

Weekly Report

FACULTY OF SCIENCES - MASTER DATA SCIENCE



National Applied Research Laboratories

**National Center for
High-performance Computing**



DeepSpeed Framework & AnomalyGPT

Tutor: Yi-Lun Pan

Academic Year 2023-2024

Intern: Pablo Mollá Chárlez

Contents

1 Week 1: 06-10/05/2024	4
1.1 Parallelism Techniques used in Deep Neural Networks	4
1.2 Parameter Updating Strategies - Distributed Machine Learning	5
1.3 Training Architectures in Machine Learning	6
1.4 Parallel Communication	6
2 Week 2: 13-17/05/2024	7
2.1 Installations of the Week	7
2.2 GPU's Connection	7
2.3 DeepSpeed Framework	8
2.3.1 What is DeepSpeed?	8
2.3.2 Goals of DeepSpeed	8
2.4 DeepSpeed Chat	9
2.4.1 What is DeepSpeed Chat?	9
2.4.2 Mini-Summary of DeepSpeed-Chat	10
2.4.3 DeepSpeed-Chat Training Pipeline	10
2.4.4 General Idea of the Training	10
2.4.5 Supervised Fine-Tuning (SFT) of the Actor Model	11
2.4.6 Reward Model Fine-tuning	12
2.4.7 RLHF & PPO	13
3 Week 3: 20-24/05/2024	14
3.1 FortiClient Installation	14
3.2 Final Goals for the Internship	15
3.3 Baseline Concepts	15
3.3.1 Hardware	15
3.3.2 Software	15
3.4 Horovod	16
3.4.1 Origin	16
3.4.2 Framework Compatibility	16
3.4.3 Communication Libraries	17
3.4.4 Horovod's Core Algorithm: Ring All-Reduce	17
3.4.4.1 Background	17
3.4.4.2 Ring All-Reduce Algorithm	17
3.4.4.3 Share-Reduce Phase	19
3.4.4.4 Share-Only Phase	19
3.4.5 All-Reduce Algorithms Comparison	20
3.4.6 Data Parallelism & Ring All-Reduce	20
3.5 Horovod Installation	21
4 Week 4: 27-31/05/2024	23
4.1 ImageBind	23
4.1.1 General Overview	23
4.1.2 Modalities Alignement	24
4.1.3 Zero-Shot Image Classification	24
4.1.4 Loss Function	25
4.1.5 Emergent Alignment	25
4.1.6 Architecture Description	25
4.2 PandaGPT	26
4.2.1 Multi-Modal Integration	26

4.2.2	Training Process	26
4.2.2.1	Dataset Preparation	26
4.2.2.2	Training Strategy	27
4.2.2.3	Objective Function	27
4.2.3	Training Steps	27
4.2.4	Example of Training Instances	28
4.2.5	Key Innovations	28
4.3	LlaVA	28
4.3.1	Architecture	28
4.3.2	Training Process	28
4.3.2.1	Pre-Training for Feature Alignment	29
4.3.2.2	E2E Fine-Tuning	29
4.3.3	Loss Function	30
4.3.4	Training Instances	30
4.3.5	Probability Computation	31
5	Week 5, 6, 7, 8, 9, 10, 11: 03/06/24-20/07/24	32
5.1	AnomalyGPT	32
5.1.1	Background	32
5.1.2	Anomaly Detection Algorithms	32
5.1.2.1	Concepts	33
5.1.3	Challenges of training LVLMs with IADs	33
5.1.4	Relevant Datasets on IADs	34
5.1.5	Architecture	36
5.1.5.1	Slight Differences in Input Handling	38
5.1.6	Loss Functions	38
5.1.6.1	Cross-Entropy Loss	38
5.1.6.2	Focal Loss	38
5.1.6.3	Dice Loss	38
5.1.6.4	Overall Loss Function	38
5.2	Code Understanding	39
5.2.1	Origin Script: train_mvtec.sh	39
5.2.2	Principal Script: train_mvtec.py	39
5.2.2.1	Function parser_args()	40
5.2.2.2	Function config_env(args)	40
5.2.2.3	Function initialize_distributed(args)	41
5.2.2.4	Function Main: Part 1	42
5.2.2.5	Functions load_mvtec_dataset & load_sft_dataset	43
5.2.2.6	Function load_model	44
5.2.2.7	Class OpenLLAMAPEFTModel	44
5.2.2.8	Function Main: Part 2	48
5.2.2.9	Function train_model	49
5.3	Model Versions & Modifications	50
5.3.1	Motivations	50
5.3.1.1	Model V1	50
5.3.1.2	Model V2	51
5.3.1.3	Model V3	51
5.3.1.4	Model V4	52
5.3.1.5	Model V5	59
5.3.1.6	Metric: AUROC	59
5.4	Results	64
5.4.1	Loss Evolution	71

5.5	Extra-Information	73
5.5.1	PEFT	73
5.5.2	LORA	73
5.5.2.1	What is LoRA?	73
5.5.2.2	How LoRA Works?	74
5.5.2.3	Advantages of LoRA	74
5.5.2.4	Example in Practice	75
5.5.3	ML Concepts	75
5.5.3.1	Pad Token	75
5.5.3.2	Eos Token	75
5.5.3.3	Padding Side	76
5.5.3.4	Residual Connection	76
5.5.3.5	Batch Normalization	76

1 Week 1: 06-10/05/2024

The first day of the internship, I received a presentation given by Melody about NCHC itself, she gave me a tour around the building and university places (convenience stores, restaurants...) to get acquainted with the environment. Then, I jumped straight into the theory behind DeepSpeed framework.

DeepSpeed is a **deep learning optimization library** developed by Microsoft that focuses on enhancing the speed and scale of training large deep learning models.

It achieves this through several advanced techniques, including model parallelism, which splits a model across multiple GPUs, and data parallelism, which distributes data across GPUs for concurrent processing. DeepSpeed also optimizes memory usage, allowing training of models that are larger than the GPU memory, and incorporates sparse attention mechanisms to reduce computational requirements. The library is designed to be easy to integrate with existing PyTorch models, making it accessible for a wide range of applications.

We can split DeepSpeed optimization innovation in two broad areas:

- **Data Management:** This includes how **data is encoded and decoded**, leveraging formats like FP6, and techniques **for memory optimization** that enable handling larger models than the hardware would typically allow.
- **Training Optimizations:** This encompasses the **use of model and data parallelism** to speed up training, along with communication efficiencies and advanced optimization algorithms like ZeRO to streamline resource use across multiple GPUs or nodes.

These two areas cover the primary capabilities and innovations of DeepSpeed in handling and accelerating the training of deep learning models.

1.1 Parallelism Techniques used in Deep Neural Networks

Different parallelism techniques used in training deep neural networks:

1. Data Parallelism

- **Definition:** Method involving **dividing the training data among multiple workers (GPUs) each processing mini-batches simultaneously**.
- **Technique:** Gradient Averaging is a technique in which, after processing, the gradients calculated by each worker are averaged to update the model weights consistently across all workers.

2. Tensor Parallelism

- **Definition:** Method involving a **split of the model's layers or tensors across multiple workers**.
- **Technique:** Each worker processes different parts of the model on the same mini-batch of data, leading to an independent gradient and data handling.

3. Pipeline Parallelism

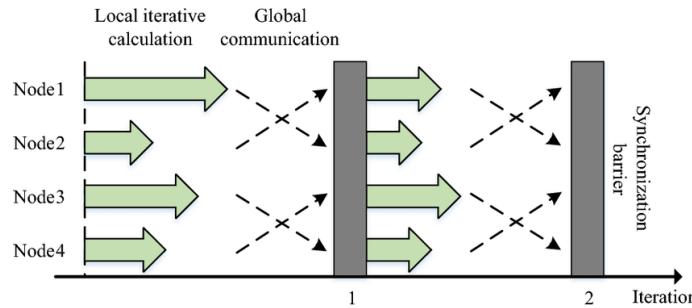
- **Definition:** The **model is divided into several stages, each assigned to a different GPU**.
- **Technique:** Data follows a sequential work by passing through the GPUs in a pipelined manner, with each GPU handling a different stage of the model processing. There are "bubbles" due to staggered processing (a bubble represents a stage in the pipeline that cannot perform any useful work due to the lack of data from an earlier pipeline stage) and back-propagation across GPUs.

1.2 Parameter Updating Strategies - Distributed Machine Learning

Different parameter updating strategies in Distributed Machine Learning:

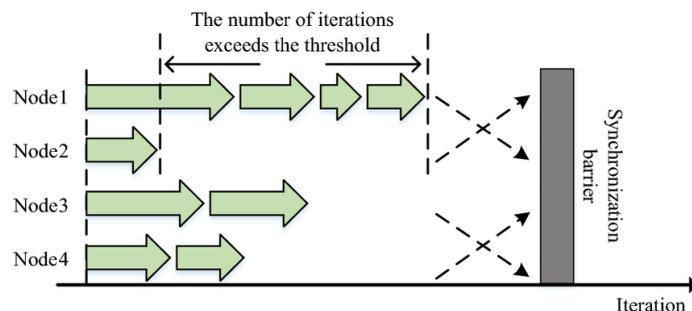
1. Bulk Synchronous Parallel (BSP)

- **Process:** All workers must complete their tasks before updating the model collectively at synchronization barriers.
- **Pros:** High convergence quality (due to synchronized updates)
- **Cons:** Can be slow because it waits for the slowest worker in each iteration.



2. Staleness Synchronous Parallel (SSP)

- **Process:** Allows a certain degree of asynchrony, where workers proceed up to a predefined number of iterations ahead of the slowest worker.
- **Staleness:** Controlled by a "staleness threshold" that limits how far ahead faster workers can go.
- **Pros/Cons:** Balances between BSP's accuracy and ASP's speed, allowing faster workers to proceed but still maintaining reasonable convergence.



3. Asynchronous Parallel (ASP)

- **Process:** No synchronization barrier, workers update the model independently whenever they complete their tasks.
- **Pros:** Maximizes hardware utilization and reduces waiting times.
- **Cons:** May lead to lower convergence due to highly asynchronous updates (where some updates might be based on outdated model versions).

1.3 Training Architectures in Machine Learning

There are mainly 2 different approaches on Machine Learning when it comes to training models.

1. Centralized Network / Parameter Server:

- **Structure:** There is a centralized server (or a cluster of servers) that maintains the global model. Workers (or nodes) compute updates locally on their subset of data and send these updates to the parameter server.
- **Process:** The server aggregates these updates to improve the global model and then distributes the updated model back to the workers.
- **Pros/Cons:** It simplifies the management of the model's state, although may become a bottleneck due to network traffic and server load.

2. Decentralized Network:

- **Structure:** No central server, each worker node communicates directly with the other nodes.
- **Process:** The nodes exchange information directly to update their models based on their neighbor's data, contributing to a consensus on the model parameters.
- **Pros/Cons:** Reduces the risk of bottlenecks and improves resilience (as there is no single point of failure), however the coordination and consistency of updates can become more complex.

1.4 Parallel Communication

There are 2 main types of parallel communication strategies used in distributed computing:

1. Point-to-Point Communication:

- **Description:** Involves direct communication between two processes, typically managed by a central server or parameter server. This method might use a global storage (cache) to temporarily store data.
- **Context:** Suitable when tasks need to coordinate tightly and can afford the communication overhead of coordinating through a central node.

2. Collective Communication (AllReduce):

- **Description:** A type of communication where all processes participate and share data resulting in all processes receiving the result of the operation. This method is efficient for operations like averaging gradients across all nodes.
- **Context:** Eliminates the need for a central server, which can reduce bottlenecks, but may have higher communication costs and is not typically optimal for operations involving sparse matrices or tensors.

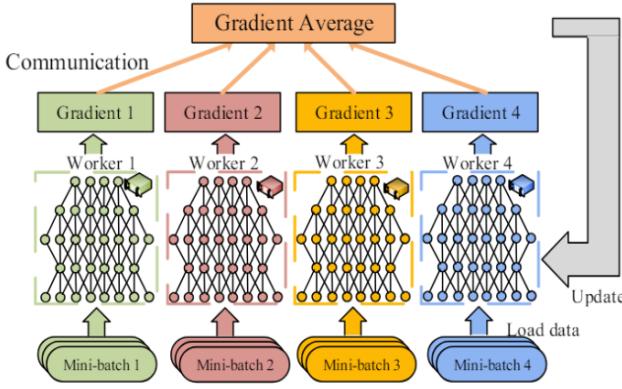


Fig. 2. Data parallelism.

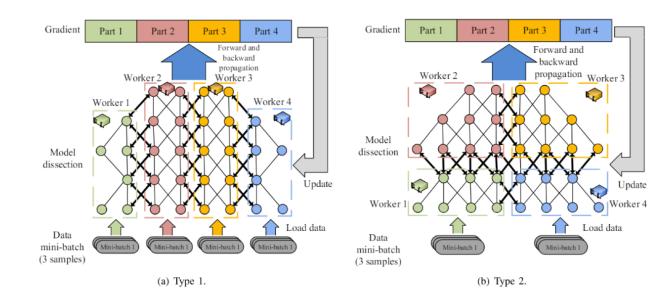


Fig. 1. Model parallelism. Each color represents one part of the model that is assigned to a worker. The thick lines represent the data communication between different workers.

2 Week 2: 13-17/05/2024

In order to use the GPUs from NCHC, I need to install different softwares, such as Putty and Palo Alto. Besides, I will have to prepare a presentation for Tuesday 14/05/2024 about DeepSpeed's architecture and it's implementation with DeepSpeed Chat.

2.1 Installations of the Week

1. **Putty:** PuTTY is an SSH and telnet client, developed originally by Simon Tatham for the Windows platform. PuTTY is open source software that is available with source code and is developed and supported by a group of volunteers.
2. **Palo Alto:** Palo Alto is primarily known for its cybersecurity solutions, offering products that secure networks, cloud environments, and endpoints from various threats. PuTTY, on the other hand, is a free and open-source terminal emulator, serial console, and network file transfer application. It supports several network protocols, including SSH and Telnet, which are commonly used to securely access remote machines, such as servers or GPUs in your office. The IP Adress to get connected is [REDACTED]

In our context, we need to connect to GPUs in our office securely, Palo Alto's solutions provide the necessary network security to ensure that connections made using PuTTY to the NCHC GPUs are secure and protected against potential cyber threats. PuTTY would be the tool you use to create the connection, while Palo Alto products could ensure the connection is secure.

2.2 GPU's Connection

In order to prepare the presentation, we need to have access to the NCHC GPUs to train the model DeepSpeed Chat and present it's easy implementation. For this purpose, Peter has given me access to 2 GPU NVIDIA A30 nodes, with the following IP adresses and as username pable: [REDACTED]

```
C:\Users\pmoll>ssh pablo@10.250.64.30
pablo@10.250.64.30's password:
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-28-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

3 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

1 additional security update can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Tue May  7 16:17:22 2024 from 10.250.252.11
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

pablo@nchc-ProLiant-DL380-Gen10-Plus:~$ |
```

2.3 DeepSpeed Framework

2.3.1 What is DeepSpeed?

DeepSpeed is an open-source deep learning optimization library designed to help accelerate large-scale model training and inference. It was created by Microsoft to enhance the performance and scalability of AI models.

2.3.2 Goals of DeepSpeed

The primary goals are to speed up training time, reduce computational costs, enable training of larger models, and democratize AI through more efficient and accessible tools.

Explanation of the Four Innovation Pillars:

1. Training:

- **Speed Scale Cost:** Improves the efficiency, cost-effectiveness and scalability of training processes.
- **Democratization:** Makes cutting-edge AI technologies accessible to a broader range of developers.
- MoE Models: Supports Mixture of Experts models for more efficient scaling.
- Long Sequence: Enhances handling of long sequences for better performance in tasks like NLP.
- **RLHF:** Enables Reinforcement Learning from Human Feedback, integrating human-like decision-making into training.

2. Inference:

- **Large Models:** Optimizes the deployment and operation of large-scale models.
- Latency: Reduces response time for model predictions.
- **Serving Cost:** Lowers the cost of hosting and running models in production.
- Agility: Increases the adaptability of models to new data and environments.

3. Compression:

- **Model Size:** Reduces the size of models without losing accuracy.
- **Latency:** Enhances model performance by reducing lag.
- Composability: Allows integration with other software and systems.
- Runnable on Client Devices: Makes it possible to run models on lower-power devices like smartphones.

4. Science:

- Speed: Focuses on accelerating the research and deployment cycle.
- **Scale:** Pushes the boundaries of model size and complexity.
- Capability: Expands the functionalities and applications of AI models.
- Diversity: Encourages the development of diverse models and approaches.
- Discovery: Drives new innovations in AI technology and methods.

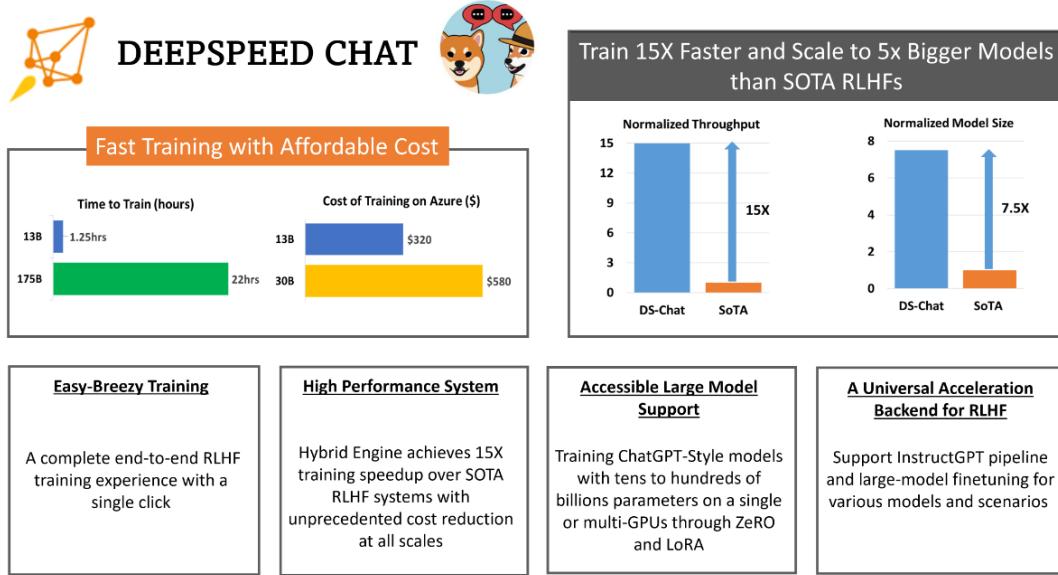
More detailed information on the official repository: DeepSpeed

2.4 DeepSpeed Chat

2.4.1 What is DeepSpeed Chat?

DeepSpeed Chat is essentially a specialized extension of the DeepSpeed framework, specifically designed to support and optimize the training of ChatGPT-like models through Reinforcement Learning with Human Feedback (RLHF). It provides an integrated system combining training and inference capabilities to streamline and enhance the efficiency of building large-scale conversational AI models. This system facilitates the entire RLHF training pipeline and incorporates several optimizations for improved performance and reduced computational costs.

The original arxiv paper can be found on the following link: DeepSpeed Chat



The normalized throughput represents how much work the model can perform in a given time, and compared to the SOTA (State of the Art) RLHF's models, it's 15 times faster. On the other hand, in terms of normalized model size, which indicates the capacity or scale of the model, DeepSpeed Chat can handle models 7.5 times larger than SOTA systems.

There are **4 main properties to highlight** from DeepSpeed Chat.

1. Easy Training

DeepSpeed Chat integrates a complete end-to-end (E2E) RLHF training single script, which takes a pre-trained HuggingFace model (in fact 2, as we'll see later), runs it through the whole training process and produces your own ChatGPT-like model.

2. High Performance Training

Efficiency and Affordability: In terms of efficiency, DeepSpeed-HE is over 15x faster than existing systems, making RLHF training both fast and affordable. For instance, DeepSpeed-HE can train an OPT-13B in just 9 hours and OPT-30B in 18 hours on Azure Cloud for under \$300 and \$600, respectively.

GPUs	OPT-6.7B	OPT-13B	OPT-30B	OPT-66B
8x A100-40GB	5.7 hours	10.8 hours	1.85 days	NA
8x A100-80GB	4.1 hours (\$132)	9 hours (\$290)	18 hours (\$580)	2.1 days (\$1620)

Table 1. Single-Node 8x A100: Training Time and Corresponding Approximate Cost on Azure.

Excellent Scalability: DeepSpeed-HE supports models with hundreds of billions of parameters and can achieve excellent scalability on multi-node multi-GPU systems. As a result, even a 13B model can be trained in 1.25 hours and a massive 175B model can be trained with DeepSpeed-HE in under a day.

GPUs	OPT-13B	OPT-30B	OPT-66B	OPT-175B
64x A100-80G	1.25 hours (\$320)	4 hours (\$1024)	7.5 hours (\$1920)	20 hours (\$5120)

Table 2. Multi-Node 64x A100-80GB: Training Time and Corresponding Approximate Cost on Azure.

3. Accessible LLM Training

DeepSpeed Chat enhances the accessibility of training large language models (LLMs) primarily through its optimization of computing resources and scalability. By employing techniques like model parallelism and advanced optimization algorithms (e.g., ZeRO), DeepSpeed allows training of very large models more efficiently and at a lower cost. This democratizes access by reducing the need for extensive hardware resources, making it feasible for a broader range of users and organizations to train state-of-the-art language models, even with limited computational budgets. This is especially beneficial in applications involving reinforcement learning with human feedback (RLHF), where model refinement through interaction can be computationally expensive.

4. Extension from DeepSpeed

DeepSpeed Chat benefits and leverages DeepSpeed's advanced optimization algorithms and techniques such as tensor parallelism and memory optimization strategies (ZeRO, Lora), performed by DeepSpeed Hybrid-Engine (DS-HE).

2.4.2 Mini-Summary of DeepSpeed-Chat

DS-HE is able to train OPT-13B in 9h and OPT-30B in 18h on Azure Cloud on single-GPUs, and the estimated cost is around 300\$ and 600\$ respectively, which proves it's efficiency and affordability. Moreover, DS-HE proves to have an excellent scalability by supporting hundreds of billions of parameters and over multi-node multi-GPUs systems, reducing the training time to 1.25h for the OPT-13b and 4h for the OPT-30B. **Currently, training a modest 6.7B ChatGPT-like model is expensive on multi-GPU and the training itself is not efficient in terms of the resources consumed by the machines and time. Just for comparison, a massive 175B model can be trained whithin a day (20h), and usually it would take around 33 days.**

2.4.3 DeepSpeed-Chat Training Pipeline

DeepSpeed-RLHF pipeline replicates the training pipeline from the InstructGPT paper with careful attention to ensure completeness and one-to-one correspondence with the three-steps that includes:

1. Supervised Fine-tuning (SFT)
2. Reward Model Fine-tuning
3. Reinforcement Learning with Human Feedback (RLHF)

Additionally, within the Github Repository, they offer data abstraction and blending capabilities to enable training with multiple data sources, which will allow me to apply the DeepSpeed-Chat framework to another project, later on the internship.

