Performance Comparison and Acceleration Study of Service Streamer-based Deep Learning and 3D Models on Local and Cloud

Jingyu Lu
20205767

Chuangxin Chu
20205753

I. Introduction

With the rise of the metaverse concept and natural language processing (NLP) models, the demand for computational resources has significantly increased. To address this, we developed a website that integrates both concepts and deployed it on cloud servers to validate their superiority.

In this project, we created three 3D virtual worlds where users can freely explore and share their comments and opinions. Additionally, we implemented two distinct NLP models: the BERT model, which fills in missing parts of user comments, and the Text-classification model, which determines whether user comments are positive or negative. This report aims to explore the potential of cloud servers in handling high computational demand applications and to provide technical references for future applications in the metaverse and NLP fields. Weiyun link to our video https://share.weiyun.com/nuZTCGIC

II. PERFORMANCE ANALYSIS

Based on our research, there are two performance bottlenecks in our program, computational resources and network transmission.

A. Computational Resources

1) Deep Learning Models: During the local training and testing process of NLP models, we discovered that GPU computational resources have become a performance bottleneck. Specifically, while training the Text-classification model using a vocabulary corpus of 251,639 words, we observed slow training speeds and excessively long iteration times. Despite GPU utilization consistently remaining high (close to 100%), it still failed to meet the computational demands.

Furthermore, deep learning models typically employ mini-batch processing to assemble samples together, leveraging the parallel computing capabilities of GPUs to accelerate computation. However, when deploying deep learning models to an online environment, user requests are usually discrete and singular. Utilizing traditional loop servers or multi-threaded servers in this scenario can lead to inefficient GPU resource usage and a linear increase in user wait times. In cases of high concurrent requests, this approach might even cause CUDA out-of-memory errors, potentially resulting in service crashes.

These findings highlight the necessity to ensure that servers have adequate computational capacity and scalability when deployed to production environments. Moreover, it is crucial to optimize performance bottlenecks in traditional methods to guarantee service stability and efficiency under high concurrency conditions.

2) 3D Virtual World: When importing a 3D model into Three.js to run a website on a cloud server, the following aspects need to be considered: 1 CPU: The loading, parsing, and rendering of 3D models all require CPU processing power. Complex models will consume more CPU resources. 2. GPU: Three.js uses WebGL for 3D rendering, which requires certain GPU performance support. Complex materials, textures, and special effects can impose a significant burden on GPUs. 3. Memory: 3D models and textures require memory space. Large models and high-resolution textures consume a significant amount of memory. Therefore, when choosing a cloud server configuration, it is necessary to comprehensively evaluate the CPU, GPU, memory, and bandwidth requirements based on the specific complexity, quantity, and scene design of the 3D model.

B. Network Transport

Bandwidth: Transferring resources such as 3D models and textures to the client requires a large amount network bandwidth support, especially for large models.

Having sufficient bandwidth on the server is crucial as it ensures that every user can quickly download the required 3D model files, avoiding delays and lag caused by slow internet speeds. Supporting more concurrent access, sufficient bandwidth can simultaneously support more users to access and download 3D models simultaneously, without affecting performance due to bandwidth saturation.

C. Computational Tasks

1) Deep Learning Models: Our project includes two natural language processing models: BERT and the Text-classification model. Given the high similarity in their training and inference processes, we can decompose both models into the same five primary computational tasks for analysis, as shown in Table II.

1) **Data Loading and Preprocessing**
In the data loading and preprocessing stage, we first

tokenize the text data and construct a vocabulary. Simultaneously, we load pre-trained word vectors (e.g., GloVe) to initialize the word embeddings. Next, the dataset is divided into training, validation, and test sets, and data iterators are created for batch processing.

2) **Constructing Model Inputs**
In the model input construction stage, batch data is retrieved from the data iterator and converted into the format required by the model. Subsequently, the data is transferred from the CPU to the GPU to accelerate subsequent computations. This step has moderate demands on both CPU and GPU.

3) **Model Training**
During the model training stage, forward propagation is executed to compute predictions and loss, followed by backpropagation to update model parameters. Gradient clipping is performed to prevent gradient explosion, and loss and accuracy are recorded throughout the training process. This step is highly dependent on GPU computation and requires substantial memory to store intermediate results and model parameters.

