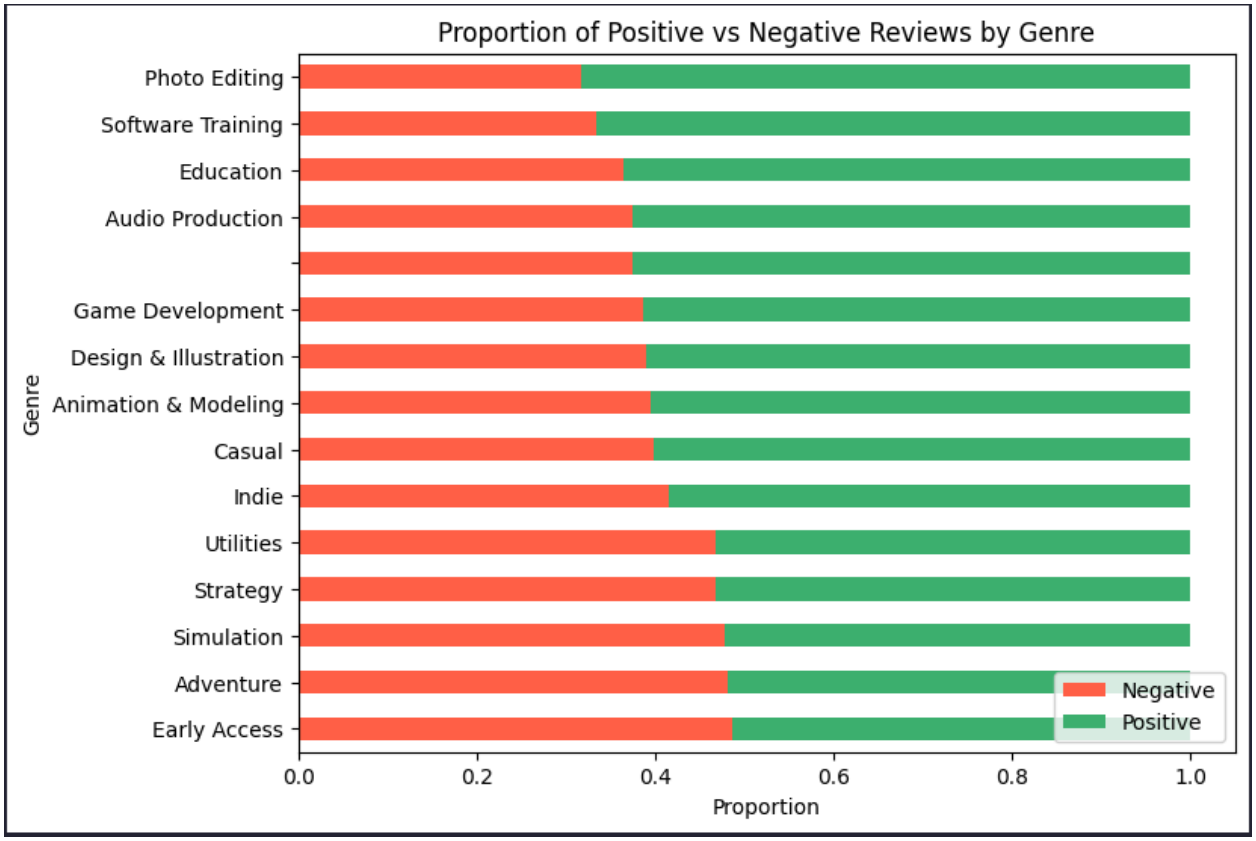
Overall user sentiment for all games from 2020 to 2024



This trend continues with separating games by genre. Surprisingly, games with more niche applications, such as Photo Editing or Education, were reviewed very positively. As for typical video game genres, Casual and Indie games overall received higher review scores (roughly ≥ 60%) than early access and adventure games, which had a more mixed distribution, hovering around the 50% mark..

Figure 26

Proportion of sentiment by genre



Positive reviews most frequently referenced words such as ‘story’, ‘pretty’, ‘amazing’, and ‘better’, while negative reviews concentrated on ‘don't’, ‘money’, ‘boring’, and ‘buy’.