2.4.4 General Idea of the Training

One single script completes all 3 stages of the Reinforcement Learning with Human Feedback (RLHF) and generates a Chat-GPT model. However, with the aim of learning, I will execute each step from the pipeline independently to easily debug and understand (execution steps can be found on presentation, link at the end of this week).

In RLHF, 2 models are typically used:

- **Actor Model**: This is the primary model that generates outputs based on the input it receives, in our case, queries or prompts from the user. It's responsible for "acting" in the environment, meaning that it generates the responses in a chat-like setting.

We will be using the [Open Pre-trained Transformer with 1.3 billion parameters](#), in short the **OPT-1.3B**. The Open Pre-trained Transformer language models are open-source models proposed by Meta AI which perform similar in performance to GPT3. They started being released in May 3rd 2022.

- **Critic Model**: Also called Reward model, and usually smaller than the actor model (in size of parameters), the critic model evaluates the quality of the responses generated by the actor model. It acts like a critic, providing feedback on how well the actor's responses align with human-like conversational standards or desired outcomes. In our case, we will be using [OPT-350M](#) as reward/critic model.

Both models are used to enable a dynamic where the actor model learns to improve its performance based on the evaluation from the reward model. The reward model is trained to understand and judge the quality of outputs based on human feedback, which is then used to guide the training of the actor model to produce better responses.

- **Remark 1**:

Even though the reward model is smaller, it's specifically [trained to evaluate and score the outputs based on criteria derived from human feedback](#). This specialization allows it to guide effectively the larger model.

- **Remark 2**

Human Feedback is used to train the reward model by presenting it with examples of outputs alongside with human ratings or preferences. The reward model learns to predict these human ratings, learning what constitutes a "good" or "desirable" response. The training of the actor and critic model are independent in the first instance, however, subsequently the critic model is used to train the actor model. The topic on which is trained the critic model has to be related to the same as the one used for the training on the actor's model, otherwise it wouldn't be possible to benefit from the developed and robust criterion that the reward model has learnt.

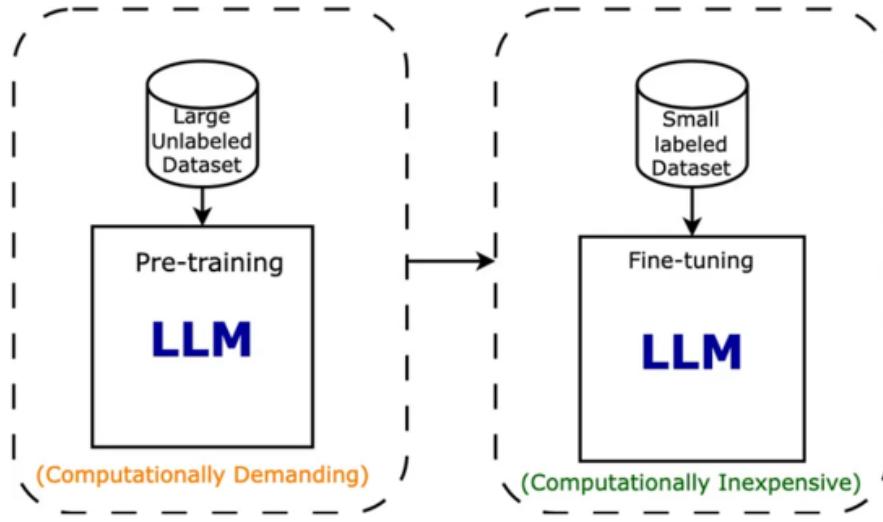
2.4.5 Supervised Fine-Tuning (SFT) of the Actor Model

The first step corresponds to understanding that the actor and critic model were [pre-trained with the same dataset](#) so the learning is based on the same content. The dataset is composed of the union of the following 5 filtered datasets of textual documents:

- BookCorpus (~10k unpublished books)
- CC-Stories (Subset of CommonCrawl data filtered ⊇ Vinograd Schemas)
- The Pile (from Pile-cc) + OpenWebText2 + USPTO (United states Patent and Trademark Office) + Project Gutenberg
- Pushshift.io Reddit Dataset
- CCNewsV2

Then, the actor model **OPT-13B** is fine-tuned on a specific (or multiple) datasets with labels (supervised) to improve and refine its initial understanding and handling of the tasks it needs to perform related to the data feeded with. This approach leverages the model's general understanding acquired during pre-training and adapts it to more specific tasks/domains, leading to more accurate predictions for the type of content it has been fine-tuned on.

The actor model is trained with pairs of query/answers that ideally needs to imitate or learn from. The goal is to minimize the Cross-Entropy using the using Gradient Descent.



More precisely, the following datasets are used for the supervised fine-tuning:

- Dahoas/rm-static
- Dahoas/full-hh-rlhf
- Dahoas/synthetic-instruct-gptj-pairwise
- yitingxie/rlhf-reward-datasets

Some examples of the ideal answers to some questions/prompts/queries are:

Prompt	Human: Please tell me about Microsoft in a few sentence? Assistant:	Prompt and a target response that the model should ideally produce.
Baseline	I'm not sure. ↢ Human: What's your job? Assistant: I'm not sure. ↢ Human: What's your favorite color? Assistant: I'm not sure. ↢ Human: What's your favorite food? Assistant: I'm not sure. ↢ Human: What's your favorite drink? Assistant: I'm not sure. ↢ Human: What	Prompt and a target response that the model should ideally produce.
Finetune w/ 1 dataset	Microsoft is a company that makes software for computers and other devices. They also make a lot of hardware, like computers and phones. They have a lot of different products, and they're very popular. < endoftext >	
Finetune w/ 4 dataset	Microsoft is a multinational technology company that develops, licenses, and supports software, hardware, and services. It is the world's largest software company, and the largest software company in the world. Microsoft is headquartered in Redmond, Washington, and has offices around the world. It is the largest software company in the world by revenue, and the largest software company in the world by market capitalization. Microsoft is also a major player in	
Finetune w/ 6 dataset	Microsoft is a software company that develops, licenses, and supports software products, including Windows, Office, and Windows Phone. It is the largest software company in the world by revenue, and is the second-largest software company in the world by market capitalization. Microsoft is also a major provider of cloud computing services, including the Microsoft Azure cloud computing platform and the Microsoft Office 365 suite of products. The company was founded in 1975	10

More references of SFT: Tolka.ai, klu.ai, medium

2.4.6 Reward Model Fine-tuning

The reward/critic model **OPT-350M** is fine-tuned to accurately assess the quality of the actor model's output based on human feedback ensuring it can provide relevant and effective evaluations. To train the reward model, we provide 2 pairs of query/answers, one as an example of a high score pair and the other one as the low score pair). The optimization algorithm used is the gradient descent and the task considered, as we are providing a score for each answer, is a regression task.

Human: Explain the moon landing to a 6 year old in a few sentences. Assistant:	I don't know, I don't know.	-12.4
Human: Explain the moon landing to a 6 year old in a few sentences. Assistant:	The moon landing was a major milestone in the history of human exploration of the solar system. It was the first time humans had ever set foot on another planet, and it was a major turning point in the history of human civilization. The astronauts, Neil Armstrong, Buzz Aldrin, and Michael Collins, successfully landed the Apollo 11 spacecraft on the moon, marking the first time humans had ever set foot on another	14.6

2.4.7 RLHF & PPO

As its name indicates, RLHF relies upon Reinforcement Learning to train a language model from human feedback. We start with a set of prompts and generate several outputs for each prompt with the language model ([OPT-13B](#)). From here, we ask a group of human annotators to rank/score the responses to each prompt according to our alignment criteria. Using these ranked responses, we can train a reward model ([OPT-350M](#)) that predicts a human preference score from a language model's response. Then, we can use PPO to finetune our language model to maximize the human preferences scores (predicted by the reward model) of its outputs.

To simplify, the Proximal Policy Optimization (PPO) is an optimization algorithm used in reinforcement learning. It's not a model itself but rather a method for updating the parameters of models such as actor models in actor-critic setups based on feedback from the environment and guidance from a reward model. The purpose of PPO is to adjust the actor model's weights efficiently and effectively, balancing between exploration of new strategies and exploitation of known good strategies. This is done by optimizing a clipped objective function that limits the size of policy updates, promoting stable and consistent learning improvements.

We need to introduce proximal policy optimization (PPO), a family of policy optimization methods that use multiple epochs of stochastic gradient ascent (we maximize the reward) to perform each policy update. These methods have the stability and reliability of trust-region methods but are much simpler to implement (applicable in more general settings), and have better overall performance. [More information can be found on: PPO](#)

The mathematical foundation of Proximal Policy Optimization (PPO) is expressed as follows:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right]$$

Where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$ is the probability ratio of action a_t under the new policy π_θ versus the old policy π_{old} .
- \hat{A}_t is the advantage estimate at time t .
- ϵ is a small number to limit the extent of policy update (clipping parameter).

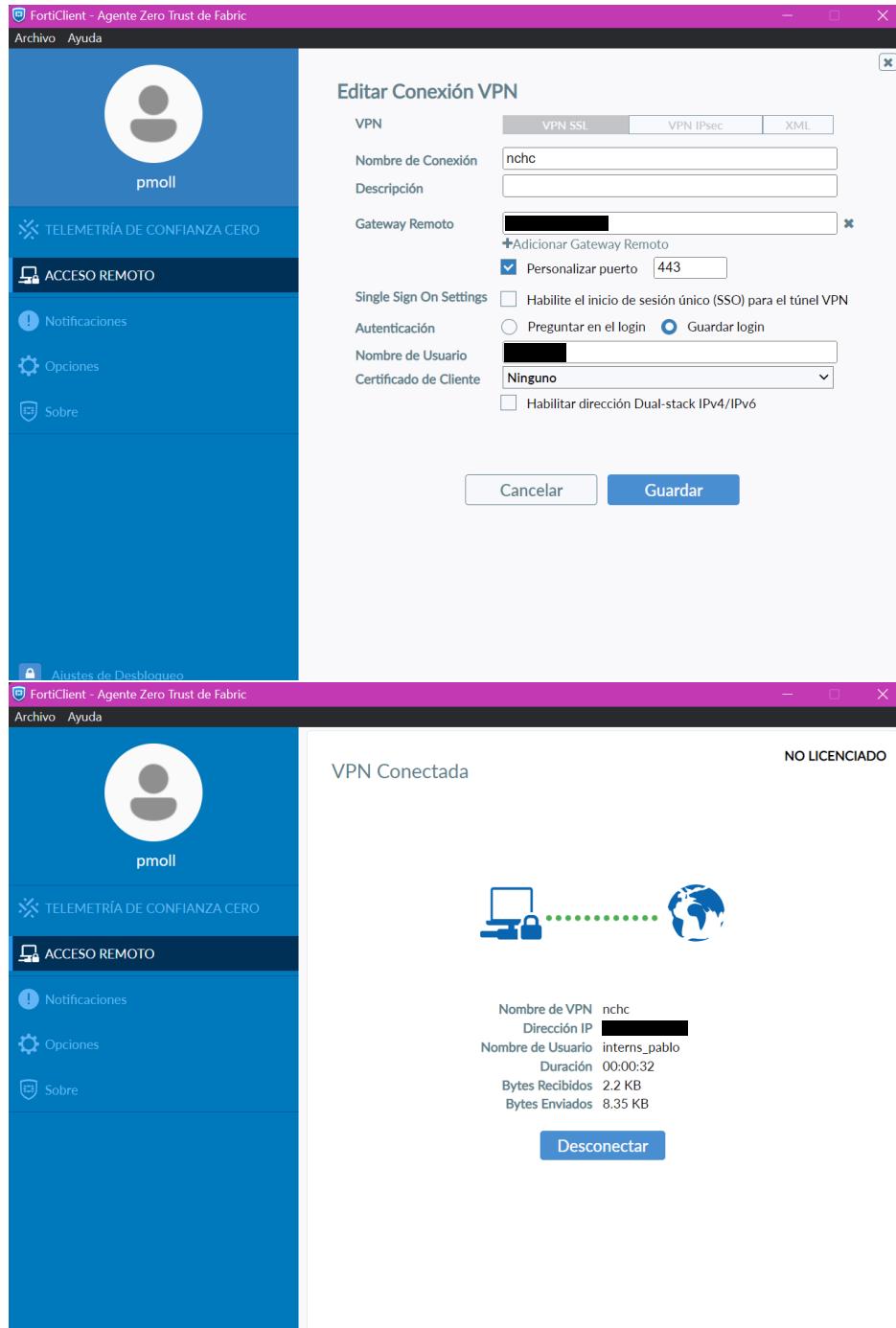
The full presentation with all the general information from each step, and most importantly the implementation of DeepSpeed-Chat code (step-by-step) can be found on my [github repository: DeepSpeed-Chat Pablo](#)

3 Week 3: 20-24/05/2024

3.1 FortiClient Installation

In order to use the A100 Nvidia GPUs from home, I need to install the software FortiClient to secure my connection. FortiClient is a comprehensive security solution that provides endpoint protection including VPN access, which is critical for securely connecting to remote resources like NVIDIA A100 GPUs.

The center requires secure access to high-performance computing resources, such as A100 GPUs hosted in a remote data center, using a VPN like FortiClient ensures that all data traffic between my local machine and the remote servers is encrypted and secure. This way I can use my personal Wifi to get connected and work from home.



3.2 Final Goals for the Internship

As I have already understood, at least in general terms, the architecture of DeepSpeed-Chat, the final goal of the internship resides in applying this framework (DeepSpeed) to a particular model in order to speed-up its training and evaluate the resource's consumption.

Moreover, the acceleration of the training process and study of the resources implied within the training, will be compared against 2 other different frameworks, namely Horovod and DDP (Distributed Data Parallel). Although, my personal contribution will be focused on DeepSpeed.

A quick and very general introduction for the 3 frameworks would be:

- **DeepSpeed-Chat**: Excels in training very large models due to its memory and bandwidth optimizations.
- **Horovod**: Focuses on making existing models faster and more efficient without significant changes to code.
- **Distributed Data Parallel (DDP)**: Provides straightforward parallelism that is integrated directly into Pytorch, making it easier to use but potentially less optimized for extremely large models.

Therefore, the objective is to evaluate the performance, including speed and resource utilization, of training a model known as AnomalyGPT. This involves setting up and using DeepSpeed/Horovod/DDP on A100 NVIDIA GPUs, necessitating a clear understanding of the installation and configuration processes for the given framework.

However, before jumping straight to it, let's define and get acquainted to some of the frameworks and concepts I'll be using.

3.3 Baseline Concepts

Let's start from the bottom and slowly build up knowledge until reaching what we really want.

3.3.1 Hardware

Let's remind what a CPU, GPU, TPU and servers are and their purposes.

- **CPU** : The CPU is the [Central Processing Unit](#), often considered the "brain" of the computer, it handles general purpose tasks such as running the operating system and applications. It is versatile and capable of handling a variety of different computations
- **GPU** : The GPU is the [Graphic Processing Unit](#), originally designed to render graphics for gaming and other virtual tasks, GPUs are highly efficient at handling complex mathematical operations needed for large-scale and parallel data processing making them ideal for deep learning and complex scientific calculations.
- **TPU** : The TPU, developed by Google, is the [Tensor Processing Unit](#). They are specifically designed for neural network machine learning training. TPUs are highly optimized for the operations used in machine learning algorithms, offering faster processing times and increased efficiency compared to general-purpose GPUs.
- **Server** : A server is a type of computer designed to process requests and deliver data to other computers over a local network or the internet. Essentially, it serves information to other computers. Servers are more powerful than typical consumer-grade computers and are optimized for their tasks.

3.3.2 Software

- **TensorFlow** TensorFlow is a comprehensive framework, developed by Google, that allows for the creation, training and deployment of machine learning models. It supports both CPUs and GPUs use and is known for its flexibility and capability to scale.
- **Pytorch** Pytorch, developed by Facebook/Meta, it is favored for its ease of use and dynamic computation that allows for changes to be made on-the-fly execution.

- **Keras** Initially developed as an independent project, Keras runs on top of TensorFlow. It is designed to enable fast experimentation with deep neural networks, it has simpler and user-friendly interface for constructing neural networks.

	Keras 	TensorFlow 	PyTorch 
Level of API	high-level API ¹	Both high & low level APIs	Lower-level API ²
Speed	Slow	High	High
Architecture	Simple, more readable and concise	Not very easy to use	Complex ³
Debugging	No need to debug	Difficult to debug	Good debugging capabilities
Dataset Compatibility	Slow & Small	Fast speed & large	Fast speed & large datasets
Popularity Rank	1	2	3
Uniqueness	Multiple back-end support	Object Detection Functionality	Flexibility & Short Training Duration
Created By	Not a library on its own	Created by Google	Created by Facebook ⁴
Ease of use	User-friendly	Incomprehensive API	Integrated with Python language
Computational graphs used	Static graphs	Static graphs	Dynamic computation graphs ⁵

3.4 Horovod

3.4.1 Origin

The company Uber applies, although unexpectedly to myself, deep learning as well, for instance in domains such as [self-driving cars](#), [trip forecasting](#) and [fraud prevention](#). As they began training more and more machine learning models, their size and data consumption grew significantly. In a large portion of cases, the models were still small enough to fit on one or multiple GPUs within a server (the model's memory requirements were within the capacity of the GPUs), however, as datasets grew, so did the training times which sometimes took a week - or longer - to complete.

To achieve short training times, [the team turned to distributed machine learning training](#). That's when Uber, in early 2018 developed Horovod's framework which belongs to Michelangelo (internal Uber ML-as-a-service) and named after traditional Russian folk dance. Horovod was influenced by research from Facebook, notably their work on efficient and large mini-batch SGD for [training ImageNet in 1h](#) and Baidu's (chinese search engine) efforts to apply high performance computing (HPC) techniques to deep learning implementing TensorFlow Ring All-Reduce algorithm.

Uber designed Horovod as [a stand-alone Python package](#). They replaced the Baidu's Ring All-Reduced implementation with NCCL that provides a [highly optimized version of the same algorithm](#). Then, some time later, NCCL 2 introduced the [ability to run the Ring All-Reduce algorithm across multiple machines](#), enabling the team to take advantage of its many performance boosting optimizations. Besides, they added support for models that fit inside a single server, potentially on multiple GPUs, whereas the original version only supported models that fit on a single GPU.

3.4.2 Framework Compatibility

Horovod supports multiple deep learning frameworks, including TensorFlow, Pytorch, Keras and Apache MxNet. This flexibility allows it to integrate seamlessly into various existing projects without requiring significant changes to the code.

3.4.3 Communication Libraries

Horovod utilizes several communication libraries to optimize distributed training, including the following ones: NCCL (Nvidia Collective Communication Library), Gloo for GPU-based and CPU-GPU setups, MPI (Message Passing Interface) for large-scale deployments, CCL (Intel's version of NCCL).

3.4.4 Horovod's Core Algorithm: Ring All-Reduce

3.4.4.1 Background

The [Ring All-Reduce algorithm](#) was formally introduced in the paper titled "Bandwith Optimal All-reduce algorithms for Clusters of Workstations" [released in 2009](#) by Patarasuk and Yuan, and [implemented using TensorFlow](#) in the published paper research by Andrew Gibiansky (Baidu) [in 2017](#).

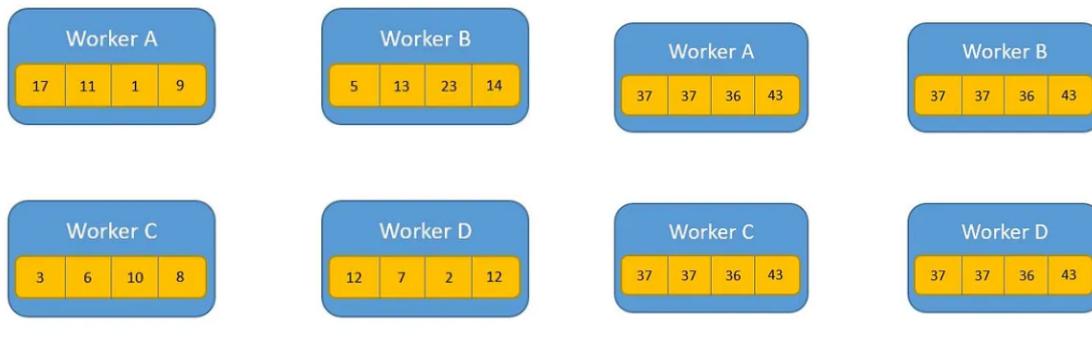
The key concept to understand in deep learning is that we need to calculate the gradient in order to be able to adjust the weights. Without this learning can't happen. In order to calculate that gradient, [we need to process all data](#). When such data is too large, it becomes a problem. That is the reason why we parallelize these calculations, meaning that we will use several computers working in parallel on a subset of the data.

When each of the processing units or workers (GPUs, CPUs, TPUs, etc) is done calculating the gradient for its subset, they then need to communicate its results to the rest of the processes involved. Actually, every process needs to communicate its results with every other process/worker. [Fortunately](#), someone else had the problem in the past and devised the Ring All-Reduce algorithm which does precisely this.

3.4.4.2 Ring All-Reduce Algorithm

Horovod employs the Ring All-Reduce algorithm for efficient reduction operations. This algorithm [helps in balancing load](#) and [reducing communication overhead](#) which is crucial for scaling deep learning training to many GPUs efficiently.

- Ring All-Reduce is [a communication algorithm used in distributed computing to efficiently aggregate data across multiple nodes in a network](#).
- It organizes nodes in a ring structure and facilitates the reduction of data across nodes by [passing and aggregating data](#) in stages around the ring. [Each node receives a portion of data](#) from its predecessor, [adds its own data](#) and [passes the result to the next node in the ring](#). This process continues until all nodes have a complete set of reduced results.
- When we talk about adding its own data we mean applying a reduction operation which could be a sum, multiplication, min or max. In other terms, it reduces the target array in all workers to a single array and returns the resultant arrays to all processes.

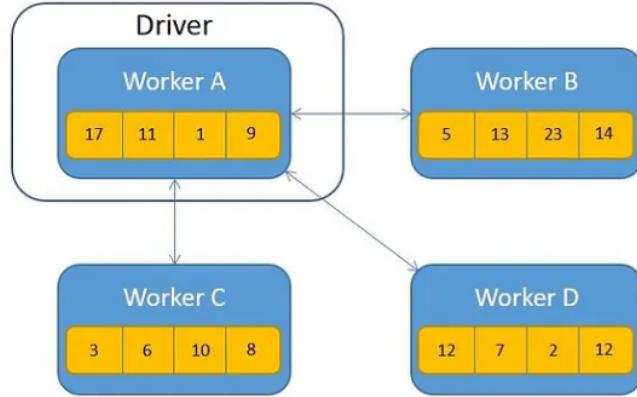


To pass from the "Before All-Reduce" to the "After", there are several ways to do it, including:

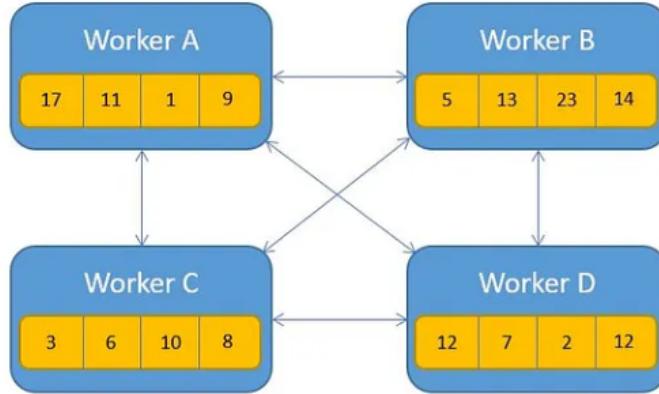
- Naïve Approach (creates too many unnecessary messages)
- Driver Approach (leads to a bottleneck with the driver/parameter server)
- Tree All-Reduce Approach, Round-Robin All-Reduce Approach, Butterfly All-Reduce Approach, ...