4) **Model Evaluation**
In the model evaluation stage, forward propagation is used to calculate predictions and loss on the validation set. Prediction accuracy is computed, and model parameters or optimization strategies are adjusted based on the evaluation results. This step also relies heavily on GPU computation and requires substantial memory to process validation data.

5) **Model Inference**
During the model inference stage, input test sentences are preprocessed and tokenized, then converted into the model input format. Forward propagation is performed to generate predictions, followed by post-processing to output the final prediction results. The inference stage requires high GPU utilization for forward propagation computation and moderate CPU and memory resources.

2) 3D Virtual World:

1) **Access to HTML, CSS, JS and Three.js backend logic**
CPU mainly used for parsing and executing HTML, CSS, JavaScript code. Bandwidth used to transfer HTML, CSS, JS files to the client. Memory used for caching and processing DOM structures as well as JavaScript runtime data.

2) **3D model scene loading:**
CPU used for loading and parsing 3D model data. GPU used for rendering 3D models, including geometry, materials, textures, etc. Memory used to store geometric, texture, and other data of 3D models. Bandwidth used to transfer resources such as 3D models and textures to the client.

3) **Player explore in the 3D scene:**
CPU used to calculate and update player's position, orientation, and other data. GPU used for real-time rendering of 3D scenes from the player's perspective. Memory used to store dynamically changing scene data, such as player positions, character models, etc. Bandwidth used for real-time transmission of player input data and scene data changes.

III. CLOUD SOLUTION

A. Cloud Server Provider Selection

We have chosen Tencent Cloud GPU servers as our cloud solution. Tencent Cloud, provided by Tencent. Known for its reliability, flexibility, and excellent performance, Tencent Cloud is highly regarded by enterprises and developers worldwide, positioning it as a leader in the cloud services market. We chose Ubuntu Server 22.04 LTS 64-bit as our operating system and opted for a billing method based on traffic with a budget of ¥25 per hour. Table III demonstrates a comparison of the specific configuration of the cloud server and the local machine.

(1) Tencent Cloud's GPU servers utilize a heterogeneous computing architecture optimized for complex computational tasks, enhancing the speed of deep learning and AI applications. (2) The platform features a user-friendly graphical interface that simplifies cloud resource management and monitoring. (3) With multiple data centers worldwide, Tencent Cloud ensures low-latency access and fast connectivity. (4) Compared to competitors, Tencent Cloud allows quick deployment of high-performance computing instances, offering a range of options to suit various needs, including general computing and memory optimization.

B. Efficient GPU Resource Management with ServiceStreamer Middleware

To effectively manage GPU resources and optimize computational efficiency, we have implemented ServiceStreamer middleware. This middleware aggregates discrete requests into batches, optimizing GPU utilization while simultaneously reducing system response times and resource consumption. ServiceStreamer mitigates bottlenecks in deep learning models that need to handle large volumes of concurrent requests, ensuring system stability and efficiency in the face of highly concurrent requests.

C. Distributed GPU Woker

To enhance overall performance and significantly increase GPU utilization, we have implemented a multi-threaded multi-GPU working mode alongside the use of ServiceStreamer. This mode supports the parallel execution of computational tasks across multiple GPU cards, with each GPU being simultaneously managed by multiple threads to fully harness each GPU's computational potential.

This mechanism allows each GPU card to independently execute different tasks or collaboratively handle different parts of the same task, thereby substantially increasing processing speed and efficiency. This parallel processing approach not only reduces the time required to complete tasks but also balances the load across various GPUs, preventing overload

TABLE I: Analysis of resource requirements for computing tasks - NLP Model

Step	CPU	GPU	Disk I/O	Storage - Memory	Storage - Disk
1. Data Loading and Preprocessing	High	Low	High	Medium	High
2. Constructing Model Inputs	Medium	Medium	Medium	High	Medium
3. Model Training	Medium	High	Low	High	Low
4. Model Evaluation	Medium	High	Low	High	Low
5. Model Inference	Medium	High	Low	High	Low