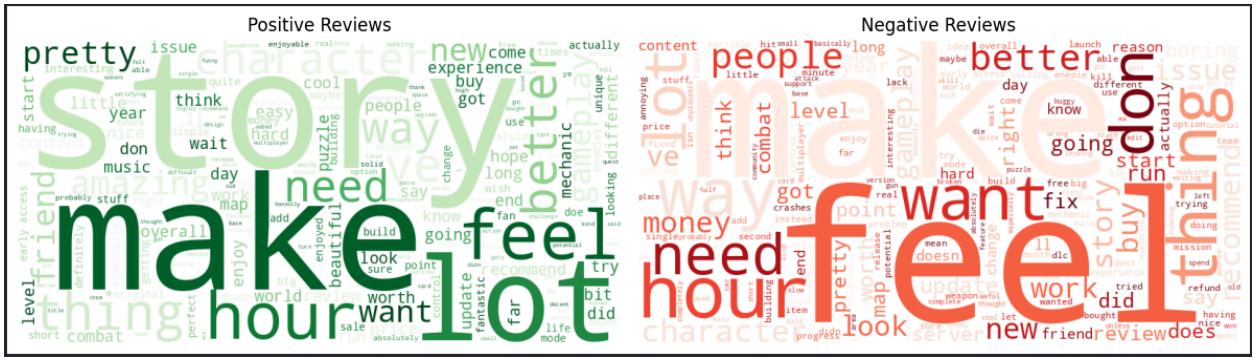
Figure 27

Top Positive and Negative keywords

```
Top Positive Keywords:
['story' 'amazing' 'recommend' 'better' 'hours' 'worth' 'pretty' 'lot'
 'nice' 'friends' 'make' 'gameplay' 'little' 'don' 'new']

Top Negative Keywords:
['don' 'buy' 'hours' 'money' 'boring' 'better' 'way' 'want' 'make'
 'recommend' 'people' 'feels' 'gameplay' 'dont' 'got']
```

Figure 28

Wordcloud of Positive and Negative sentiments



Interpretation and Business Implications

These results indicate that automated sentiment analysis can provide meaningful, data-driven insight for indie developers. By tracking sentiment and keywords, studios can make informed decisions about which features and aspects resonate most positively with players. For example, negative sentiment tends to cluster around topics such as money and bugs, suggesting that the overall time-value proposition and the game's operating performance are fundamental to players. On the other hand, there is strong positive sentiment towards stories and games with narrative depth, highlighting areas where developers can flex more creatively and produce content loved by players.

Limitations

One limitation of this study is that the dataset includes only reviews from 2020 through 2024, excluding both newer and older games from the analysis. Additionally, some linguistic patterns, such as sarcasm and irony, occasionally misled the model, as discussed in section D. These limitations suggest that, while the model overall captures sentiment effectively, certain aspects of human language remain difficult for automated systems to interpret.

Recommendations

It is recommended that indie development studios integrate automated sentiment monitoring tools into their post-release analytics workflows. Regularly analyzing post-release reviews can help developers understand what mistakes they made during development and correct those issues in future title releases, and identify features and decisions that resonated positively.

Future Research Directions

A few logical extensions to this research project could be:

- **Multilingual Expansion** - Integrating a multilingual sentiment analysis model to extend the analysis to non-English reviews and capture global player sentiment.
- **Thematic Clustering** - Applying unsupervised topic modeling, such as LDA or BERTopic, to uncover deeper themes within player feedback beyond positive/negative sentiment.

Conclusion

In summary, the model successfully validated the research hypothesis and demonstrated the utility of transformer-based NLP models in understanding user sentiment on large gaming platforms, such as Steam. The findings of this project underscore that data-driven sentiment analysis can provide meaningful insight to indie game studios, supporting both their design and community management decisions.

F. Sources

- Team, G. (2025, June 27). *How many games are released on Steam everyday?* - Games Learning Society. Games Learning Society. https://www.gameslearningsociety.org/wiki/how-many-games-are-released-on-steam-everyday/
- *Porter: The State of PC and Console Games in 2024*. (2024). Gdcvault.com. https://www.gdcvault.com/play/1034323/The-State-of-PC-and
- Abdelqader, H. (2025). “Steam Games Metadata and Player Reviews (2020–2024)”, Mendeley Data, V2, doi: 10.17632/jxy85cr3th.
- Steam Review Rating System Explained - *Overwhelmingly Positive, Negative and Mixed*. (2024, March 13). GearsRealm.com. https://gearsrealm.com/steam-review-rating-system-explained/
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., … Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (pp. 38–45). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-demos.6
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury Google, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., Devito, Z., Raison Nabla, M., Tejani, A., Chilamkurthy, S., Ai, Q., Steiner, B., & Facebook, L. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. https://papers.nips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

G. Acknowledgment

Portions of this report were developed with assistance from AI tools, including OpenAI's ChatGPT (GPT-5) and Google Gemini. These tools were used to support grammar and formatting revisions, outline refinement, and code-review suggestions. The author conducted all analyses, implemented the code, and interpreted the results.