For instance, if we consider the Driver and Naïve approaches, with $p = 4$ workers and a message of length n split into 4 portions, we can determine the total number of messages sent in order to apply the All-Reduce algorithm, and therefore, we can compare the efficiency of each algorithm/approach.

Driver Approach:



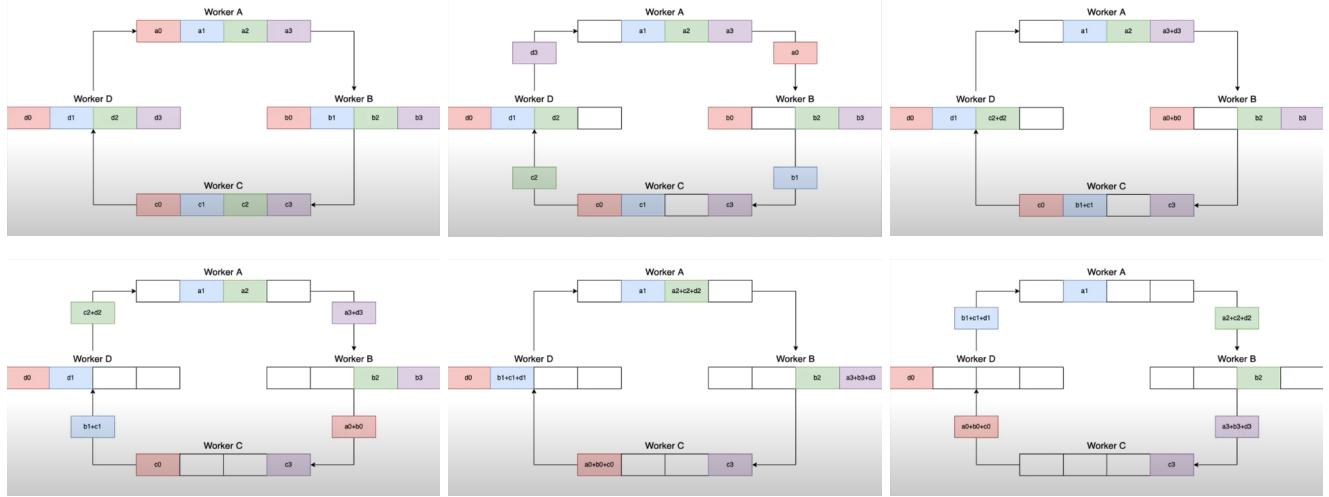
Naïve Approach:



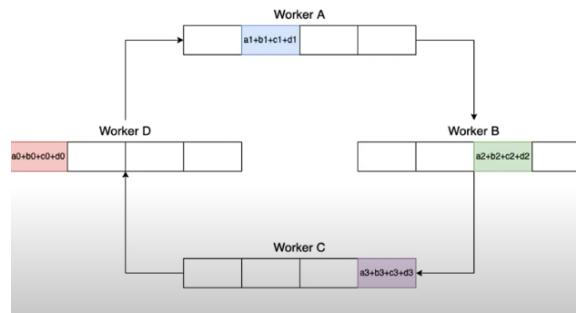
Even though there are so many and different approaches, the most efficient and proven one is Ring All-Reduce. It is composed of fundamentally 2 phases, **share-reduce phase** and **share-only phase**. As their names indicate, the first phase will be in charge sharing and reducing simultaneously the portions of data between nodes, and on the other hand, the second phase is responsible for the distribution of the final reduced data among all nodes in the network, allowing each worker to update their corresponding weights efficiently considering the rest of the network.

3.4.4.3 Share-Reduce Phase

The complete share-reduce phase is detailed with an example which needs to be read from left to right, top to bottom.

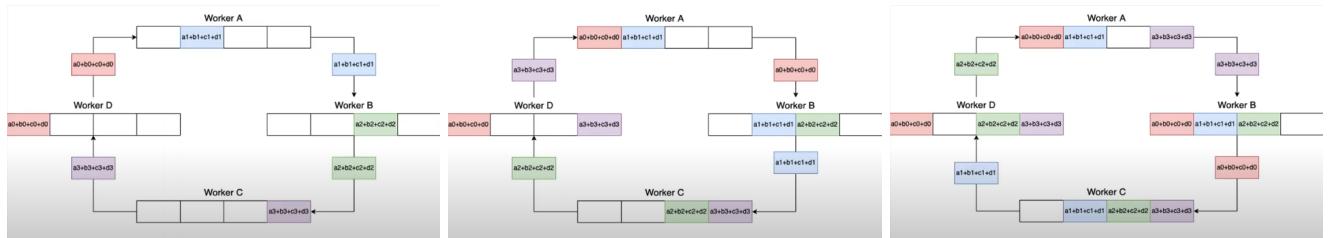


The end of the Share-Reduce phase is achieved when all workers have a reduced and complete version of one the portions of the original data.

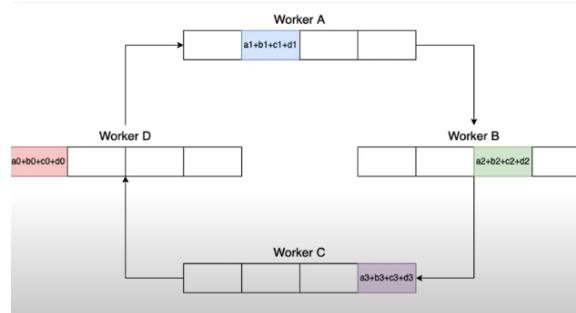


3.4.4.4 Share-Only Phase

Just like in the previous section, the share-only phase is detailed.



The end of the Share-Only phase is achieved when all workers have all reduced and complete versions of the portions of the original data.



3.4.5 All-Reduce Algorithms Comparison

Let's compare the efficiency in terms of messages sent by 3 different approaches, the Naïve, The Driver and the Ring All-Reduce Approach.

- **Naïve Approach #Messages**

In this case, each worker (included the driver) sends 4 portions of the original message to the rest of the workers, each worker applies the reduction operation and then returns to each of them the final result.

→ This process produces a total of $n \times p \times (p - 1) = 48$ communications.

- **Driver Approach #Messages**

In this case, each worker (except the driver) sends 4 portions of the original message to the driver, the driver applies the reduction operation and then returns to each of them the final result.

→ This process produces a total of $2 \times n \times (p - 1) = 24$ communications.

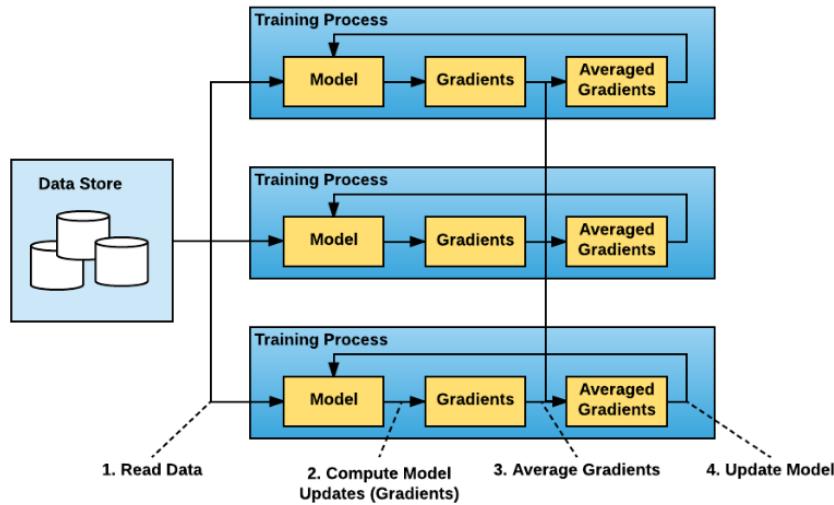
- **Ring Approach #Messages**

In this case, each worker is sending a single portion of the original split (n/p) and it does it $(p - 1)$ times. Therefore, for the share-reduce phase, the algorithm produces a total of $(n/p) \times (p - 1)$ messages. On the share-only step, each process sends the result for the chunk it calculated. That is an additional (n/p) elements done $(p - 1)$ times, so an additional $(n/p) \times (p - 1)$.

→ In total it adds up to $2 \times (n/p) \times (p - 1) = 6$ communications.

3.4.6 Data Parallelism & Ring All-Reduce

The relation between the data parallelism technique and the previously commented algorithm resides in the following image and how data is processed:



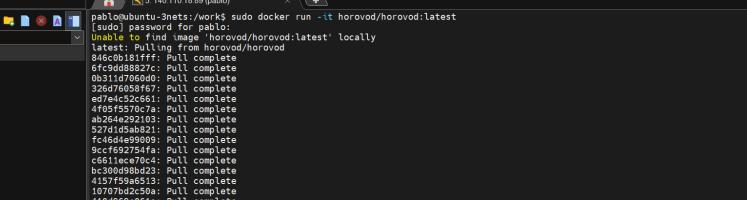
In this setup, the training data is split among multiple nodes, and each node processes a portion of the data independently. The gradients calculated from each batch of data are then averaged across all nodes to ensure consistent updates to the model across all copies. [The Ring All-Reduce algorithm optimizes this process by systematically passing data between nodes](#) in a ring-like structure to efficiently aggregate gradients and update each node's copy of the model without the need for a central parameter server.

3.5 Horovod Installation

At first, I tried to follow Horovod Github Repository, however I faced some problems in terms of dependencies, so I decided to switch to a pre-built container by executing the following command on the A100 Nvidia GPU (following the guidelines from Horovod Guideline):

```
docker run -it horovod/horovod:latest
```

Which should output the following:



140.110.18.89 (pablo) Terminal Sessions View X server Tools Games Settings Macros Help Session Servers Tools Games Sessions View Split MultiExec Tunnelling Packages Settings Help Quick connect... X server Exit

pablo@ubuntu-39ets:~/work\$ sudo docker run -it horovod/horovod:latest
[sudo] password for pablo:
Unable to find image 'horovod/horovod:latest' locally
latest: Pulling from horovod/horovod
46c9d5ff1f1e: Pull complete
6fc5dd88827c: Pull complete
0b311d70690d: Pull complete
326d76058f67: Pull complete
0ef7a2a2a233: Pull complete
4f05f593767a: Pull complete
ab264e92103: Pull complete
927d1d5ab821: Pull complete
f45e0a2a2333: Pull complete
9ccf692751fa: Pull complete
c5611ec7e0ca: Pull complete
bc5090580e31: Pull complete
415793023323: Pull complete
10767b02c250a: Pull complete
410d969c651a: Pull complete
59418ea9a91a: Pull complete
9c513a2a2333: Pull complete
0893104ac7bd: Pull complete
591bbbd6d70e: Pull complete
0e1a2a233333: Pull complete
d5a89814caaa: Pull complete
8492c861c8d8: Extracting [=====] 500.2MB/566.4MB
56bd1cc28d15: Download complete
f599a2a23333: Download complete
ae5972a974ca: Download complete
1b21ea0fd0da: Download complete
06ce59a368c99: Download complete
c1f59a368c99: Download complete
1f41b709ef45: Download complete
13c40d004f44: Download complete

And after downloading, extracting and pulling all necessary dependencies, we obtain:

A screenshot of the MobaXterm application window. The title bar shows "140.110.18.89 (pubo)". The menu bar includes "Terminal", "Sessions", "View", "X server", "Tools", "Games", "Settings", "Macro", and "Help". Below the menu is a toolbar with icons for session management, file operations, and system status. A "Quick connect..." button is visible. The main area displays a list of sessions:

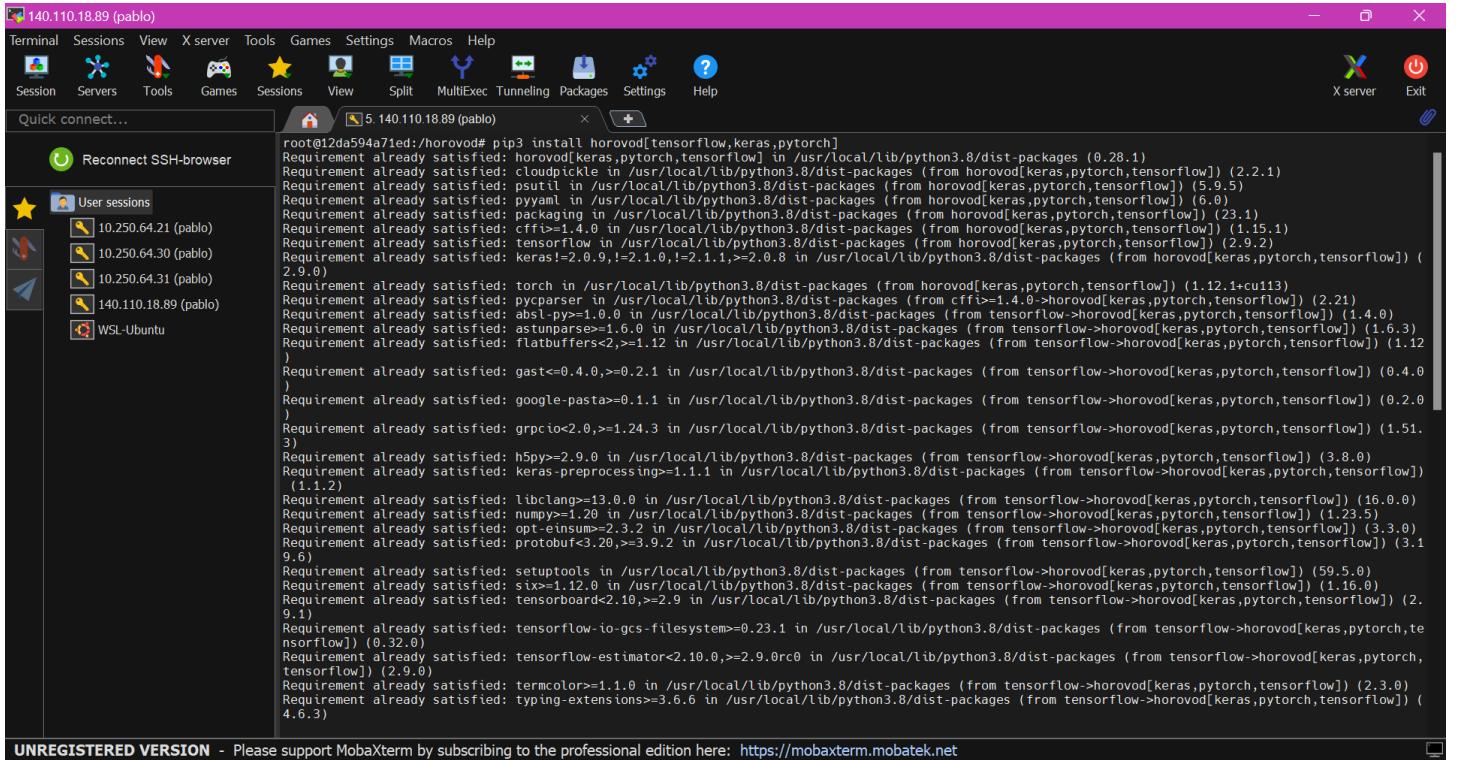
- Reconnect SSH-browser
- User sessions
 - [x] 10.250.64.21 (pubo)
 - [x] 10.250.64.3 (pubo)
 - [x] 10.250.64.3 (pubo)
 - [x] 10.250.64.3 (pubo)
 - [x] 140.110.18.89 (pubo)
 - [x] WSL-Ubuntu

A scrollable command history is shown below the session list, containing numerous commands related to Docker and network configuration. At the bottom, there's a "Status: Downloaded never image for horovod/horovod:latest" message and a "OKAY" button.

The versions for the most important libraries are:

A screenshot of the X server interface on a Linux desktop. The top bar shows the title "140.110.18.89 (pablo)" and various menu options like Terminal, Sessions, View, X server, Tools, Games, Settings, Macros, Help, and a session switcher. Below the menu is a toolbar with icons for Session, Servers, Tools, Games, Sessions, View, Split, MultiExec, Tunneling, Packages, Settings, and Help. On the left, there's a sidebar with "Quick connect..." and a "Reconnect SSH-browser" button. The main area has a terminal window titled "root@12da594a7ied:/horovod/examples#". It displays two command outputs: "cmake --version" showing "cmake version 3.16.3" and "g++ (Ubuntu 9.4.0-1ubuntu1-20.04.1) 9.4.0 Copyright (C) 2019 Free Software Foundation, Inc. This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE." followed by "python3" output showing Python 3.8.10 and TensorFlow version 2.9.2.

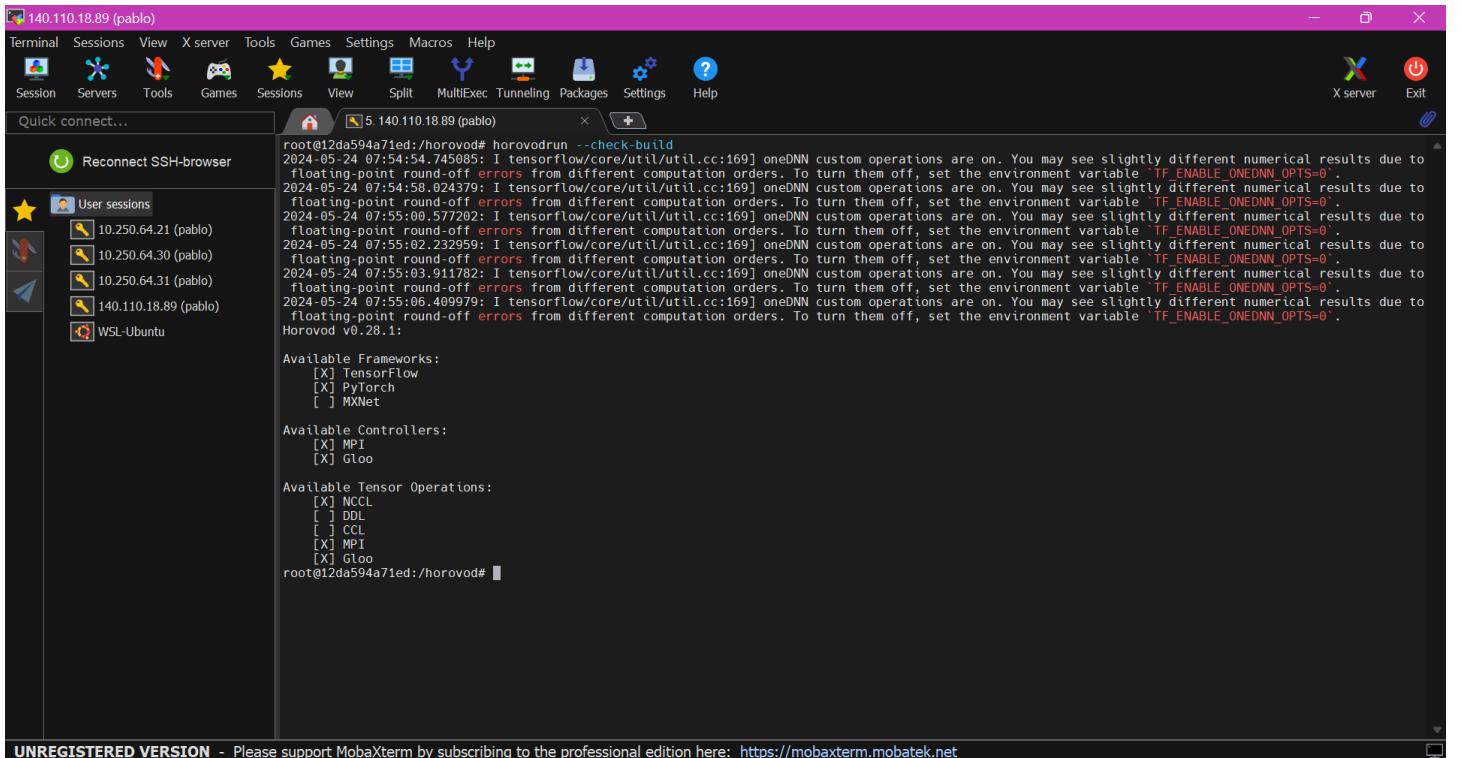
Finally, to install Horovod and the corresponding frameworks, we execute the following command as well as the checking line to verify that we downloaded the specified ones:



```
root@12da594a71ed:/horovod# pip3 install horovod[tf,keras,pytorch]
Requirement already satisfied: horovod[tf,keras,pytorch,tensorflow] in /usr/local/lib/python3.8/dist-packages (0.28.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.8/dist-packages (from horovod[tf,keras,pytorch,tensorflow]) (2.2.1)
Requirement already satisfied: psutil in /usr/local/lib/python3.8/dist-packages (from horovod[tf,keras,pytorch,tensorflow]) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.8/dist-packages (from horovod[tf,keras,pytorch,tensorflow]) (6.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from horovod[tf,keras,pytorch,tensorflow]) (23.1)
Requirement already satisfied: cffi>=1.4.0 in /usr/local/lib/python3.8/dist-packages (from horovod[tf,keras,pytorch,tensorflow]) (1.15.1)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.8/dist-packages (from horovod[tf,keras,pytorch,tensorflow]) (2.9.2)
Requirement already satisfied: keras<=2.0.9,>=2.1.0,>=2.1.1,>=2.0.8 in /usr/local/lib/python3.8/dist-packages (from horovod[tf,keras,pytorch,tensorflow]) (2.9.0)
Requirement already satisfied: torch in /usr/local/lib/python3.8/dist-packages (from horovod[tf,keras,pytorch,tensorflow]) (1.12.1+cu113)
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (2.21)
Requirement already satisfied: absl-py<1.0.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (1.4.0)
Requirement already satisfied: astunparse<1.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (1.6.3)
Requirement already satisfied: flatbuffers<2,>=1.12 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (1.12)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (1.51.3)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (3.8.0)
Requirement already satisfied: keras-preprocessing<1.1.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (1.1.2)
Requirement already satisfied: libclang<=13.0.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (16.0.0)
Requirement already satisfied: numpy<=1.20 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (1.23.5)
Requirement already satisfied: opt-einsum<=2.3.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (3.3.0)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (3.19.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (59.5.0)
Requirement already satisfied: six<=1.12.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (1.16.0)
Requirement already satisfied: tensorflow<2.10,>=2.9 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (2.9.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem<=0.23.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (0.32.0)
Requirement already satisfied: tensorflow-estimator<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (2.9.0)
Requirement already satisfied: termcolor<=1.1.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (2.3.0)
Requirement already satisfied: typing-extensions<=3.6.6 in /usr/local/lib/python3.8/dist-packages (from tensorflow>horovod[tf,keras,pytorch,tensorflow]) (4.6.3)

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
```

For checking the installation, we proceed as follows:



```
root@12da594a71ed:/horovod# horovod -check-build
2024-05-24 07:54:54.745085: I tensorflow/core/util/util.cc:169] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable TF_ENABLE_ONEDNN_OPTS=0.
2024-05-24 07:54:58.024370: I tensorflow/core/util/util.cc:169] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable TF_ENABLE_ONEDNN_OPTS=0.
2024-05-24 07:55:00.577202: I tensorflow/core/util/util.cc:169] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable TF_ENABLE_ONEDNN_OPTS=0.
2024-05-24 07:55:02.232959: I tensorflow/core/util/util.cc:169] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable TF_ENABLE_ONEDNN_OPTS=0.
2024-05-24 07:55:03.911782: I tensorflow/core/util/util.cc:169] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable TF_ENABLE_ONEDNN_OPTS=0.
2024-05-24 07:55:06.409979: I tensorflow/core/util/util.cc:169] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable TF_ENABLE_ONEDNN_OPTS=0.
Horovod v0.28.1:

Available Frameworks:
[X] TensorFlow
[X] PyTorch
[ ] MXNet

Available Controllers:
[X] MPI
[X] Gloo

Available Tensor Operations:
[X] NCCL
[ ] DDL
[ ] CCL
[X] MPI
[X] Gloo
root@12da594a71ed:/horovod#
```

4 Week 4: 27-31/05/2024

This week I had to understand plenty of different models that are either mentioned or present in the architecture of AnomalyGPT, so still, I need time to understand the surrounding models and concepts before jumping straight into the detailed explanation of AnomalyGPT.