TABLE II: Analysis of resource requirements for computing tasks - 3D Virtual World

Step	CPU	GPU	Disk I/O	Storage - Memory	Storage - Disk	Bandwidth
1. Access to HTML, CSS, JS and Three.js backend logic	Medium	Low	Low	Low	Low	Low
2. 3D model scene loading	High	High	High	Medium	Medium	High
3. Player explore in the 3D scene	High	High	High	Medium	High	Medium

TABLE III: Local and Server Configuration

	CPU	Core	GPU	Memory	Bandwidth
Local	Intel(R) Core(TM) i5-1035G1	8-core	NVIDIA Gefore MX250	8GB	*
Tencent Cloud	Intel Xeon Cascade Lake(2.5GHz/-)	40-core	2 * NVIDIA T4	32GB	15Mbps

on any single unit and ensuring system stability and reliability. Furthermore, multi-threaded management facilitates more flexible task scheduling and resource allocation, enabling the system to respond more efficiently to high-concurrency computational demands.

D. Docker Deployment

During our collaborative development process, we encountered issues related to the inconsistent behavior of software across different development and production environments due to varying versions of torchtext (e.g., syntax differences in referencing the Field variable between version 0.9.0 and earlier versions) and incompatibilities between higher versions of PyTorch and lower versions of torchtext. Additionally, we aimed to achieve rapid deployment and more efficient resource utilization.

Docker and Docker Compose perfectly meet our development needs:

1) Docker containers encapsulate the application and all its dependencies, ensuring consistent operation across any environment.
2) Each Docker container can include its unique dependencies, avoiding conflicts between different applications.
3) Containers consume fewer resources than virtual machines, allowing more service instances to run on the same hardware.
4) Docker Compose manages inter-service networks, storage, and dependencies through a single YAML file, simplifying the configuration of complex applications.

By utilizing Docker and Docker Compose, we can more effectively manage our development and production environments, ensuring consistency, optimizing resource utilization, and simplifying the deployment process.

Listing 1: *DockerFile*

```
FROM python:3.9
WORKDIR /app
COPY . /app
RUN pip install --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

Listing 2: *docker-compose.yaml*