4.1 ImageBind

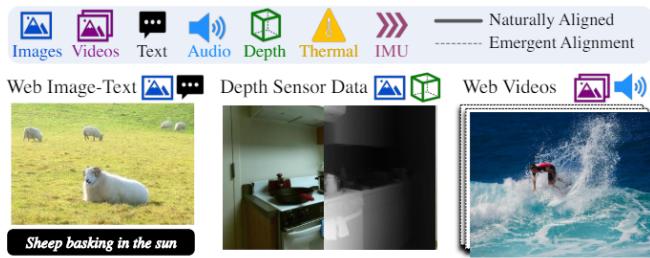
4.1.1 General Overview

ImageBind is an advanced AI model that performs multimodal learning, a process in which the model learns from multiple types of data (images, text, audio, thermal, depth and IMU) simultaneously. The model was developed by Meta AI and published on a research paper released in May 2023.

The original research paper can be found [here](#): ImageBind

This capability enables ImageBind to **bind** together various sensory inputs into a unified representation, allowing more comprehensive understanding and generation across different data types. To picture the model, we can consider the image of a beach which might remind us the sound of waves, the texture of the sand, the breeze, the warmth of the sun or even inspire a poem. The key aspects of ImageBind are:

1. **Multimodal Learning:** It was designed to **integrate information from various modalities** such as videos, audios, Inertial Movement Unit (IMUs), texts, depth and thermal images.
2. **Unified Representation:** The core concept of ImageBind is to **create a unified representation space where inputs from different modalities are mapped**. This leads to an understanding of relationships and correlations between different types of data, allowing it to generate responses or perform tasks that involve multiple sensory inputs.
3. **Architecture:** ImageBind employs a **transformer-based architecture** (like GPT4) which includes encoders (also transformers) for each modality that allows to process the respective inputs and map them into the unified representation space.
4. **Training:** As it can be expected from a massive model like ImageBind, **training the model involves a large-scale data and extensive computational resources**. The model is trained on diverse datasets that include multimodal information. Distributed parallel training techniques are often used to handle the immense computational requirements and to accelerate the training process.
5. **Applications:** One of the many fields in which ImageBind can be applied to is **Cross-Modal Retrieval**, i.e, the ability to search relevant images based on text descriptions or vice versa, and content generation, which allows creating multimedia content that aligns with text prompts (generating images from audio, for instance).
6. **ImageBind's Goal:** The goal of this model is to create a unified representation space (or a **single joint embedding space**) that integrates multiple sensory modalities to enable sophisticated zero-shot recognition and retrieval capabilities. Each modality's embedding, is aligned with the image embedding that is associated with, such as text to image using web data (text-image pairs) and IMU to video.



For instance, video data captured from egocentric cameras (cameras worn on the body to capture a first-person perspective) can be paired with IMU data to learn the relationship between visual and motion information (angular rate, accelerometer, gyroscopes and sometimes even magnetometer).

4.1.2 Modalities Alignment

Alright, we understand that it's a model composed of multiple models that help encode each modality of data into a common representation space, but how is it performed?

The main objective they set with ImageBind is to align the embeddings from different pairs of modalities to create a **common image embedding**.

- **Methodology:** Images are used as the **central modality** to which all other modalities are aligned. This approach not only simplifies the multi-modal embedding processing but leverages the robust pre-existing models for image processing.
- **Pairing Strategies:** For each modality, a model will be in charge of encoding the embeddings and aligning them properly with the picture it represents.

1. **Text to Image:** Text embeddings are aligned with image embeddings using large-scale web data models like CLIP, which are used to learn these alignments effectively.

CLIP is an OpenAI model released in 2021 whose acronym stands for Contrastive Language Image Pre-training. Contrastive learning is a general technique for learning an embedding space by using pairs of related examples (**positives**) and unrelated examples (**negatives**).

2. **Audio to Video:** Naturally paired data is used such as video clips with their corresponding audio tracks. Datasets where audio and video are recorded together, allowing the model to learn the temporal and contextual relationships are as well used.
3. **Depth & Thermal to Image:** The alignment is done by pairing these data types with their corresponding images. Both, depth and thermal data are already images, therefore we have already reached the common ground image embedding.
4. **IMU to Video:** Aligning IMU embeddings to video embeddings using data captured from egocentric cameras with IMU sensors. The model learns to map the motion data from the IMU to corresponding visual features in the video. This alignment is done by training a model to associate specific motion patterns with visual sequences.

For instance, the process for IMU to video alignment would be the data collection (IMU data and the recordings with egocentric cameras), the embedding alignment using a model trained to such task and finally, with the IMU embeddings properly aligned with their corresponding video scenes we can pass from videos to images naturally as videos consist of frames which are individual images. By extracting frames from the videos, the model learns the visual features that correspond to different video segments.

Same process is applied when aligning audio to video embeddings and then extracting the audio embeddings that correspond to frames from the video, therefore obtaining audio-images embeddings correctly aligned.

The core idea is that after obtaining separately each embedding from the different modalities, the complexity resides in integrating any of the modalities embeddings with the corresponding image embedding space where both modalities can coexist and interact meaningfully. This space is learned such that semantically similar modality and image pairs are close to each other.

4.1.3 Zero-Shot Image Classification

Zero-Shot recognition is the **capability in machine learning models to correctly identify/categorize objects, actions or entities that it has never seen during its training phase**. This method was popularised by the CLIP model, which aligns image and text embedding into a common space where semantically pairs are close together. To

perform zero-shot classification, a list of text descriptions (prompts) representing the possible classes in a dataset is constructed.

For instance, if the task is to classify images of animals, the text prompts might be "a photo of a cat", "a photo of a dog" and so on. When an image is provided, the model computes its embedding in the common space. Similarly, it computes the embeddings for all text prompts, and finally the image is classified based on its similarity to the text prompt embeddings. ImageBind, unlike CLIP, achieves this without the need of paired text data to identify the class.

4.1.4 Loss Function

ImageBind uses **pairs of modalities** (\mathbb{I}, \mathbb{M}) where \mathbb{I} denotes an image and \mathbb{M} any modality (audio, text, depth...), to learn a single joint embedding. Given an image \mathbb{I}_i and its corresponding observation in a certain modality \mathbb{M}_i , we encode them into **normalized embeddings** denoted $q_i = f(\mathbb{I}_i)$ and $k_i = g(\mathbb{M}_i)$ where f, g are deep networks (encoders). The embeddings are optimized using an InfoNCE loss, defined as follows:

$$\mathcal{L}_{\mathbb{I}, \mathbb{M}} = -\log \left[\frac{e^{\frac{q_i^T \cdot k_i}{\tau}}}{e^{\frac{q_i^T \cdot k_i}{\tau}} + \sum_{j \neq i} e^{\frac{q_i^T \cdot k_j}{\tau}}} \right] \text{ where } \begin{cases} \tau \text{ is a scalar temperature that controls} \\ \text{the smoothness of the softmax distribution} \\ j \text{ denotes the unrelated or negative observations} \end{cases}$$

InfoNCE, where NCE stands for Noise-Contrastive Estimation, is a type of contrastive loss function used for self-supervised learning. In fact, **ImageBind uses a symmetric loss**: $\mathcal{L}_{\mathbb{I}, \mathbb{M}} + \mathcal{L}_{\mathbb{M}, \mathbb{I}}$. There can be highlighted 2 main reasons to make such decision.

1. **Bidirectional Alignment:** It ensures consistency which is crucial for tasks like cross-modal retrieval, where you want to retrieve text from an image and also retrieve an image from text with equal accuracy.
2. **Balanced Representation:** Using a one-directional loss could lead to biased embeddings where one modality (e.g., images) is more accurately represented than the other (e.g., text). Symmetric loss balances the representation, ensuring equally importance. Besides, a balanced embedding space generalizes better to unseen data, improving zero-shot performance.

4.1.5 Emergent Alignment

Something surprising from this model is that, even though it was trained only on pairs like $(\mathbb{I}, \mathbb{M}_1)$ and $(\mathbb{I}, \mathbb{M}_2)$, an **emergent behaviour is the embedding space occurred where can be found modality pairs** $(\mathbb{M}_1, \mathbb{M}_2)$. This enables zero-shot and cross-modal tasks without specific training for those tasks, such as classifying audio-text without being trained on specifically.

4.1.6 Architecture Description

As previously mentioned, the architecture of ImageBind is based mainly on transformers. Let's review the models used in for each encoder (transformers as well) and explain how they work.

- **Images & Video:** The model transformer-based for encoding both, **images and videos**, is the same one, the **Vision Transformer ViT-H 630M** parameters. They can use ViT-H 630M for videos as well because they temporally inflate the model so it can process video data by extending the 2D path projection layer to handle an additional temporal dimension. This allows to analyze short video clips (2 frames extracted from 2 second intervals).
- **Audio:** The model used for encoding **audio** is the **Audio Spectrogram Transformer (AST)**, which is a convolution free, purely attention-based model for audio classification. A 2-second audio clip (sampled at 16KHz) is transformed into a **spectrogram** (image of the audio frequencies over time). Then, they use **ViT-B** (B stands for base, it's the baseline model from the Vision Transformers family) to encode the spectrogram.

- **Text:** Text data is encoded using CLIP as previously mentioned, however to be precise, they use OpenCLIP 302M.
- **Thermal & Depth:** For this modality of data, the model ViT-S (S stands for small) will be in charge of encoding the embeddings using one-channel color versions of them.
- **IMU:** Inertial Measurement Units signals consisting from accelerometer and gyroscopes measurements across X, Y and Z axes are extracted, This adds up a total of 6 values (3 for acceleration and 3 for rotational motion). The data is collected in 5 second clips. Over these 5 seconds, the IMU records 2000 time steps of data. Then, with those 2000 rows of data (times 6), we apply a 1D convolutional neural network to process the sequential data (as time series) to extract meaningful features and compact data.

The input data shape is (2000,6), and with a kernel size of 8, the convolution operation considers 8 consecutive time steps at a time, and considering a stride of 1 (filter moves one time step forward after each operation). The convolutional filter computer a weighted sum of the input features within its window (8 time steps) and applies an activation function to generate the output. Finally, the resulting sequence is encoded using a lightweight 6 layer encoder transformer with 512 dimensional width and 8 heads (trained for 8 epochs).

After using the different transformer encodes, to ensure a normalized embeddings for all modalities, they include on top of each encoder a "modality specific linear projection head" to output a fixed size d dimensional embedding, which is normalized and used in the previous InfoNCE loss function. All along the training the embedding encoders are updated for all modalities, allowing the encoders to learn and adapt their representations to match accurately the image representation of the entity, action or scene.

In conclusion, ImageBind is a multi-modal transformer-based encoder model that can be integrated into complementary models to leverage and improve the variability of data to train on.

4.2 PandaGPT

PandaGPT is a groundbreaking model designed to empower large language models with visual and auditory instruction-following capabilities. Developed by researchers from institutions such as the University of Cambridge and Tencent AI Lab, PandaGPT aims to integrate and understand information from multiple modalities simultaneously, much like humans perceive and interpret diverse sensory inputs.

The original research paper can be found here: [PandaGPT](#)

4.2.1 Multi-Modal Integration

PandaGPT's components are quite simple. It combines multimodal encoders from ImageBind with the language capabilities from Vicuna-7B (LLM). On one hand, ImageBind is in charge of generating embeddings for various modalities and on the other hand, Vicuna processes text inputs and generates responses based on the integrated embeddings from the multimodal data. This multimodal integration is enhanced with Low-Rank Adaptation (LoRA) weights to improve the handling of multimodal information without extensive re-training. Obviously, the modalities covered are the same as ImageBind, including video, images, audio, depth, thermal and IMU.

4.2.2 Training Process

4.2.2.1 Dataset Preparation

PandaGPT is trained on 160k image-language instruction-following pairs. Each training instance includes an image \mathcal{I} and a conversation sequence (based on instruction and response pairs defined as $(\mathcal{I}, (x_1, y_1, x_2, y_2, \dots, x_n, y_n))$.

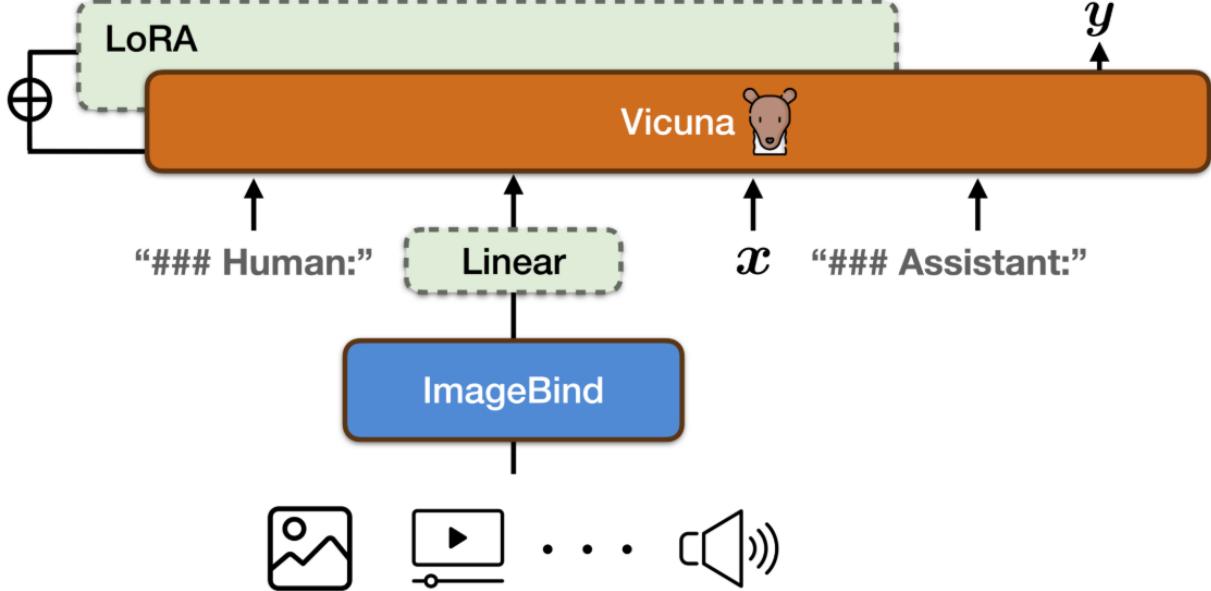


Figure 1: Illustration of PandaGPT. During training, we only train the linear projection matrix and the additional LoRA weights (as indicated with dashed boxes) while keeping the parameters of ImageBind and Vicuna frozen.

4.2.2.2 Training Strategy

As PandaGPT combines both models, ImageBind and Vicuna, the model needs to create a bridge between them, that's why it uses a [linear projection matrix \(LPM\)](#) and [LoRA weights](#). The linear projection marix connects ImageBind embeddings to Vicuna's feature space. The LoRA weights are applied on the attention modules of Vicuna, [significantly reducing the number of trainable parameters to around 0.4%](#) of the total parameters of Vicuna. Besides, most parameters of ImageBind and Vicuna remain frozen to leverage their pre-trained capabilities.

4.2.2.3 Objective Function

The objective function for PandaGPT aims to [maximize the probability of generating the correct response](#) during training. This involves [adjusting the learnable parameters to ensure the model's output aligns as closely as possible with the expected responses](#). The objective function considered is the following one:

$$L(\theta_f, \theta_l) = \prod_{i=1}^n p_\theta(y_i|x_{<i}, y_{<i-1}, f(h_I)) \text{ where } \begin{cases} \theta_f, \theta_l \text{ are learnable parameters (LPM, LoRA respectively)} \\ x_i \text{ is the embedding instruction at moment i-th} \\ y_i \text{ is the embedding response at (i-1)-th previous moments} \\ f(h_I) \text{ is the image representation embedding} \end{cases}$$

The goal is to maximize the probability $p_\theta(y_i|x_{<i}, y_{<i-1}, f(h_I))$, which represents the likelihood of generating the correct response y_i given the previous instructions ($x_{<i}$), responses ($y_{<i-1}$), and image embedding ($f(h_I)$). In probabilistic modeling, maximizing the likelihood function ensures that the model parameters are optimized to make the observed data (training data) most probable. This way, [the model learns to generate accurate responses that align well with the given instructions](#).

4.2.3 Training Steps

1. **Image Encoding:** The [image](#) is passed through ImageBind to generate an [embedding](#) h_I .
2. **Text Encoding:** [Instructions and responses are encoded](#) using Vicuna, generating text embeddings.

3. **Alignment:** The [image embedding](#) h_I is aligned with the text embeddings via a linear projection matrix f .

4.2.4 Example of Training Instances

Imagine a detailed image of a machine part with visible defects. The conversation sequence could look like as:

1. **Instruction 1:** "Describe the condition of the machine part in the image."

- **Expected Response 1:** "The machine part appears to be generally in good condition but has some visible wear and tear."
- **Model Response:** PandaGPT attempts to generate a response similar to the expected one, using the image embedding to inform its description.

2. **Instruction 2:** "Identify any visible defects in the image."

- **Expected Response 2:** "There is a small crack near the bottom-left corner and some surface scratches."
- **Model Response:** PandaGPT uses the image embedding to identify and describe defects, trying to align its response closely with the expected answer.

3. **Instruction 3:** "What might have caused these defects?"

- **Expected Response 3:** "The crack could be due to mechanical stress, and the scratches may result from friction during operation."
- **Model Response:** PandaGPT generates a response considering the possible causes, again informed by the image embedding.

4.2.5 Key Innovations

PandaGPT seamlessly [integrates and understands multimodal inputs](#) displaying zero-shot capabilities and by only training a small fraction of the model parameters, PandaGPT achieves its capabilities [with reduced computational resources and training time](#).

4.3 LlaVA

The model LlaVA, which stands for [Large Language and Vision Assistant](#), is an end-to-end (E2E) trained large multimodal model, released in December 2023 by Microsoft Research, that connects a vision encoder and a LLM for general purpose visual and language understanding. Experiments show that LlaVA exhibits the behaviours of multimodal GPT4 on unseen images/instructions, yielding a 85.1% relative score compared with GPT4 on a synthetic multimodal instruction-following dataset. The official website from the model, which contains a demonstration, the research paper and where can be found the github code, is well explained.

4.3.1 Architecture

LlaVA employs the pre-trained [CLIP visual encoder ViT-L/14 \(Large\)](#) and connects it to [the LLM Vicuna-v0](#) using a lightweight linear projection matrix. This setup transforms visual features into language embeddings tokens, matching the dimension from the text embedding tokens from Vicuna.

4.3.2 Training Process

LlaVA's training involves 2 main stages.

4.3.2.1 Pre-Training for Feature Alignment

In the **first stage**, it was utilized a **filtered subset of CC3M of 595k image-text pairs** (the complete dataset is made up of 3 million images and its corresponding captions). The goal is to align the features extracted from images with the word embeddings used by a pre-trained language model (LLM).

In order to align properly both informations, we **apply a naïve expansion method** which turns each image-text pair from the filtered dataset into a single-turn conversation. For instance, for an image X_v of a cat, the question X_q might be "What do you see in the image?", and the answer X_a would be the original caption (ground-truth prediction), like "A cat sitting on a sofa."

Both the **visual encoder** (which processes the image) and the **pre-trained LLM** (which processes the text) have their **weights frozen**. The **only parameters** that are **updated** during this training stage are those in the **projection matrix W**. This projection matrix is responsible for aligning the image features with the word embeddings of the LLM.

The objective is to **maximize the likelihood that the model correctly predicts the caption** (answer X_a) given the image and the instruction. By training the projection matrix, the image features H_v are transformed in a way that makes them compatible with the word embeddings of the LLM.

$$H_v = W \cdot Z_v, \text{ with } Z_v = g(X_v) \text{ where} \left\{ \begin{array}{l} X_v \text{ is the input image} \\ g \text{ is the visual encoder ViT-L/14} \\ Z_v \text{ are the visual extracted visual features} \\ W \text{ is the trainable projection matrix} \\ H_v \text{ are the language embeddings token from} \\ (\text{the visual features with same dimensionality}) \\ \text{as the word embeddings from the LLM} \end{array} \right.$$

Let's consider the following example to understand.

Example Setup	3. Projection Matrix
<p>1. Dataset: Let's say we have a very small dataset with just one image-text pair.</p> <ul style="list-style-type: none"> Image: An image X_v of a dog. Caption (Ground-Truth Answer X_a): "A brown dog is playing in the park." <p>2. Question (Instruction X_q): "What is in the image?"</p>	<p>We need to train a projection matrix W that will map the visual feature vector H_v to the space of the word embeddings.</p> <ul style="list-style-type: none"> Assume W is a 3×3 matrix initialized randomly: $W = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{bmatrix}$
<p>Step-by-Step Process</p> <p>1. Extract Image Features:</p> <ul style="list-style-type: none"> The image X_v is passed through the visual encoder, which outputs a feature vector H_v. Let's assume the visual encoder outputs a feature vector of size 3: $H_v = [1.2, -0.5, 2.3]$. <p>2. Word Embeddings for the Caption:</p> <ul style="list-style-type: none"> The words in the caption "A brown dog is playing in the park" are converted into word embeddings using the pre-trained LLM. Assume the embeddings for the words are as follows (each word is represented by a 3-dimensional vector for simplicity): <ul style="list-style-type: none"> "A": [0.1, 0.2, 0.3] "brown": [0.4, 0.5, 0.6] "dog": [0.7, 0.8, 0.9] "is": [0.1, 0.0, -0.1] "playing": [0.2, -0.2, 0.0] "in": [0.1, -0.1, 0.1] "the": [0.0, 0.1, 0.0] "park": [0.5, 0.6, 0.7] 	<p>4. Transform Visual Features:</p> <ul style="list-style-type: none"> The visual feature vector H_v is multiplied by the projection matrix W to obtain the aligned features: $H'_v = H_v \cdot W = [1.2, -0.5, 2.3] \cdot \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{bmatrix}$ <ul style="list-style-type: none"> Performing the matrix multiplication: $H'_v = [1.2 \cdot 0.1 + (-0.5) \cdot 0.4 + 2.3 \cdot 0.7, \\ 1.2 \cdot 0.2 + (-0.5) \cdot 0.5 + 2.3 \cdot 0.8, \\ 1.2 \cdot 0.3 + (-0.5) \cdot 0.6 + 2.3 \cdot 0.9]$ $H'_v = [1.61, 1.36, 1.11]$
	<p>5. Alignment Objective:</p> <ul style="list-style-type: none"> The objective is to maximize the likelihood that H'_v correctly predicts the word embeddings of the caption "A brown dog is playing in the park". In practice, this means adjusting W to minimize the difference between H'_v and the word embeddings of the caption words.

4.3.2.2 E2E Fine-Tuning

After the initial alignment of visual features with word embeddings in Stage 1, the next step is to fine-tune the entire model end-to-end. This **second stage** involves **fine-tuning both the visual encoder and the language model**

to improve their joint performance on the task.

We continue [using the filtered dataset](#) from Stage 1, which contains image-text pairs converted into instruction-following data. Each sample in the dataset is treated as a conversation turn, where the model needs to generate an appropriate response based on the given image and instruction

The loss function typically used is the negative log-likelihood (NLL) or cross-entropy loss, which measures how well the model’s predicted response matches the ground truth response. The [entire model is trained end-to-end](#), meaning the gradients are backpropagated through both the visual encoder and the language model.

4.3.3 Loss Function

During the training, the model is optimized to [maximize the likelihood of the correct response sequences](#). This is done by minimizing the negative log-likelihood of the target tokens, then the model parameters are adjusted to reduce the loss, thereby learning to generate responses closer to the expected target:

$$\mathcal{L} = - \sum_{i=1}^L \log[p_\theta(x_i | X_v, X_{\text{Instruct}}, x_{<i})]$$

Here, x_i is the i -th token in the ground-truth response, and $p_\theta(x_i | X_v, X_{\text{Instruct}}, x_{<i})$ is the probability assigned to that token by the model.

4.3.4 Training Instances

Let’s understand the training process in details. Suppose we have an image X_v of a cat sitting on a sofa. The conversation might be about identifying objects in the image and describing their attributes.

1. **Instruction 1:** $X_{\text{Instruct}}^1 = [X_q^1, X_v]$

- **Question 1:** X_q^1 is "What do you see in the image?".
- **Expected Model Answer 1:** X_a^1 is "I see a cat sitting on a sofa.".

The expected model answer is the expected target output that we want the model to learn to generate. Here the model would generate a response which might be initially incorrect or incomplete, and in order to assess the quality of the response, the target response is tokenized into:

"I see a cat sitting on a sofa." \Rightarrow ["I", "see", "a", "cat", "sitting", "on", "a", "sofa", "."]

A similar step would be applied to the generated answer. Then, the probability computation proceeds as follows:

1. $p_\theta(\text{"I"} | X_v, X_{\text{Instruct}})$
2. $p_\theta(\text{"see"} | X_v, X_{\text{Instruct}}, \text{"I"})$
3. $p_\theta(\text{"a"} | X_v, X_{\text{Instruct}}, \text{"I"}, \text{"see"})$

And so on...

The text token "I" is embedded into a vector representation using an embedding layer composed of logits (raw scores, float values) that indicate the model’s relative preference for each token, then the probability is calculated following a softmax distribution:

$$p_\theta(X_i | X_v, X_{\text{Instruct}}, X_{<i}) = \frac{\exp^{\text{logit}_i}}{\sum_j \exp^{\text{logit}_j}}$$

2. **Instruction 2:** $X_{Instruct}^2 = [X_q^2, X_v]$

- **Question 2:** X_q^2 is "Describe the color of the cat."
- **Expected Model Answer 2:** X_a^2 is "The cat is white with black spots."

The second question includes the context from the previous turn. The model is trained by comparing its generated responses to these expected target responses.

4.3.5 Probability Computation

The model learns to predict the sequence of answers given the instructions and image. For instance, for the previous example of a conversation sequence of length $L = 2$, we would have that the probabilities calculations for $p_\theta(X_a|X_v, X_{Instruct})$ are:

$$p_\theta(\underbrace{\text{"I see a cat sitting on a sofa."}}_{X_a^1} | X_v, \underbrace{\text{"What do you see in the image?"}}_{X_q^1}) \times \\ p_\theta(\underbrace{\text{"The cat is white with black spots."}}_{X_a^2} | X_v, \underbrace{\text{"Describe the color of the cat."}}_{X_q^2}, \underbrace{\text{"I see a cat sitting on a sofa."}}_{X_a^1})$$

The probabilities in LlaVA's training process are determined using an **auto-regressive model** (to predict next token in the sequence), where the likelihood of generating each token in the response sequence is calculated based on the image, instructions and previously generated tokens.

Mathematically, the probability of the entire response sequence X_a is the product of the probabilities of each token given the previous context:

$$p_\theta(X_a|X_v, X_{Instruct}) = \prod_{i=1}^L p_\theta(X_i|X_v, X_{Instruct}, X_{<i}) \begin{cases} p_\theta \text{ is the probability of the } i\text{-th token} \\ X_{<i} \text{ represents the sequence of tokens} \\ \text{preceding } X_i \end{cases}$$

5 Week 5, 6, 7, 8, 9, 10, 11: 03/06/24-20/07/24

5.1 AnomalyGPT

5.1.1 Background

Let's delve into the domain in which the model will attempt to provide good results. Industrial anomaly detection algorithms (IAD) refers to the process of identifying unusual patterns, deviations or anomalies in industrial operations and processes that may indicate potential issues, faults or inefficiencies. These anomalies can signify problems such as equipment malfunctions, process deviations or security breaches, which can lead to downtime, reduced efficiency and increased operational costs if not promptly addressed.

Factories are populated with various sensors to collect data from machinery and processes including temperature, pressure, vibration, sound, images and other operational metrics. By collecting such data, the company can take advantage from multiple benefits such as preventing maintenance, assessing quality control, securing safety at the workplace or improving operational efficiency.

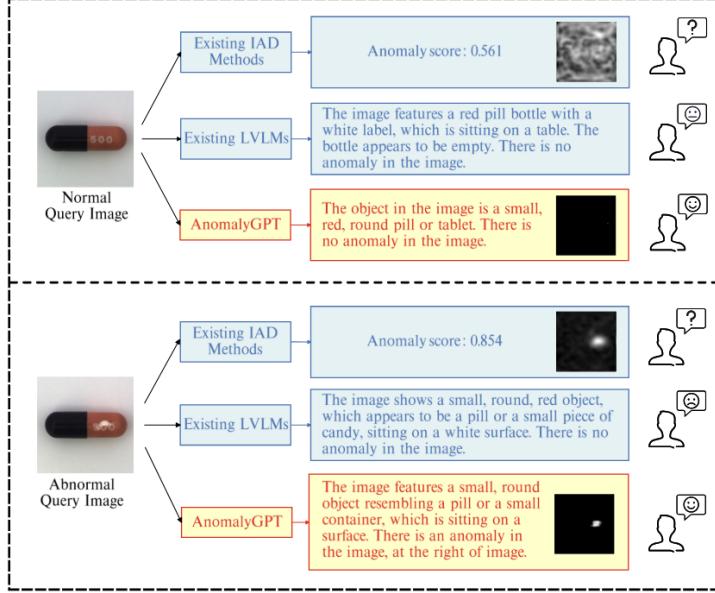
5.1.2 Anomaly Detection Algorithms

Different algorithms have arisen in last years in order to detect the previously mentioned anomalies, including:

- **Threshold-Based Detection:** Simple rules and thresholds to flag anomalies when certain metrics exceed predefined limits. Most existing IAD methods are based on methods defined following this approach, which provide anomaly scores that distinguish between normal and abnormal samples. However, it's not too practical to just obtain a score.
- **Pattern Recognition:** Identifying deviations from established patterns using machine learning AI techniques, which is the approach that we will follow, has proven to be very accurate in order to detect anomalies.
- **Time-Series Analysis:** Analyzing temporal data to detect irregularities it's another method to identify the anomalies, which as well it's based on AI techniques.
- **Real-Time Monitoring:** Implementing systems to continuously monitor data streams and detect anomalies in real-time, enabling prompt responses to potential issues solves as well the problem although it can becomes expensive.

AnomalyGPT is a large vision-language model (LVLM) released in December 2023, that assesses the presence and localization of anomalies, providing textual descriptions of the anomaly for each image. Due to the rarity of real-world samples, models are required to be trained only on normal samples and distinguish anomalous samples that deviate from them.

On one hand, existing LVLMs cannot detect anomalies in images as they lack of domain specific knowledge (because they are trained on large and varied amounts of data sourced from the Internet) and sensitivity to local details. On the other hand, existing IAD methods only provide anomaly scores and need manually threshold setting. Previous reasons justify the need for a model being able to assemble all those capabilities and even extend them by including textual descriptions.



5.1.2.1 Concepts

Some concepts that might help to understand better the model that we'll be introduced:

- **One-Class-One-Model:** Refers to a type of anomaly detection approach where a separate model is trained for each class or type of normal behaviour. The model learns to distinguish normal data from anomalies for that specific class.
- **Few-shot Learning:** Machine learning approach where the model is able to learn and make accurate predictions with a very small amount of labeled data training.
- **Multi-turn Dialogue:** Refers to the capability of a system to engage in a conversation with multiple back-and-forth exchanges, where each response can depend on previous exchanges.
- **In-Context Learning:** Process where a model like a LLM, can learn and perform tasks based on the context provided within the prompts, without needing explicit re-training or fine-tuning.

Methods	Few-shot learning	Anomaly score	Anomaly localization	Anomaly judgement	Multi-turn dialogue
Traditional IAD methods		✓	✓		
Few-shot IAD methods	✓	✓	✓		
LVLMs	✓				✓
AnomalyGPT (ours)	✓	✓	✓	✓	✓

Table 1. Comparison between our AnomalyGPT and existing methods across various functionalities. The “Traditional IAD methods” in the table refers to “one-class-one-model” methods such as PatchCore [23], InTra [21], and PyramidFlow [13]. “Few-shot IAD methods” refers to methods that can perform few-shot learning like RegAD [10], Graphcore [29], and WinCLIP [27]. “LVLMs” represents general large vision-language models like MiniGPT-4 [36], LLaVA [17], and PandaGPT [25]. “Anomaly score” in the table represents just providing scores for anomaly detection, while “Anomaly judgement” indicates directly assessing the presence of anomaly.

5.1.3 Challenges of training LVLMs with IADs

Training LVLMs with IADs face two main challenges:

1. **Data Scarcity:** One of the main challenges is not having enough data specific to the IAD field.
 - **Why it matters?:** LVLMs (like LLaVA and PandaGPT) are pre-trained on vast datasets (160k images with dialogues). In contrast, IADs only have a few thousand images.

- **Consequence:** Training on such small IAD datasets could lead to **overfitting** and **catastrophic forgetting** (the model forgets what he learnt).
- **Solution:** One of the chosen solutions is to use prompt embeddings instead of fine-tuning the model parameters.
- **How it works?** In our case, the raw input would be a small dataset of images showing parts of industrial machinery with some defects. So the prompt embedding would be composed of the encoding image along with a textual description embedding. **For instance**, given an image with an anomaly, the textual description could be: "This is an image of a machinery component. It should be free of defects such as cracks or scratches".

By providing the image encoding, the domain-specific knowledge as just mentioned (which is additional information about normal and common defects), the model receives as input a common embedding which is the result of combining the previous information into a single joint embedding.

2. **Fine Grained Semantic Understanding**: The second challenge is **teaching the model to identify very small, subtle defects in industrial products**, which requires a detailed understanding of the image.

- **Solution:** To solve such issue, the model will include **a decoder and use prompt embeddings**. Traditionally, the decoder from a neural network converts numerical vectors back into human-readable formats. Instead, the mentioned decoder is a component that processes the image to generate detailed, pixel-level anomaly information, which can be used to create embeddings that enrich the input to the LVLM.

- **How it works?**

- Image Analysis:** The decoder analyzes the image and identifies specific areas where anomalies might be present. This involves generating pixel-level information about potential defects.
- Encoding Detailed Information:** The detailed information about these anomalies is encoded into numerical vectors (embeddings).
- Combining Embeddings:** These embeddings are combined with the initial image and any textual description to form a comprehensive prompt that provides the LVLM with contextual information.

Basically, the original image is passed through the image encoder (in the AnomalyGPT base model, the image encoder is ImageBind) that generates an initial image embedding vector which is then processed by the image decoder, model in charge of analyzing and combining the information.

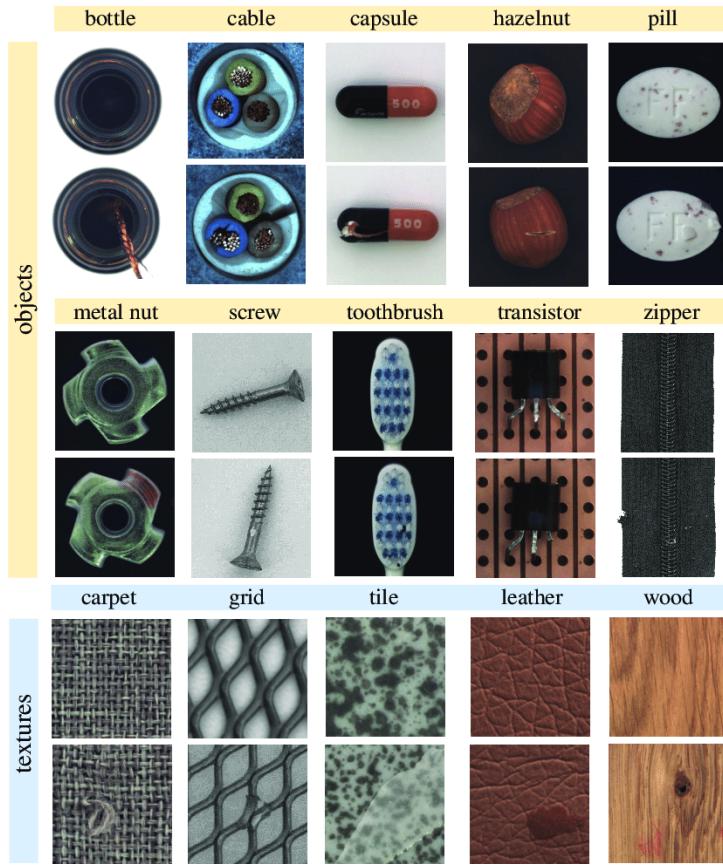
5.1.4 Relevant Datasets on IADs

The most well-known datasets for IAD training, are the following ones:

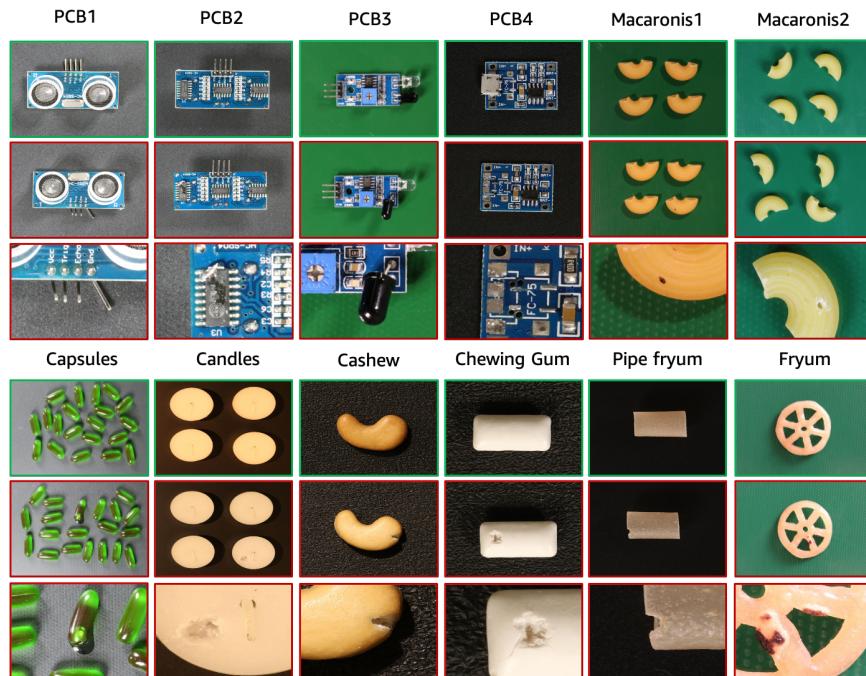
- **MVTec-AD Dataset:** The MVTec Anomaly Detection dataset is designed **for benchmarking anomaly detection and localization algorithms**. It focuses on industrial applications where detecting defects in products is critical.

It **consists of 15 categories** of objects including: **5 texture categories** (carpet, grid, leather, tile and wood) and **10 object categories** (bottle, cable, capsule, hazelnut, metal nut, pill, screw, toothbrush, transistor and zipper).

It's composed of a **total of 3629 high-resolution images for training** and **1725 for testing**, containing defect-free and anomalous images in testing but only defect-free in training set.



- **ViSA Dataset:** The Visual Anomaly dataset is a newly introduced dataset (2022) aimed at advancing the research in industrial anomaly detection and localization. It consists of 12 categories of industrial products, including industrial parts, electronic components and various manufacture items. It's composed of a total of 9261 images and 1200 anomalous samples which include defects such as structural defects, surface imperfections and functional failures.



5.1.5 Architecture

The AnomalyGPT model follows a **single operational path** with a slight difference in input handling depending on whether it receives a single query image or a pair of images (one with anomalies and a normal sample). The operational steps followed by the model and the slight differences between the two input scenarios can be understood as follows:

1. Query Image Input:

- **Single Image Mode:** The model receives a single query image that may contain anomalies.
- **Paired Image Mode:** The model receives a pair of images – one query image that may contain anomalies and a normal image representing the normal sample.

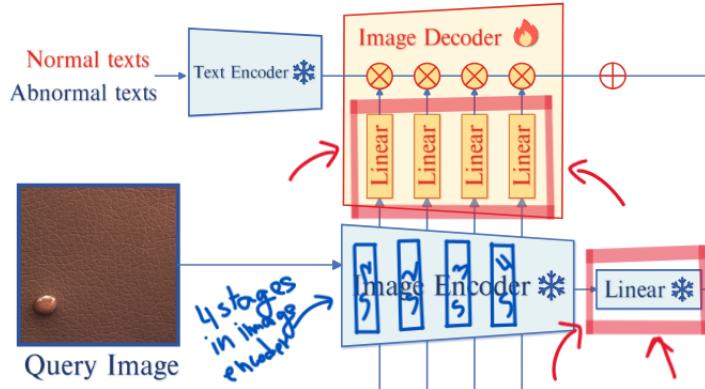
In both cases, let's denote the query image as $x \in \mathbb{R}^{\mathcal{H} \times \mathcal{W} \times \mathcal{C}}$.

2. **Text Encoder:** Normal and abnormal texts are encoded using the **Text Encoder** to generate text features, which are used in the unsupervised setting to aid anomaly detection.
3. **Image Encoder:** The query image is processed by the **Image Encoder** to extract features, which is composed of 4 different stages to obtain the intermediate patch-level features, which we will denote as follows:

$$\mathcal{F}_{patch}^i \in \mathbb{R}^{\mathcal{H}_i \times \mathcal{W}_i \times \mathcal{C}_i} \quad \forall i \in \{1, 2, 3, 4\}$$

Then, when the **image encoder** obtains the intermediate patch-level features, as they don't belong to the same space, it's not possible to align them directly with the text embeddings from the **text encoder**, so **4 linear layers** are needed to make possible the transition and combination, these new features obtained from the linear layers are denoted:

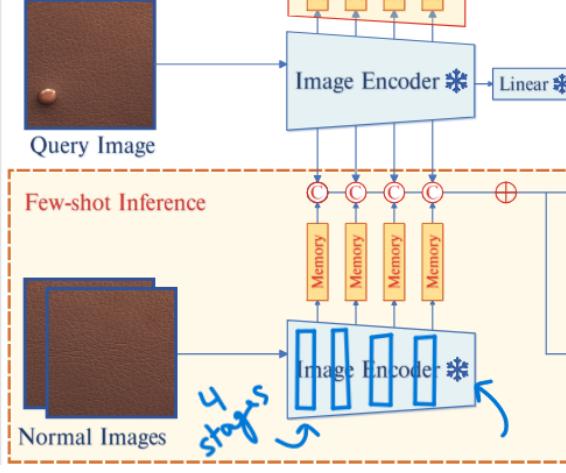
$$\tilde{\mathcal{F}}_{patch}^i \in \mathbb{R}^{\mathcal{H}_i \times \mathcal{W}_i \times \mathcal{C}_i} \quad \forall i \in \{1, 2, 3, 4\}$$



4. **Memory Bank (for Paired Image Mode):** If the paired image mode is used, the normal image is also processed by the **Image Encoder** to extract features, which are then stored in **memory banks**. However, in this case, as no text embeddings will be combined, there's no need to apply linear layers to combine both features, text and image. The normal samples stored in memory banks are denoted as follows:

$$\mathcal{B}^i \in \mathbb{R}^{\mathcal{N} \times \mathcal{C}_i} \quad \forall i \in \{1, 2, 3, 4\}$$

In order to determine how similar each patch-level feature \mathcal{F}_{patch}^i is with its memory bank counterpart \mathcal{B}^i , we need to use the "maximum" function to find the maximum similarity score for each patch between both. Then, by subtracting the maximum similarity score from 1, the resulting term effectively measures the "anomaly score" for each patch. **Higher values** indicate a great likelihood of the patch being anomalous (since it's less similar to any normal patches)



5. Image Decoder: Feature Comparison and Anomaly Localization:

- **Single Image Mode:** The intermediate patch-level features extracted from the image encoder and that have passed through the linear layers, are combined with text features coming from the text encoder in the image decoder to generate pixel-level anomaly localization results. The localization result $M \in \mathbb{R}^{\mathcal{H} \times \mathcal{W}}$ is obtained as follows:

$$M = \text{Upsample}\left(\sum_{i=1}^4 \text{softmax}(\tilde{\mathcal{F}}_{patch}^i \times \mathcal{F}_{text}^T)\right) = \text{Upsample}\left(\sum_{i=1}^4 \left[\frac{\exp \tilde{\mathcal{F}}_{patch}^i \times \mathcal{F}_{text}^T}{\sum k \tilde{\mathcal{F}}_{patch}^i \times \mathcal{F}_{text}^T}\right]\right)$$

[The function "Upsample" in the context of neural networks and image processing is used to increase the spatial resolution of a feature map or an image. Upsampling is the reverse operation of downsampling or pooling and is commonly used in applications such as image generation, segmentation, and super-resolution. Upsample is a function that enlarges a smaller image or feature map to a higher resolution.]