```
services:
  cloud-computing:
    build:
      dockerfile: Dockerfile
    container_name: cloud-computing-flask
    ports:
      - "5000:5000"
    depends_on:
      - mysql
    networks:
      - cloud-computing
  mysql:
    image: mysql:8
    container_name: mysql
    command: --default-authentication-plugin=mysql_native_password
    ports:
      - "3306:3306"
    environment:
      - MYSQL_DATABASE=comments
      - MYSQL_ROOT_PASSWORD=123456
    volumes:
      - mysql_datadir:/var/lib/mysql
    networks:
      - cloud-computing
# Nginx is managed by the Ubuntu system
volumes:
  mysql_datadir:
networks:
  - cloud-computing:
```

IV. RESULTS AND ANALYSIS

A. Performance Comparison: Model Training

In order to compare the performance of models training in different environments, we chose the Text-classification model for testing. Since this model uses a larger corpus in training, it is able to show the performance difference more clearly. The following table IV shows the experimental results in different environments, as the significantly higher hardware capabilities of cloud servers compared to local machines, it is evident that the training speed of the model on cloud servers is substantially faster than on local machines.

TABLE IV: Performance Metrics in Different Environments

	Runtime	GPU Utilization	CPU Utilization
Local environment	545.4s	95.20%	17.88%
Tecent Cloud	61.2s	81.7%	0.82%

B. Performance Comparison: Model Inference

To compare the performance of the model inference process in different environments, we choose to use the BERT model as an example, and apply wrk - a HTTP benchmarking tool to conduct performance tests, the following are the experimental results in different environments.

TABLE V: Attributes Description

Attribute	Description
Original	ServiceStreamer acceleration is not used
Streamer	Use only ServiceStreamer acceleration
Multi-Streamer	Acceleration with ServiceStreamer and distributed GPU woker

1) Local environment: Table VI shows that the model optimized with ServiceStreamer middleware significantly outperforms the unoptimized model in all key performance metrics. The optimized service not only has lower average and maximum latency, but also handles more requests and transfers larger amounts of data. In addition, their response times are more consistent, demonstrating greater processing power and efficiency. This illustrates that the ServiceStreamer middleware is effective in optimizing the performance of the deep learning model, making it outstanding in terms of speed, throughput and stability, and is an effective solution to improve the performance of the model.

TABLE VI: Local Environment Performance Metrics

Metrics	Original	Streamer
Avg Latency	3.12s	1.12s
Requests/sec	38.37	109.97
Transfer/sec	6.33KB	18.15KB
Total requests	768	2210
Stdev	1.20s	112.34ms
Max Latency	6.18s	1.81s

2) Tencent Cloud: Since the cloud server provides 2 GPUs, this supports us to apply distributed techniques to further optimize the inference process of the model, where we use 8 threads equally distributed to both GPUs.

Table VII presents a performance comparison of three different configurations of the BERT model. Although the use of distributed technology in the Multi-Streamer model introduces communication overhead, resulting in a slightly higher average latency compared to the Streamer model, it demonstrates significantly superior performance in terms of requests per second and data transfer rates. Consequently, the Multi-Streamer model is considered the optimal configuration for overall performance due to its high throughput and relatively low latency. By integrating the ServiceStreamer middleware with distributed technology, the Multi-Streamer model significantly enhances processing capability and efficiency.

TABLE VII: Tencent Cloud Metrics

Metrics	Original	Streamer	Multi-Streamer
Avg Latency	1.94s	227.50ms	358.92ms
Requests/sec	63.08	559.49	1412.37
Transfer/sec	10.53KB	93.43KB	156.13KB
Total requests	1263	11200	28318
Stdev	180.88ms	15.07ms	130.21ms
Max Latency	2.85s	376.40ms	1.36s

C. Performance Comparison: 3D Virtual World

CPU performance: The better the CPU performance of a cloud server, the faster it can process geometric data, materials, and other information of 3D models, thereby accelerating model loading and rendering speed and improving user experience.

GPU performance: Cloud servers have more powerful GPUs that can perform graphics processing tasks required for 3D rendering faster, such as texture mapping, lighting calculation, etc., thereby improving overall rendering efficiency.

Bandwidth: Cloud servers have higher network bandwidth, which enables faster transfer of 3D model data to the user end, shortens model loading time, and reduces network transmission bottlenecks. (Due to the inability of local testing to reflect the speed of bandwidth, we used the server of our UCD final year project. Bandwidth: 1Mbps)

Memory: Cloud servers are equipped with more graphics memory, which can accommodate larger 3D model data, avoiding frequent swapping in and out due to insufficient graphics memory, and reducing lag during loading.

TABLE VIII: Cloud Server Optimization Result

Value	Local Environment	Tencent Cloud
Average FPS	56 fps	162 fps
Load time	15s	4s

D. Synthetic Discussion

We successfully leveraged cloud servers to address the computational resource bottlenecks of our deep learning models. Furthermore, by integrating ServiceStreamer middleware and distributed technologies, we enhanced service performance on the foundation of the cloud servers' superior hardware, resolving potential wastage of GPU computational resources and CUDA out-of-memory errors under high concurrency.

However, the high cost of GPU cloud servers highlights a sharp cost issue. We need to find a balance between effectiveness and expense, and explore further technological methods to optimize service performance, rather than overly relying on the superior hardware capabilities of the cloud servers themselves.

As for 3D graphics section, with the significant improvement of CPU performance, GPU performance, bandwidth, and memory RAM through cloud server. The improvement results during the loading model stage can be measured by the loading response time, while the improvement results during the player exploration model stage can be measured by the average frame rate. Both have significant improvements.

V. Conclusion

In conclusion, our cloud computing project integrates advanced concepts from the metaverse and natural language processing models. The project identified performance bottlenecks in computing resources and network transmission, and successfully overcame these issues by leveraging the superior hardware capabilities of cloud servers, coupled with the implementation of ServiceStreamer middleware and distributed technologies. This accomplishment further demonstrates the superiority of cloud computing over local execution in terms of efficiency and scalability.