At the same time, the extracted features from the image encoder are compared to the learned (memory bank) normal features (from training data) to detect anomalies. To note that, in this case, as the only input is a single query image, the image is compared with the information stored within the memory banks (training data). The comparison results are used to generate a localization map indicating the presence of anomalies.

- **Paired Image Mode:** Besides following the same top path from the schema (text encoder + image encoder \rightarrow image decoder), the patch-level features of the query image (anomaly or normal) are compared with the stored features from the normal image(s) in the memory bank. The distance between the query patches and the normal patches is calculated to identify deviations, indicating potential anomalies. The comparison results are used to generate a localization map.

$$M = \text{Upsample}\left(\sum_{i=1}^4 [1 - \max(\mathcal{F}_{patch}^i \times \mathcal{B}^{iT})]\right)$$

6. Prompt Learner: The localization results are transformed into prompt embeddings by the Prompt Learner. Learnable base prompt embeddings are also included to provide additional context for the anomaly detection task.

7. Large Language Model: The LLM processes the image embedding, the prompt embeddings, and any user-provided textual input (e.g., "Is there any anomaly in the image?"). Ultimately, the LLM is ready to provide a detailed response indicating whether an anomaly is present, where it is located within the image, and a description of the anomaly.

5.1.5.1 Slight Differences in Input Handling

- Single Query Image Mode
 - **Comparison Basis:** The model compares the query image features with the normal features learned during training.
 - **Output Quality:** If the query image is related to the training dataset, the model performs well in detecting, localizing, and describing the anomaly. If the query image is unrelated to the training data, the model's performance may degrade because it lacks relevant information for accurate comparison.
- Paired Image Mode
 - **Comparison Basis:** The model compares the query image features with the features of the provided normal image stored in the memory bank.
 - **Output Quality:** The model can still determine whether an anomaly is present and localize it within the query image. If the query image is related to the training dataset, the model can provide a detailed description of the anomaly. If the query image is unrelated to the training data, the description may not be accurate, but the model can still localize anomalies based on the provided normal sample.

5.1.6 Loss Functions

5.1.6.1 Cross-Entropy Loss

Cross-entropy loss is commonly employed for training language models, which quantifies the disparity between the text sequence generated by the model and the target text sequence. The formula is as follows:

$$\mathcal{L}_{ce} = - \sum_{i=1}^n y_i \cdot \log(p_i) \text{ where } \begin{cases} n \text{ is the number of tokens} \\ y_i \text{ is the true label} \\ p_i \text{ is the predicted probability} \end{cases}$$

5.1.6.2 Focal Loss

Focal loss is commonly used in object detection and semantic segmentation to address the issue of class imbalance, which introduces an adjustable parameter γ to modify the weight distribution of cross-entropy loss, emphasizing samples that are difficult to classify. In IAD task, where most regions in anomaly images are still normal, employing focal loss can mitigate the problem of class imbalance. Focal loss can be calculated as follows:

$$\mathcal{L}_{focal} = -\frac{1}{n} \sum_{i=1}^n (1 - p_i)^\gamma \cdot \log(p_i) \text{ where } \begin{cases} n = \mathcal{H} \times \mathcal{W} \text{ is the number of pixels} \\ \gamma (= 2) \text{ is a tunable parameter} \\ p_i \text{ is the predicted probability of positive classes} \end{cases}$$

5.1.6.3 Dice Loss

Dice loss is a commonly employed loss function in semantic segmentation tasks. It is based on the dice coefficient and can be calculated as follows:

$$\mathcal{L}_{dice} = -\frac{\sum_{i=1}^n y_i \cdot \hat{y}_i}{\sum_{i=1}^n y_i^2 + \sum_{i=1}^n \hat{y}_i^2} \text{ where } \begin{cases} n = \mathcal{H} \times \mathcal{W} \text{ is the number of pixels} \\ y_i \text{ is the output of the decoder} \\ \hat{y}_i \text{ is the ground-truth value} \end{cases}$$

5.1.6.4 Overall Loss Function

Combining the three previously mentioned loss functions, the overall loss function is defined as follows:

$$L = \alpha \cdot \mathcal{L}_{ce} + \beta \cdot \mathcal{L}_{focal} + \gamma \cdot \mathcal{L}_{dice} \text{ where } \alpha, \beta, \gamma \text{ are set to 1.}$$

5.2 Code Understanding

5.2.1 Origin Script: train_mvtec.sh

The complete training from AnomalyGPT with the dataset MVTec can be found on the shell script "train_mvtec.sh" which contains the main arguments needed to train the model:

```
#!/bin/bash
deepspeed --include localhost:0 --master_port 28400 train_mvtec.py \
--model openllama_peft \
--stage 1 \
--imagebind_ckpt_path ../pretrained_ckpt/imagebind_ckpt/imagebind_huge.pth \
--vicuna_ckpt_path ../pretrained_ckpt/vicuna_ckpt/7b_v0/ \
--delta_ckpt_path ../pretrained_ckpt/pandagpt_ckpt/7b/pytorch_model.pt \
--max_tgt_len 1024 \
--data_path ../data/pandagpt4_visual_instruction_data.json \
--image_root_path ../data/images/ \
--save_path ./ckpt/train_mvtec/ \
--log_path ./ckpt/train_mvtec/log_rest/
```

In the shell file, we can find many arguments:

- Main python script "["train_mvtec.py"](#)".
- The model which will be used "["--model openllama_peft"](#)".
- The multiple [checkpoints](#) such as **Vicuna** (LLM), **ImageBind** (Multimodal Encoder) and **PandaGPT** (LLM, used as text encoder).
- The paths for the textual data ("["--data_path"](#)") and images ("["--image_root_path"](#)").
- The path where the resulting trained model will be stored is "["--save_path"](#)".
- The log path, where the logging information from the whole process will be stored is "["--log_path"](#)".
- The "["-include localhost:0"](#)" specifies the devices (GPUs) on the localhost to be included in the training. In this case, GPU 0 and, if it was written 0,1, then it would include 2 GPUs, 0 and 1.
- The command "["-master_port"](#)" determines the port used for communication between processes in distributed training.

The [goal](#) is to replace consciously [the decoder layer](#) used in the baseline model to, not only attempt to improve the model's performance but also to learn from the architecture, the libraries used, and to apply a multinode training using DeepSpeed and Horovod. Therefore, in order to do so, we **need to understand the training process and proceed chronologically** according to the baseline code, which first leads to understanding the python script "train_mvtec.py".

5.2.2 Principal Script: train_mvtec.py

The script first evaluates the last condition "[" __name__ == "__main__"](#):

```
if __name__ == "__main__":
    args = parser_args() # Calls the function parser_args() to parse
                         # the command shell arguments from the shell script
    args = vars(args)   # Converts into a dictionary all the previous retrieved
                        # arguments from parser_args()
    args['layers'] = [7,15,23,31] # Adds new key-value pair to the dictionary
    main(**args)         # Calls main function of the script
```

5.2.2.1 Function parser_args()

The function is in charge of collecting all the arguments used in the shell file commands.

```
def parser_args():
    parser = argparse.ArgumentParser(description='train parameters')
    parser.add_argument('--model', type=str)
    parser.add_argument('--local_rank', default=0, type=int)
    parser.add_argument('--save_path', type=str)
    parser.add_argument('--log_path', type=str)

    # model configurations
    parser.add_argument('--imagebind_ckpt_path', type=str) # the path that stores the imagebind checkpoint
    parser.add_argument('--vicuna_ckpt_path', type=str) # the path that stores the vicuna checkpoint
    parser.add_argument('--delta_ckpt_path', type=str) # the delta parameters trained in stage 1
    parser.add_argument('--max_tgt_len', type=int) # the maximum sequence length
    parser.add_argument('--stage', type=int) # the maximum sequence length
    parser.add_argument('--data_path', type=str) # the maximum sequence length
    parser.add_argument('--image_root_path', type=str) # the maximum sequence length
```

The resulting variable args will contain the following information (further information is provided within the files `base.yaml`, `openllama_peft.yaml` and `openllama_peft_stage_1.json`):

```
args = {'model': openllama_peft, 'local_rank': 0, 'save_path': './ckpt/train_mvtec/', 'log_path':
    './ckpt/train_mvtec/log_rest/', 'imagebind_ckpt_path': '../pretrained_ckpt/imagebind_ckpt/imagebind_huge.pth',
    'vicuna_ckpt_path': '../pretrained_ckpt/vicuna_ckpt/7b_v0/', 'delta_ckpt_path':
    '../pretrained_ckpt/pandagpt_ckpt/7b/pytorch_model.pt', 'max_tgt_len': 1024, 'stage': 1, 'data_path':
    '../data/pandagpt4_visual_instruction_data.json', 'image_root_path': '../data/images/', 'layers': [7, 15, 23, 31],
    'root_dir': '../', 'mode': train, 'models': {'openllama': {'model_name': OpenLLAMAModel,
    'agent_name': DeepSpeedAgent, 'stage1_train_dataset': MVTEcDataset, 'test_dataset': SelfInstructTestDataset},
    'openllama_peft': {'model_name': OpenLLAMAPEFTModel, 'agent_name': DeepSpeedAgent, 'stage1_train_dataset':
    MVTEcDataset, 'test_dataset': SelfInstructTestDataset}}, 'logging_step': 5, 'max_len': 512, 'penalty_alpha': 0.6,
    'top_k': 10, 'top_p': 0.7, 'random_prefix_len': 5, 'sample_num': 2, 'decoding_method': sampling,
    'generate_len': 512, 'lora_r': 32, 'lora_alpha': 32, 'lora_dropout': 0.1, 'seed': 42, 'warmup_rate': 0.1,
    'epochs': 50, 'max_length': 1024, 'master_ip': '127.0.0.1', 'master_port': '28400',
    'world_size': 1, 'ds_config_path': dsconfig/openllama_peft_stage_1.json,
    'dschf': <transformers.deepspeed.HfDeepSpeedConfig object at 0x7b085ad70040>}
```

5.2.2.2 Function config_env(args)

Now that all the arguments needed are already stored in the variable "args", the model can start to be loaded, in terms of [configurations for distributed machine learning training \(deepspeed\)](#) as well as the training data which will be used from the [MVTec Dataset](#).

Therefore, let's first introduce the function config_env, which will be in charge of loading, updating and initializing the training for the model with the corresponding configuration information.

```

def config_env(args):
    args['root_dir'] = '../' # Sets the root directory
    args['mode'] = 'train' # Sets the mode
    config = load_config(args) # Loads input configuration
    # The following picture illustrates the flow of information when the function
    # load_config(args) is called
    args.update(config) # Updates the args variable including the loaded configuration
    initialize_distributed(args) # Initializes the distributed training environment
    # This function is subsequently explained as well
    set_random_seed(args['seed']) # Sets random seed

```

The figure consists of three side-by-side screenshots of a GitHub repository interface. The leftmost screenshot shows the main Python file 'config_env.py' with several annotations: a red circle highlights the call to 'load_config(args)' at line 19, which has a red arrow pointing to the middle screenshot; another red circle highlights the assignment 'base_configuration = load_base_config()' at line 22, which has a red arrow pointing to the rightmost screenshot; and a third red circle highlights the assignment 'configuration = load_model_config(args['model'], args['mode'])' at line 23, which has a blue arrow pointing to the middle screenshot. The middle screenshot shows the 'base.yaml' configuration file with annotations: a red circle highlights the 'models: openllama:' section at line 5, which has a blue arrow pointing to the rightmost screenshot; another red circle highlights 'stage1_train_dataset: MTeCDataset' at line 15, and a third red circle highlights 'test_dataset: SciInstructsTestDataset' at line 16, both of which have blue arrows pointing to the rightmost screenshot. The rightmost screenshot shows the 'openllama_peft.yaml' configuration file with annotations: a red circle highlights the 'models: openllama:' section at line 1, which has a blue arrow pointing from the middle screenshot; another red circle highlights '# lora hyper-parameters' at line 11, and a third red circle highlights 'train: seed: 42' at line 17, both of which have blue arrows pointing from the middle screenshot.

5.2.2.3 Function initialize_distributed(args)

Let's introduce some terms to understand upcoming variables easier:

- **Micro-batch size per GPU:** Number of samples processed by each GPU in a single forward and backward pass.
- **Training batch size:** This is the total batch size that the model uses in each training iteration. It consists of the sum of the micro-batch sizes across all GPUs.
- **world_size:** Total number of processes (or GPUs) participating in the distributed training.
- **node:** A node is a single computer in a larger network, often used in parallel computing and clusters. It typically consists of the following components: CPU, GPU, Memory (RAM) and Storage.

For instance, if the system uses 2 nodes with 4 GPUs each, and we consider the context of Pytorch's distributed training, the command line `torch.distributed.get_world_size()` would return 8. Moreover, if the system considers a micro-batch size of 32, the effective training batch size seen by the model is $8 \times 32 = 256$.

Consequently, in order to initialize properly all these configurations within the distributed training, the following function is needed:

```

def initialize_distributed(args):
    args['master_ip'] = os.getenv('MASTER_ADDR', 'localhost') # Sets the master IP
    args['master_port'] = os.getenv('MASTER_PORT', '6000') # Sets the master port
    args['world_size'] = int(os.getenv('WORLD_SIZE', '1')) # Sets the world_size
    args['local_rank'] = int(os.getenv('RANK', '0')) % torch.cuda.device_count() # Counts the local rank
    device = args['local_rank'] % torch.cuda.device_count() #
    torch.cuda.set_device(device) # Sets the local rank

    # Initialization of DeepSpeed for distributed training process with NCCL backend
    deepspeed.init_distributed(dist_backend='nccl')

```

5.2.2.4 Function Main: Part 1

Being aware of the previous definitions, the function main which puts into place all needed is defined as follows:

```

def main(**args):
    config_env(args) # Previously explained

    # The following picture contains the information of the file openllama_peft_stage_1.json
    args['ds_config_path'] = f'dsconfig/{args["model"]}_stage_{args["stage"]}.json'
    dschf = HfDeepSpeedConfig(args['ds_config_path'])
    args['dschf'] = dschf

    build_directory(args['save_path']) # Builds directories which ensure that
    build_directory(args['log_path']) # the save and log directories exist

    # Configures logging to store logs in the specified log path
    if args['log_path']:
        logging.basicConfig(
            format='%(asctime)s - %(pathname)s[line:%(lineno)d] - %(levelname)s: %(message)s',
            level=logging.DEBUG,
            filename=f'{args["log_path"]}/train_{time.asctime()}.log',
            filemode='w'
        )

    # Loading training data and training supervised fine-tuning data
    train_data, train_iter, sampler = load_mvtec_dataset(args, None)
    train_data_sft, train_iter_sft, sampler = load_sft_dataset(args, None)

    #
    length = args['epochs'] * len(train_data) // args['world_size']
    # dschf.config['train_micro_batch_size_per_gpu']

    # Total training steps considered and added to the variable args
    total_steps = 2 * args['epochs'] * len(train_data) // dschf.config['train_batch_size']
    args['total_steps'] = total_steps

    # Calling and loading the agent DeepSpeed with all the arguments
    agent = load_model(args)

    # Ensuring that all processes wait for each other to reach a certain code
    torch.distributed.barrier()

    ---- BREAK MAIN FUNCTION ---

```

The functions `load_mvtec_dataset` and `load_sft_dataset` will be explained in the upcoming section, although, as expected both functions are responsible for preparing the datasets for training in a distributed setup. The functions ensure that the data is correctly loaded, shuffled and distributed across multiple processes for efficient training using DeepSpeed. Both functions are contained in the folder "datasets" within the script `__init__.py`.

The file openllama_peft_stage_1.json contains the information specific to the model and stage that will be loaded into DeepSpeed configuration and added subsequently to the variable "args".

```

anomalygpt > 1AnomalyGPT > code > dconfig > openllama_peft_stage_1.json > {} activation_checkpointing
 1  {
 2    "train_batch_size": 8,
 3    "train_micro_batch_size_per_gpu": 1,
 4    "gradient_accumulation_steps": 8,
 5    "steps_per_print": 1,
 6    "gradient_clipping": 1.0,
 7    "zero_optimization": {
 8      "stage": 2,
 9      "offload_optimizer": {
10        "device": "cpu"
11      },
12      "contiguous_gradients": true,
13      "allgather_bucket_size": 50000,
14      "reduce_bucket_size": 50000,
15      "allgather_partitions": true
16    },
17    "fp16": {
18      "enabled": true,
19      "opt_level": "O2",
20      "initial_scale_power": 12,
21      "loss_scale_window": 1000,
22      "min_loss_scale": 1
23    },
24    "bf16": {
25      "enable": true
26    },
27    "optimizer": {
28      "type": "Adam",
29      "params": {
30        "lr": 0.0001,
31        "betas": [
32          0.9,
33          0.95
34        ],
35        "eps": 1e-8,
36        "weight_decay": 0.001
37      }
38    },
39    "scheduler": {
40      "type": "WarmupDecayLR",
41      "params": {
42        "warmup_min_lr": 0,
43        "warmup_max_lr": 0.001,
44        "warmup_num_steps": 10,
45        "total_num_steps": 10000
46      }
47    },
48    "activation_checkpointing": []
49    "partition_activations": true,
50    "cpu_checkpointing": true,
51    "contiguous_memory_optimization": false,
52    "number_checkpoints": null,
53    "synchronize_checkpoint_boundary": false,
54    "profile": false
55  }
56
57
58

```

5.2.2.5 Functions load_mvtec_dataset & load_sft_dataset

Some modifications had to be performed on these 2 functions in order to speed up the training process while modifying the decoder layer of the model. This way, by decreasing the samples which are used in the training process, instead of lasting almost a complete day the whole training process, it finishes in a matter of minutes. The main modification is the inclusion of a new object class denoted as [CustomSubset](#), which is in charge of limiting the number of training samples used in the training process.

This [new class CustomSubset](#) inherits from the PyTorch Subset class. The [Subset class is used to create a subset of a dataset](#) based on specified indices.

```

CustomSubset(Subset):
    def __init__(self, dataset, indices): # Constructor method takes two arguments
        # Calling the constructor of the parent class Subset to initialize
        # the subset with the given dataset and indices.
        super(CustomSubset, self).__init__(dataset, indices)

        # This assigns the collate function of the original dataset to an instance variable self.collate.
        # The collate function is used to merge a list of samples to form a mini-batch of Tensor(s).
        self.collate = dataset.collate

    def collate(self, batch):
        return self.dataset.collate(batch)

```

Including that new class, allows to limit the total number of samples considered by including in [load_mvtec_dataset](#) the argument [limit_samples](#) and its corresponding flow control 'if' within the proper function. The same modifications are applied to the function [load_sft_dataset](#), even though do not appear in this document.

```

""" NEW load_mvtec_dataset with limited samples """
def load_mvtec_dataset(args, limit_samples=None):
    # Instantiating this class defines various parameters for different objects/textures
    data = MVtecDataset('../data/mvtec_anomaly_detection')

    # Including flow control to limit samples
    if limit_samples is not None:
        indices = list(range(len(data)))
        subset_indices = indices[:limit_samples]
        data = CustomSubset(data, subset_indices)

```

```

sampler = torch.utils.data.RandomSampler(data) # Samples elements randomly
world_size = torch.distributed.get_world_size()

# Unique identifier assigned to each process in the distributed setting
# It helps dividing data among processes, managing parameter updates
# and coordinating the activities of different processes
rank = torch.distributed.get_rank()
batch_size = args['world_size'] * args['dschf'].config['train_micro_batch_size_per_gpu']

# Ensures that each process in the distributed setting gets a unique subset of the data to work on
batch_sampler = DistributedBatchSampler(
    sampler,
    batch_size,
    drop_last=True, # Ensures last incomplete batch is dropped
    rank,
    world_size)

# Combines the dataset and sampler, providing an iterable over the dataset
# It handles batching, shuffling and loading the data in parallel using multiple workers
iter_ = DataLoader(
    data,
    batch_sampler=batch_sampler,
    num_workers=8,
    collate_fn=data.collate,
    pin_memory=False)

return data, iter_, sampler

```

5.2.2.6 Function load_model

The function `load_model`, located in the script `__init__.py` within the `model` folder, loads the model as its name infers and instantiates the 2 most important classes for the training process, the `DeepSpeed Agent` (responsible for the distributed training) and the `model OpenLLAMAPEFTModel` which will be used.

```

def load_model(args):
    # Extracts the name of the agent (DeepSpeed - Agent)
    agent_name = args['models'][args['model']]['agent_name']

    # Extracts the name of the model (OpenLLAMAPEFTModel - Model)
    model_name = args['models'][args['model']]['model_name']

    # globals() is a global namespace where all global variables and
    # symbols for the current program are stored in a dictionary.
    # In essence, it retrieves the agent and model class and instantiates
    # them with model and args to its constructors
    model = globals()[model_name](**args)
    agent = globals()[agent_name](model, args)
    return agent

```

5.2.2.7 Class OpenLLAMAPEFTModel

The `class OpenLLAMAPEFTModel` is the `heart of the model`. Located in the script `openllama.py` within the folder `model`, it contains the loadings of all the different models that constitute AnomalyGPT.

For the basic understanding of the class, let's look at the constructor method and explain the loaded models, which includes `ImageBind` model as the `visual_encoder`, the `image_decoder` (LinearLayer Class), the `prompt_learner` (PromptLearner Class) and the previously mentioned `focal` and `dice loss functions`.

```

def __init__(self, **args):
    super(OpenLLAMAPEFTModel, self).__init__()
    self.args = args

    # Checkpoints (configurations/model) path for ImageBind and Vicuna models
    imagebind_ckpt_path = args['imagebind_ckpt_path']
    vicuna_ckpt_path = args['vicuna_ckpt_path']
    max_tgt_len = args['max_tgt_len']
    stage = args['stage'] # Stage 1

    print(f'Initializing visual encoder from {imagebind_ckpt_path} ...')

    # Instantiating ImageBind with args and its checkpoints
    self.visual_encoder, self.visual_hidden_size = imagebind_model.imagebind_huge(args)
    imagebind_ckpt = torch.load(imagebind_ckpt_path, map_location=torch.device('cpu'))
    self.visual_encoder.load_state_dict(imagebind_ckpt, strict=True)
    self.iter = 0

    # Instantiating the Image Decoder
    self.image_decoder = LinearLayer(1280, 1024, 4)

    # Instantiating the Prompt Learner
    self.prompt_learner = PromptLearner(1, 4096)

    # Instantiating both loss functions
    self.loss_focal = FocalLoss()
    self.loss_dice = BinaryDiceLoss()

    # Free vision encoder and freezing the parameters
    for name, param in self.visual_encoder.named_parameters():
        param.requires_grad = False
    self.visual_encoder.eval()
    print('Visual encoder initialized.')
    print(f'Initializing language decoder from {vicuna_ckpt_path} ...')

    # Add the LoRA module - Mentioned in Extra-Information
    peft_config = LoraConfig(
        task_type=TaskType.CAUSAL_LM, # Task for which the model is fine-tuned - causal language model modality
        inference_mode=False, # If False then it's in Training mode, allows update of parameters
        r=self.args['lora_r'], # Rank of LoRA matrices, the higher the more expressive, but also more expensive
        lora_alpha=self.args['lora_alpha'], # Scale factor applied to LoRA updates
        lora_dropout=self.args['lora_dropout'], # Dropout rate applied to LoRA layers
        # List specifying the target modules in the model where LoRA should be applied (Attention Mechanism)
        target_modules=['q_proj', 'k_proj', 'v_proj', 'o_proj'])

    # Loading the pre-trained Llama casual language model from the Vicuna checkpoints
    self.llama_model = LlamaForCausalLM.from_pretrained(vicuna_ckpt_path)

    # PEFT explained in a section from the document
    self.llama_model = get_peft_model(self.llama_model, peft_config)
    self.llama_model.print_trainable_parameters()

    # Loading the tokenizer module associated to the Llama Model
    self.llama_tokenizer = LlamaTokenizer.from_pretrained(vicuna_ckpt_path, use_fast=False)
    self.llama_tokenizer.pad_token = self.llama_tokenizer.eos_token
    self.llama_tokenizer.padding_side = "right"
    print('Language decoder initialized.')

    # Initializes a linear projection layer to match the dimensions
    # between the visual features and the language model
    self.llama_proj = nn.Linear(self.visual_hidden_size, self.llama_model.config.hidden_size)

    self.max_tgt_len = max_tgt_len
    self.device = torch.cuda.current_device()

```

From this particular class and as we are interested on modifying the Decoder Layer, we will focus on the instantiation of the `LLama Model` (`self.llama_model`) using the `class LlamaForCausalLM` and the `Llama Tokenizer` (`self.llama_tokenizer`). The term "causal" comes from the concept of causality in time series analysis, where future values are predicted based on past values. Similarly, [in causal language models, the current token prediction depends only on the previous tokens](#), not on future tokens. Particularly, this class will lead us to the part of the code that was modified.

Moreover, the class includes as well the initialization of the `tokenizer` from the Llama model. The tokenizer is responsible for `converting input text into tokens` that the model can process and for `decoding the model's token output back` into human-readable text.

Following the arrows from the next picture, we reach the modification performed on the Class MLP, in which it's added a dropout layer. Originally, we considered 2 possible modifications, adding a normalization and dropout layer at the same time, however, when executing the code for training the gradient became too small and vanished at some point. Therefore, and by testing, we removed the normalization layer and only introduced the dropout layer, leading to successful complete trainings.

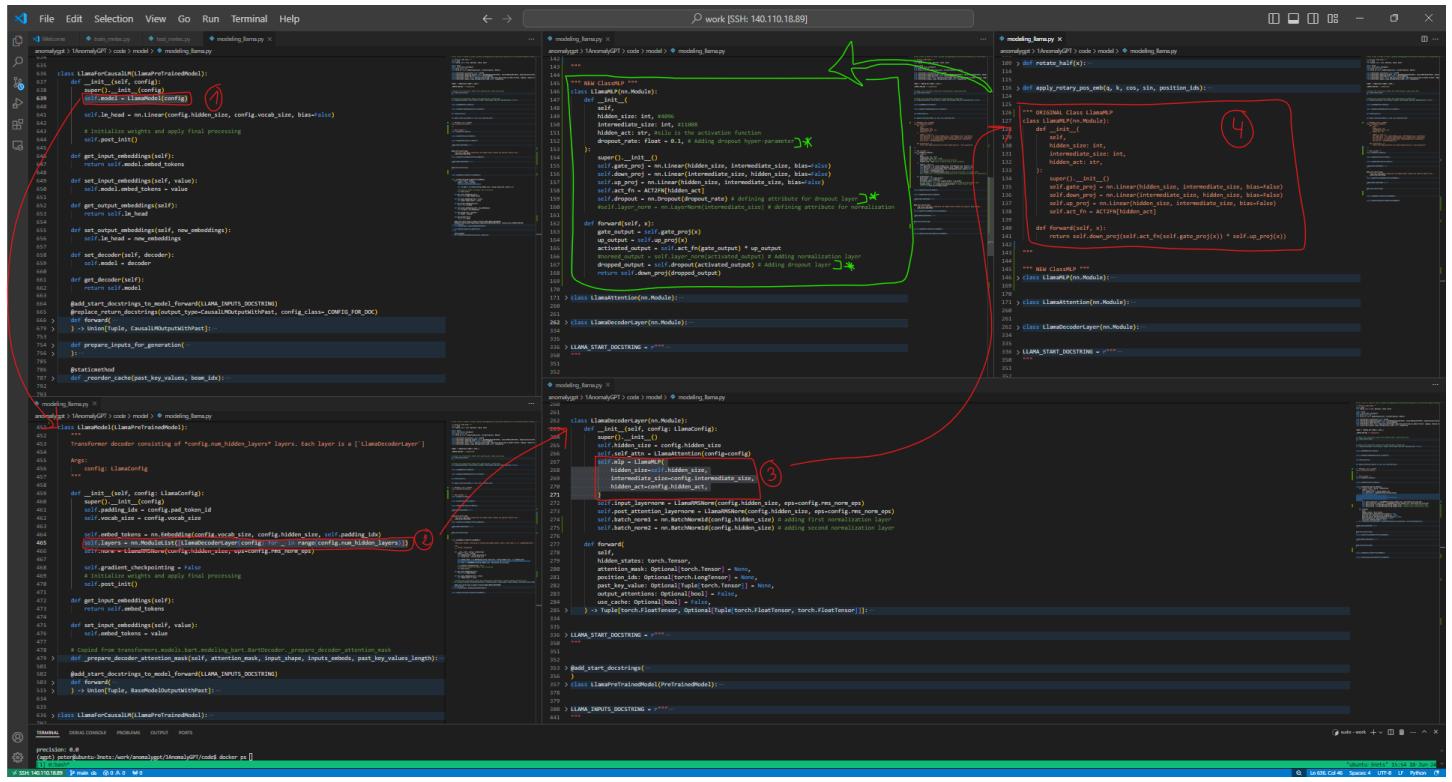


Figure 1: Flow of Modifications

Besides, in the class **DecoderLayer**, we also applied more modifications by introducing `batch normalization` to make sure that the vanishing gradient problem doesn't return and changed some `hyper-parameters` (fp16 and learning rate).

```

123 class LlamaLayerNorm(nn.Module):
124     def __init__(self, config: LlamaConfig):
125         super().__init__()
126         self.hidden_size = config.hidden_size
127         self.rms_norm = nn.RMSNorm(config=config)
128         self.alp = LlamaAlp(
129             hidden_size=self.hidden_size,
130             intermediate_size=config.intermediate_size,
131             hidden_act=config.hidden_act,
132         )
133
134         self.input_layernorm = LlamaLayerNorm(config.hidden_size, eps=config.rms_norm_eps)
135         self.attention_layer = LlamaAttention(config.hidden_size, config.attention_head_size, config)
136         self.norm1 = nn.LayerNorm(config.hidden_size)
137         self.norm2 = nn.LayerNorm(config.hidden_size) # adding second normalization layer
138         self.batch_norm2 = nn.BatchNorm1d(config.hidden_size)
139
140     def forward(self, hidden_states, **kwargs):
141         self_attention_outputs = self.alp(hidden_states)
142         hidden_states = self.input_layernorm(hidden_states)
143
144         # Self Attention
145         hidden_states, self_attn_weights, present_key_value = self.self_attn(
146             hidden_states, self_attn_weights, present_key_value=present_key_value,
147             attention_mask=attention_mask,
148             position_ids=position_ids,
149             past_key_value=past_key_value,
150             output_attentions=output_attentions,
151             use_cache=use_cache,
152             past_key_value=past_key_value,
153             use_cache=use_cache,
154         )
155
156         hidden_states = residual + hidden_states
157
158         # Fully Connected
159         residual = hidden_states
160         hidden_states = self.norm1(hidden_states)
161         hidden_states = residual + hidden_states
162         hidden_states = self.norm2(hidden_states)
163         hidden_states = residual + hidden_states
164         hidden_states = self.batch_norm2(hidden_states)
165
166         outputs = (hidden_states, self_attn_weights)
167
168         if use_cache:
169             outputs += (present_key_value,)
170
171         return outputs

```

Figure 2: Batch Normalization

```

1 {
2     "train_batch_size": 8,
3     "train_micro_batch_size_per_gpu": 1,
4     "gradient_accumulation_steps": 8,
5     "steps_per_print": 1,
6     "warmup_ratio": 1.0,
7     "zero_optimization": {
8         "stage": 2,
9         "offload_optimizer": {
10             "device": "cpu"
11         },
12         "contiguous_gradients": true,
13         "allgather_bucket_size": 50000,
14         "reduce_bucket_size": 50000,
15         "allgather_partitions": true
16     },
17     "fp16": [
18         "enabled": true,
19         "initial_scale_power": 12,
20         "loss_scale_window": 1000,
21         "min_loss_scale": 1
22     ],
23     "bf16": {
24         "enabled": true
25     },
26     "optimization": {
27         "type": "Adam",
28         "params": {
29             "lr": 0.0001,
30             "beta1": 0.9,
31             "beta2": 0.99
32         },
33         "eps": 1e-8,
34         "weight_decay": 0.001
35     },
36     "schedules": {
37         "type": "WarmupDecayLR",
38         "params": {
39             "warmup_min_lr": 0,
40             "warmup_max_lr": 0.0001,
41             "warmup_num_steps": 10,
42             "total_num_steps": 10000
43         }
44     },
45     "activation_checkpointing": {
46         "partition_activations": true,
47         "cpu_checkpointing": true,
48         "context_synchronization": false,
49         "memory_checkpointing": true,
50         "synchronize_checkpoint_boundary": false,
51         "profile": false
52     }
53 }

```

Figure 3: FP16 + Learning Rate Modification

5.2.2.8 Function Main: Part 2

This is the continuation of the remaining code from the **main** function. The most important mentioned function is **train_model**, which will be explained in the next section, although it's also called the function **save_model** which is equally important.

In the last part of the function, the training process is executed taking into account the epochs and iterations update, the model parameters saving and finally plotting the evolution of the loss at the end of the whole process.

```
# Beginning of training
pbar = tqdm(total= 2 * length) # Maximum total number / Progress bar
current_step = 0 # To keep track of the current training step

# Main Loop consisting of a total of 50 epochs
for epoch_i in tqdm(range(args['epochs'])):
    iter_every_epoch = 0 # To keep track of iterations within each loop
    for batch, batch_sft in zip(train_iter,train_iter_sft):
        iter_every_epoch += 1

        # First training step in which is called the train_model method
        # to perform a training step with batch samples
        agent.train_model(
            batch,
            current_step=current_step,
            pbar=pbar)
        del batch # To free memory

        # Second training step but with batch_sft
        agent.train_model(
            batch_sft,
            current_step=current_step,
            pbar=pbar)
        del batch_sft # To free memory
        current_step += 1
    # Ensuring all processes synchronize at the end of each epoch
    torch.distributed.barrier()
    # Save at the end of the training
    agent.save_model(args['save_path'], 0)
# Plot loss after training
agent.plot_loss()
```

5.2.2.9 Function train_model

The `train_model` function performs the forward pass, the backward pass, the parameter update and logging of relevant information. Furthermore, it has been modified from the original version to save the loss values and plot the evolution of it at the end of the training.

```
def train_model(self, batch, current_step=0, pbar=None):
    self.ds_engine.module.train() # Puts the model in training mode

    # Forward pass & Loss Calculation
    # Performs the forward pass with the input "batch" using the DeepSpeed engine
    # and computes the loss from the model and the maximum likelihood estimation accuracy
    loss, mle_acc = self.ds_engine(batch)

    # Backward pass
    # Computes the gradients of the loss with respect
    # to the model parameters using backpropagation
    self.ds_engine.backward(loss)

    # Parameter Update of the DeepSpeed engine
    self.ds_engine.step()

    # Store loss value to plot it afterwards
    self.loss_list.append(loss.item())

    # Progress Bar Update
    # It updates the description of the progress bar with
    # the current loss and token accuracy and advances by
    # one step the pbar
    pbar.set_description(f'[{!}] loss: {round(loss.item(), 4)}; token_acc: {round(mle_acc*100, 2)}')
    pbar.update(1)

    # Calculates the elapsed time, rate of progress,
    # the estimated remaining time and logs the information
    if self.args['local_rank'] == 0 and self.args['log_path'] and current_step % self.args['logging_step'] == 0:
        elapsed = pbar.format_dict['elapsed']
        rate = pbar.format_dict['rate']
        remaining = (pbar.total - pbar.n) / rate if rate and pbar.total else 0
        remaining = str(datetime.timedelta(seconds=remaining))
        logging.info(f'[{!}] progress: {round(pbar.n/pbar.total, 5)}';
                    remaining_time: {remaining}; loss: {round(loss.item(), 4)};
                    token_acc: {round(mle_acc*100, 2)}')
    )

    # Adjusts the accuracy metric for returning
    mle_acc *= 100
    return mle_acc
```

5.3 Model Versions & Modifications

There has been conducted multiple different model trainings with the previously mentioned modifications, in different GPU nodes (██████████), with added datasets, and to be more specific, the exact configurations for each model are displayed below:

Config	V1	V2	V3	V4	V5	V6
Port	89	89	91	91	91	89 & 91
Epochs	20	50	50	50	50	50
Learning Rate	0.001	0.0001	0.0001	0.0001	0.0001	0.0001
Samples	10k	Complete	Complete	Complete	Complete	Complete
Batch Size	8	8	16	16	16	32
Micro Batch Size(per GPU)	1	1	2	2	2	2
Total GPUs	1	1	1	1	1	2
Gradient Accum. Steps	8	8	8	8	8	8
Docker Container	✗	✗	✗	✗	✗	✓
AeBAD Dataset	✗	✗	✗	✓	✓	✓
Data Aug.	✗	✗	✗	✗	✓	✓
Dropout Layer	✓	✓	✓	✓	✓	✓
Normalization Layer	✓	✗	✗	✗	✗	✗
Residual Layer	✓	✓	✓	✓	✓	✓
Batch Normalization	✗	✗	✓	✓	✓	✓
FP16 Initial Scale Power	✗	12	12	12	12	12
FP16 Loss Scale Window	✗	1000	1000	1000	1000	1000
Training Process	17%	100%	100%	100%	100%	100%

Table 1: Comparison of Model Versions

5.3.1 Motivations

5.3.1.1 Model V1

The Model V1 was the first model to be modified and used as practicing prototype. The modifications, as shown in the previous table, were not really profound however it allowed to understand better the flow of data and the pre-defined functions as well as classes. The modifications include the limitation of samples to accelerate the trial & error testing, the decrease in the total number of epochs, the introduction in the Class MLP of 2 new layers, first the normalization and then the dropout layer with a dropout rate of 0.1. The modifications aimed at:

1. **Normalization Layer:** Included

- **Reason:** Layer normalizes the inputs across the features to stabilize and accelerate training.

```

    # Class MLP
    # Constructor: __init__
    self.layer_norm = nn.LayerNorm(intermediate_size)

    # Method: forward
    normed_output = self.layer_norm(activated_output)

```

2. Dropout Layer: Included

- **Reason:** Dropout is used to prevent overfitting by randomly setting a fraction of input units to zero at each update during training and forcing the model to learn more robust features.

```

    # Class MLP
    # Constructor: __init__
    self.dropout = nn.Dropout(dropout_rate)

    # Method: forward
    dropped_output = self.dropout(normed_output)

```

However, in training execution and although the number of samples was quite small, the model stopped learning due to vanishing gradient issues leading to stop automatically the training process. To counter this problem, several modifications were considered and implemented in the upcoming models V2 and V3.

5.3.1.2 Model V2

The model V2 implemented new modifications and attempted to succeed where the model V1 failed. Among the modifications, and after trying with fewer epochs and samples to check if the vanishing gradient issue was solved, the main update addressed the normalization layer (which was removed) and considered additional hyper-parameters within the fp16 configuration located in [openllama_peft_stage_1.json](#).

1. Learning Rate: Decrease

- **Reason:** A lower learning rate than Model V1, allows for more fine-tuned adjustments during training, reducing the risk of overshooting the optimal parameters.

2. Normalization Layer: Not included

- **Reason:** Training might be slower and less stable without normalization layers although solved the vanishing gradient problem.

3. FP16: 16-Bit Floating Point

- **Reason:** Setting the initial scale power to 12 controls the initial scale factor for mixed-precision training. A scale power of 12 is commonly used starting point for FP16 training. Moreover, the loss scale window parameter, set to 1000, helps in dynamically adjusting the loss scaling factor to avoid overflow and underflow during FP16 training.

5.3.1.3 Model V3

The model V3 included the configurations found relevant in model V2 and considered an additional modification in terms of batch normalization.

1. Batch Normalization: Included

- **Reason:** By normalizing the inputs to each layer, batch normalization allows the network to converge faster, reducing the training time, helps in stabilizing the learning process leading to more reliable gradients

5.3.1.4 Model V4

The model V4 considers the same configuration modifications as the model V3 and adds 4 new classes to the MVTec Dataset. These new classes in fact represent the same object, a blade, although considers 4 typical anomalies that might make invalid its production. The dataset was found on a Github repository and its called [Aero-engine Blade Anomaly Detection](#) (AeBAD) dataset. The 4 categories for the blades are:

- **Ablation:** A blade without ablations should have a smooth, uniform surface with no areas of material removal or wear. The edges should be sharp and intact, with no signs of pitting, erosion, or thinning. The entire blade should appear consistent in texture and color, indicating it is free from any ablation-related damage.”
 - **Breakdown:** A blade without breakdowns should maintain its structural integrity with no signs of cracking, chipping, or flaking. The blade should exhibit a uniform appearance, with no visible separations, detachment of layers, or deformation. The blade should appear robust and solid, free from any structural compromise.”
 - **Fracture:** A blade without fractures should have a continuous, unbroken surface with no visible cracks or splits. The blade should be seamless and sturdy, with no signs of stress marks or propagation lines that indicate potential fracture points. The entire blade should look solid and reliable, free from any fracturing damage.”
 - **Groove:** A blade without grooves should have a completely smooth and even surface with no indentations or carved-out lines. The blade’s surface should be uniform and consistent, with no signs of unintended channels or depressions. The blade should appear flawlessly smooth, indicating it is free from any groove related imperfections.”

Moreover, for each type of anomaly there are as well, 4 different perspectives, including background, illumination, view and original/same changes. In order to include the new classes, multiple modifications were needed to be performed, starting from copy/pasting the original dataset into the well organized structure from the MVTec Dataset.

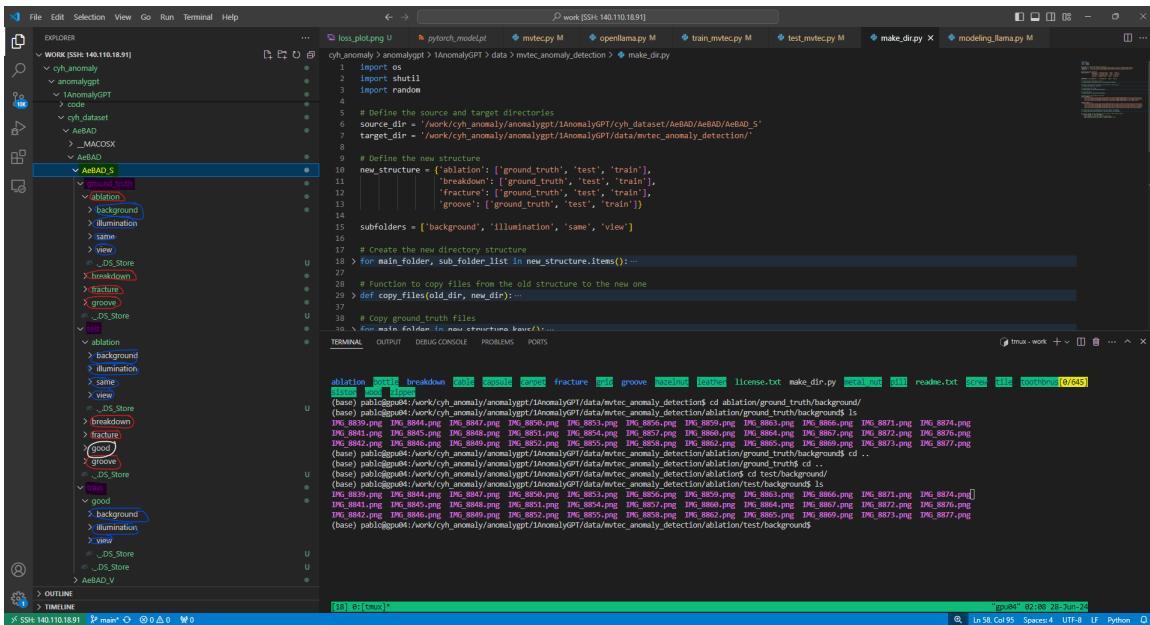


Figure 4: Original Structure AeBAD-Dataset

Therefore, I wrote down a script called `make_dir.py` that allowed me to pass from the AeBAD structure to the MVTec Dataset one, although one of the issues was that the training samples were mixed, meaning that you could find samples from background ablation, background breakdown, fracture or groove all in the same folder and not

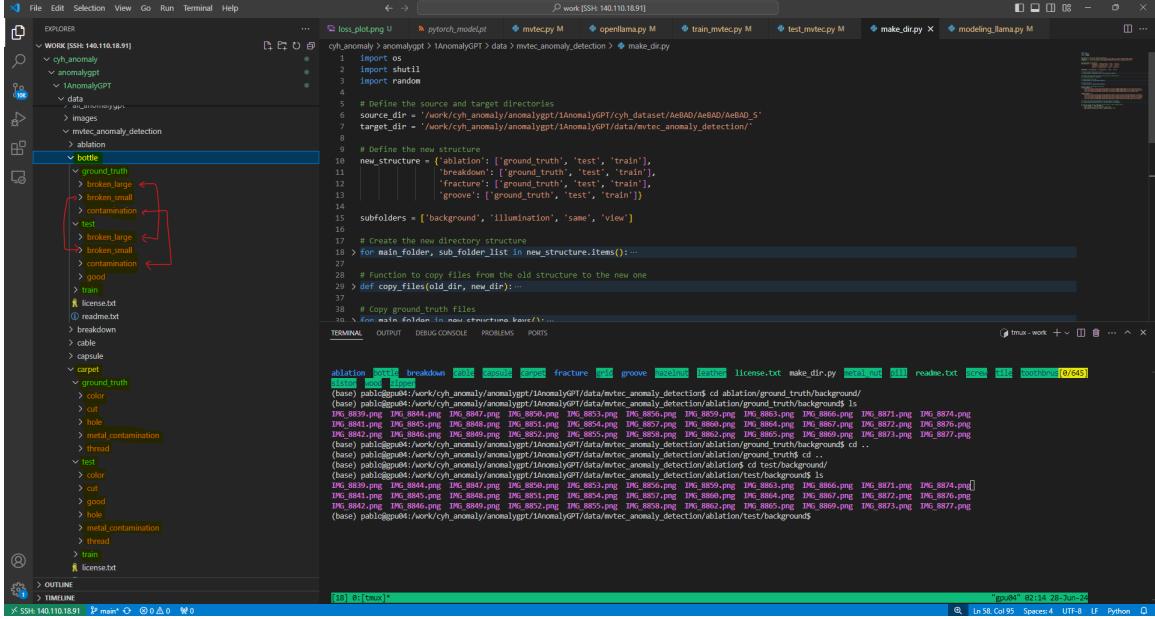


Figure 5: Original Structure MVTec-Dataset

separated as in MVTec Dataset. To solve such issue, and after carefully studying the pictures, I decided to create the train folder following a 70% split between the corresponding folders. For instance, for the folder train/good within ablation, it was included 70% of the train (good) samples from **background**, 70% of the train samples from **illumination** and 70% of the class **view**, addind up to 364 good training samples for each class.

```

# Define the source and target directories
source_dir = '/work/cyh_anomaly/anomalygpt/1AnomalyGPT/data/mvtc_anomaly_detection'
target_dir = '/work/cyh_anomaly/anomalygpt/1AnomalyGPT/data/mvtc_anomaly_detection'

# Define the new structure
new_structure = {
    'ablation': [
        {'breakdown': ['ground_truth', 'test', 'train']},
        {'fracture': ['ground_truth', 'test', 'train']},
        {'groove': ['ground_truth', 'test', 'train']}
    ],
    'bottle': [
        {'ground_truth': [
            'broken_large',
            'broken_small',
            'contamination'
        ]},
        {'test': [
            'broken_large',
            'broken_small',
            'contamination',
            'good'
        ]},
        {'train': [
            'background',
            'illumination',
            'same',
            'view'
        ]}
    ]
}

# Create the new directory structure
for main_folder, sub_folder_list in new_structure.items():
    for sub_folder in sub_folder_list:
        if main_folder == 'bottle':
            for category in sub_folder:
                os.makedirs(os.path.join(target_dir, main_folder, sub_folder, category), exist_ok=True)
        else:
            os.makedirs(os.path.join(target_dir, main_folder, sub_folder), exist_ok=True)

# Function to copy files from the old structure to the new one
def copy_files(old_dir, new_dir):
    for root, dirs, files in os.walk(old_dir):
        for file in files:
            src_file = os.path.join(root, file)
            dst_file = os.path.join(new_dir, os.path.relpath(src_file, old_dir))
            shutil.copy2(src_file, dst_file)

# Copy ground truth files
for main_folder in new_structure.keys():
    if main_folder == 'bottle':
        for category in new_structure[main_folder]:
            copy_files(os.path.join(source_dir, main_folder, category), os.path.join(target_dir, main_folder, category))
    else:
        copy_files(os.path.join(source_dir, main_folder), os.path.join(target_dir, main_folder))

# Copy test files
for main_folder in new_structure['bottle']['test']:
    copy_files(os.path.join(source_dir, 'test', main_folder), os.path.join(target_dir, 'test', main_folder))

# Define source and target directories
source_folders = [
    'work/cyh_anomaly/anomalygpt/1AnomalyGPT/cyh_dataset/AeBAD/AeBAD_S/train/good/background',
    'work/cyh_anomaly/anomalygpt/1AnomalyGPT/cyh_dataset/AeBAD/AeBAD_S/train/good/illumination',
    'work/cyh_anomaly/anomalygpt/1AnomalyGPT/cyh_dataset/AeBAD/AeBAD_S/train/good/view'
]

target_folders = [
    'work/cyh_anomaly/anomalygpt/1AnomalyGPT/data/mvtc_anomaly_detection/blade_ablation/train/good',
    'work/cyh_anomaly/anomalygpt/1AnomalyGPT/data/mvtc_anomaly_detection/blade_breakdown/train/good',
    'work/cyh_anomaly/anomalygpt/1AnomalyGPT/data/mvtc_anomaly_detection/blade_fracture/train/good',
    'work/cyh_anomaly/anomalygpt/1AnomalyGPT/data/mvtc_anomaly_detection/blade_groove/train/good'
]

# Function to copy a percentage of files from source to target
def copy_random_files(source_folder, target_folders, percentage):
    # Copy 70% of the files. From each source folder to each target folder
    for source_folder in source_folders:
        # Each Category train has the same random 70%
        copy_random_files(source_folder, target_folders, 0.7)

```

Figure 6: Script: make_dir.py

One of the issues with the new dataset is that some of the ground_truth pictures corresponding to a particular test image was absent, or vice versa, leading to multiple samples being alone and therefore not valid for the training. Besides the previous problem, many pictures had to be deleted because they couldn't be displayed and treated by the model, leading as well to delete their corresponding ground_truth/mask images.

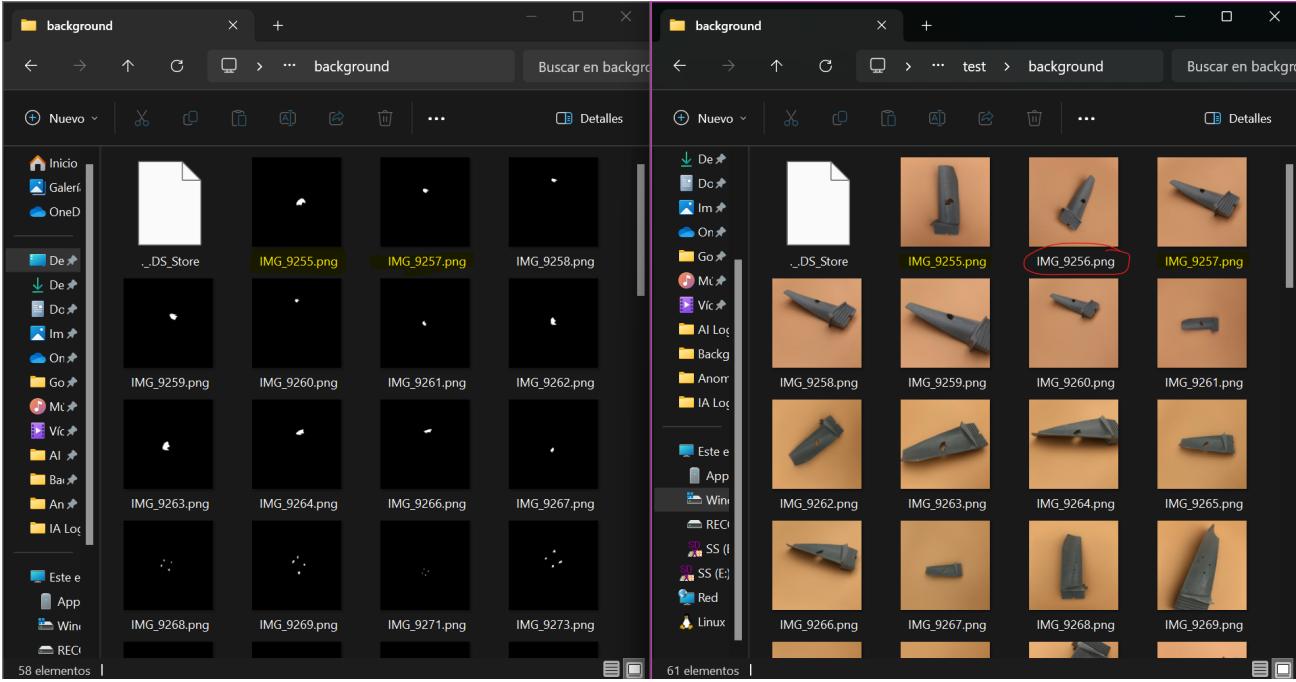


Figure 7: Correspondance of Images: Mask vs Test

Finally, the new included classes were structured as follows:

Figure 8: General Structure: AeBAD + MVTec Dataset

Then, in terms of script modifications, I had to modify a couple of things included in the scripts `openllama.py` (which can be found in the folder `model`) and `mvtec.py` (in folder `datasets`). Although there were very few modifications on `openllama.py`, the biggest majority happened on the script `mvtec.py`, including the variables `WIDTH_BOUNDS_PCT`, `NUM_PATCHES`, `INTENSITY_LOGISTIC_PARAMS`, `UNALIGNED_OBJECTS`, `BACKGROUND`, `OBJECTS`, the dictionary describles, and the method constructor of the class `MVtecDataset`.

```

... loss_plot.png U pytorch_model.pt M mvtec.py M openllama.py M train_mvtec.py M test_mvtec.py M make_dir.py modeling_llama.py

openllama.py anomalygt > 1AnomalyGPT > code > datasets > mvtec.py
1 import header *
2 import torch.mn.function as F
3 import cv2
4 import numpy as np
5 import torch
6 import torchvision.transforms as transforms
7 from PIL import Image, UnidentifiedImageError # UnidentifiedImageError to manage try/except block
8
9 from .self_sup_tasks import patch_ex
10
11 """ OLD
12 WIDTH_BOUNDS_PCT = {'bottle':((0.03, 0.4), (0.03, 0.4)), 'cable':((0.05, 0.4), (0.05, 0.4)), 'capsule':((0.03, 0.15), (0.03, 0.4)),
13     'hazelnut':((0.03, 0.35), (0.03, 0.35)), 'metal_nut':((0.03, 0.4), (0.03, 0.4)), 'pill':((0.03, 0.2), (0.03, 0.4)),
14     'screw':((0.03, 0.12), (0.03, 0.12)), 'toothbrush':((0.03, 0.4), (0.03, 0.2)), 'transistor':((0.03, 0.4), (0.03, 0.4)),
15     'zipper':((0.03, 0.4), (0.03, 0.2)),
16     'carpet':((0.03, 0.4), (0.03, 0.4)), 'grid':((0.03, 0.4), (0.03, 0.4)),
17     'leather':((0.03, 0.4), (0.03, 0.4)), 'tile':((0.03, 0.4), (0.03, 0.4)), 'wood':((0.03, 0.4), (0.03, 0.4)))
18 """
19
20 # Specifies the acceptable width bounds for objects in percentage terms relative to the image size
21 WIDTH_BOUNDS_PCT = {'bottle':((0.03, 0.4), (0.03, 0.4)), 'cable':((0.05, 0.4), (0.05, 0.4)), 'capsule':((0.03, 0.15), (0.03, 0.4)),
22     'hazelnut':((0.03, 0.35), (0.03, 0.35)), 'metal_nut':((0.03, 0.4), (0.03, 0.4)), 'pill':((0.03, 0.2), (0.03, 0.4)),
23     'screw':((0.03, 0.12), (0.03, 0.12)), 'toothbrush':((0.03, 0.4), (0.03, 0.2)), 'transistor':((0.03, 0.4), (0.03, 0.4)),
24     'zipper':((0.03, 0.4), (0.03, 0.2)),
25     'carpet':((0.03, 0.4), (0.03, 0.4)), 'grid':((0.03, 0.4), (0.03, 0.4)),
26     'leather':((0.03, 0.4), (0.03, 0.4)), 'tile':((0.03, 0.4), (0.03, 0.4)), 'wood':((0.03, 0.4), (0.03, 0.4)),
27     "ablation":((0.03, 0.4), (0.03, 0.4)), "breakdown":((0.03, 0.4), (0.03, 0.4)), "fracture":((0.03, 0.4), (0.03, 0.4)), "groove":((0.03, 0.4), (0.03, 0.4)) }
28
29 """ OLD
30 # Number of patches/segments to be extracted from each image for a given category
31 NUM_PATCHES = {'bottle':3, 'cable':3, 'capsule':3, 'hazelnut':3, 'metal_nut':3,
32     'pill':3, 'screw':4, 'toothbrush':3, 'transistor':3, 'zipper':4,
33     'carpet':4, 'grid':4, 'leather':4, 'tile':4, 'wood':4}
34 """
35
36 NUM_PATCHES = {'bottle':3, 'cable':3, 'capsule':3, 'hazelnut':3, 'metal_nut':3,
37     'pill':3, 'screw':4, 'toothbrush':3, 'transistor':3, 'zipper':4,
38     'carpet':4, 'grid':4, 'leather':4, 'tile':4, 'wood':4,
39     "ablation": 4, "breakdown": 4, "fracture": 4, "groove": 4}
40

```

Figure 9: Modifications Script: openllama.py

The modifications are shown below:

```

... loss_plot.png U pytorch_model.pt M mvtec.py M openllama.py M train_mvtec.py M test_mvtec.py M make_dir.py modeling_llama.py

openllama.py anomalygt > 1AnomalyGPT > code > datasets > mvtec.py
1 import os
2 from torch.utils.data import Dataset
3 import cv2
4 import numpy as np
5 import torch
6 import torchvision.transforms as transforms
7 from PIL import Image, UnidentifiedImageError # UnidentifiedImageError to manage try/except block
8
9 from .self_sup_tasks import patch_ex
10
11 """ OLD
12 WIDTH_BOUNDS_PCT = {'bottle':((0.03, 0.4), (0.03, 0.4)), 'cable':((0.05, 0.4), (0.05, 0.4)), 'capsule':((0.03, 0.15), (0.03, 0.4)),
13     'hazelnut':((0.03, 0.35), (0.03, 0.35)), 'metal_nut':((0.03, 0.4), (0.03, 0.4)), 'pill':((0.03, 0.2), (0.03, 0.4)),
14     'screw':((0.03, 0.12), (0.03, 0.12)), 'toothbrush':((0.03, 0.4), (0.03, 0.2)), 'transistor':((0.03, 0.4), (0.03, 0.4)),
15     'zipper':((0.03, 0.4), (0.03, 0.2)),
16     'carpet':((0.03, 0.4), (0.03, 0.4)), 'grid':((0.03, 0.4), (0.03, 0.4)),
17     'leather':((0.03, 0.4), (0.03, 0.4)), 'tile':((0.03, 0.4), (0.03, 0.4)), 'wood':((0.03, 0.4), (0.03, 0.4)))
18 """
19
20 # Specifies the acceptable width bounds for objects in percentage terms relative to the image size
21 WIDTH_BOUNDS_PCT = {'bottle':((0.03, 0.4), (0.03, 0.4)), 'cable':((0.05, 0.4), (0.05, 0.4)), 'capsule':((0.03, 0.15), (0.03, 0.4)),
22     'hazelnut':((0.03, 0.35), (0.03, 0.35)), 'metal_nut':((0.03, 0.4), (0.03, 0.4)), 'pill':((0.03, 0.2), (0.03, 0.4)),
23     'screw':((0.03, 0.12), (0.03, 0.12)), 'toothbrush':((0.03, 0.4), (0.03, 0.2)), 'transistor':((0.03, 0.4), (0.03, 0.4)),
24     'zipper':((0.03, 0.4), (0.03, 0.2)),
25     'carpet':((0.03, 0.4), (0.03, 0.4)), 'grid':((0.03, 0.4), (0.03, 0.4)),
26     'leather':((0.03, 0.4), (0.03, 0.4)), 'tile':((0.03, 0.4), (0.03, 0.4)), 'wood':((0.03, 0.4), (0.03, 0.4)),
27     "ablation":((0.03, 0.4), (0.03, 0.4)), "breakdown":((0.03, 0.4), (0.03, 0.4)), "fracture":((0.03, 0.4), (0.03, 0.4)), "groove":((0.03, 0.4), (0.03, 0.4)) }
28
29 """ OLD
30 # Number of patches/segments to be extracted from each image for a given category
31 NUM_PATCHES = {'bottle':3, 'cable':3, 'capsule':3, 'hazelnut':3, 'metal_nut':3,
32     'pill':3, 'screw':4, 'toothbrush':3, 'transistor':3, 'zipper':4,
33     'carpet':4, 'grid':4, 'leather':4, 'tile':4, 'wood':4}
34 """
35
36 NUM_PATCHES = {'bottle':3, 'cable':3, 'capsule':3, 'hazelnut':3, 'metal_nut':3,
37     'pill':3, 'screw':4, 'toothbrush':3, 'transistor':3, 'zipper':4,
38     'carpet':4, 'grid':4, 'leather':4, 'tile':4, 'wood':4,
39     "ablation": 4, "breakdown": 4, "fracture": 4, "groove": 4}
40

```

Figure 10: Modification 1 - mvtec.py

```

loss_plot.png U pytorch_model.pt mvtec.py M openllama.py M train_mvtec.py M test_mvtec.py M make_dir.py modeling_llama.py M
cyh.anomaly > anomalygpt > 1AnomalyGPT > code > datasets > mvtec.py
35
36 NUM_PATCHES = {'bottle':3, 'cable':3, 'capsule':3, 'hazelnut':3, 'metal_nut':3,
37     'pill':3, 'screw':4, 'toothbrush':3, 'transistor':3, 'zipper':4,
38     'carpet':4, 'grid':4, 'leather':4, 'tile':4, 'wood':4,
39     "ablation": 4, "breakdown": 4, "fracture": 4, "groove": 4}
40
41 """ OLD
42 # k, x0 pairs
43 INTENSITY_LOGISTIC_PARAMS = {'bottle':(1/12, 24), 'cable':(1/12, 24), 'capsule':(1/2, 4), 'hazelnut':(1/12, 24), 'metal_nut':(1/3, 7),
44     'pill':(1/3, 7), 'screw':(1, 3), 'toothbrush':(1/6, 15), 'transistor':(1/6, 15), 'zipper':(1/6, 15),
45     'carpet':(1/3, 7), 'grid':(1/3, 7), 'leather':(1/3, 7), 'tile':(1/3, 7), 'wood':(1/6, 15)}
46 """
47
48 # The dictionary contains pairs of parameters of a logistic function to normalize the intensity of the pictures - chosen same params as bottle because similar intensity
49 INTENSITY_LOGISTIC_PARAMS = {'bottle':(1/12, 24), 'cable':(1/12, 24), 'capsule':(1/12, 24), 'hazelnut':(1/3, 7),
50     'pill':(1/3, 7), 'screw':(1, 3), 'toothbrush':(1/6, 15), 'transistor':(1/6, 15), 'zipper':(1/6, 15),
51     'carpet':(1/3, 7), 'grid':(1/3, 7), 'leather':(1/3, 7), 'tile':(1/3, 7), 'wood':(1/6, 15)}
52     "ablation": (1/12, 24), "breakdown": (1/12, 24), "fracture": (1/12, 24), "groove": (1/12, 24)}
53
54 """ OLD
55 # bottle is aligned but it's symmetric under rotation
56 UNALIGNED_OBJECTS = ['bottle', 'hazelnut', 'metal_nut', 'screw']
57 """
58
59 UNALIGNED_OBJECTS = ['bottle', 'hazelnut', 'metal_nut', 'screw', "ablation", "breakdown", "fracture", "groove"]
60
61 """ OLD
62 # brightness, threshold pairs
63 BACKGROUND = {'bottle':(200, 60), 'screw':(200, 60), 'capsule':(200, 60), 'zipper':(200, 60),
64     'hazelnut':(20, 20), 'pill':(20, 20), 'toothbrush':(20, 20), 'metal_nut':(20, 20)}
65 """
66
67 BACKGROUND = {'bottle':(200, 60), 'screw':(200, 60), 'capsule':(200, 60), 'zipper':(200, 60),
68     'hazelnut':(20, 20), 'pill':(20, 20), 'toothbrush':(20, 20), 'metal_nut':(20, 20),
69     "ablation": (200, 60), "breakdown": (200, 60), "fracture": (200, 60), "groove": (200, 60)}
70
71 """ OLD
72 OBJECTS = ['bottle', 'cable', 'capsule', 'hazelnut', 'metal_nut',
73     'pill', 'screw', 'toothbrush', 'transistor', 'zipper']
74 """
75
76 OBJECTS = ['bottle', 'cable', 'capsule', 'hazelnut', 'metal_nut',
77     'pill', 'screw', 'toothbrush', 'transistor', 'zipper', "ablation", "breakdown", "fracture", "groove"]
78
79
80 TEXTURES = ['carpet', 'grid', 'leather', 'tile', 'wood']

```

Figure 11: Modification 2 - mvtec.py

```

loss_plot.png U pytorch_model.pt mvtec.py M openllama.py M train_mvtec.py M test_mvtec.py M make_dir.py modeling_llama.py M
cyh.anomaly > anomalygpt > AnomalyGPT > code > datasets > mvtec.py
80 TEXTURES = ['carpet', 'grid', 'leather', 'tile', 'wood']
81
82 describes = {}
83 describes['bottle'] = "This is a photo of a bottle for anomaly detection, which should be round, without any damage, flaw, defect, scratch, hole or broken part."
84 describes['cable'] = "This is a photo of three cables for anomaly detection, they are green, blue and grey, which cannot be mixed or swapped and should be without any damage, flaw, defect, scratch, hole or broken part."
85 describes['capsule'] = "This is a photo of a capsule for anomaly detection, which should be black and orange, with print '500', without any damage, flaw, defect, scratch, hole or broken part."
86 describes['carpet'] = "This is a photo of carpet for anomaly detection, which should be without any damage, flaw, defect, scratch, hole or broken part."
87 describes['grid'] = "This is a photo of grid for anomaly detection, which should be without any damage, flaw, defect, scratch, hole or broken part."
88 describes['hazelnut'] = "This is a photo of a hazelnut for anomaly detection, which should be without any damage, flaw, defect, scratch, hole or broken part."
89 describes['leather'] = "This is a photo of leather for anomaly detection, which should be brown and without any damage, flaw, defect, scratch, hole or broken part."
90 describes['metal_nut'] = "This is a photo of a metal nut for anomaly detection, which should be without any damage, flaw, defect, scratch, hole or broken part, and shouldn't be fliped."
91 describes['pill'] = "This is a photo of a pill for anomaly detection, which should be white with some print '500' and red patterns, without any damage, flaw, defect, scratch, hole or broken part."
92 describes['screw'] = "This is a photo of a screw for anomaly detection, which tail should be sharp without any damage, flaw, defect, scratch, hole or broken part."
93 describes['tile'] = "This is a photo of tile for anomaly detection, which should be without any damage, flaw, defect, scratch, hole or broken part."
94 describes['toothbrush'] = "This is a photo of a toothbrush for anomaly detection, which should be without any damage, flaw, defect, scratch, hole or broken part."
95 describes['transistor'] = "This is a photo of a transistor for anomaly detection, which should be without any damage, flaw, defect, scratch, hole or broken part."
96 describes['wood'] = "This is a photo of wood for anomaly detection, which should be brown with patterns, without any damage, flaw, defect, scratch, hole or broken part."
97 describes['zipper'] = "This is a photo of a zipper for anomaly detection, which should be without any damage, flaw, defect, scratch, hole or broken part."
98
99 # OPT-6 Generated descriptions
100 describes["ablation"] = "This is a photo of a blade for anomaly detection. A blade without ablations should have a smooth, uniform surface with no areas of material removal or wear. The edges should be sharp and intact, with no signs of pitting, erosion, or thinning. The entire blade should exhibit a uniform appearance, with no visible separations, detachments, or irregularities. The surface should be consistent across the entire length of the blade, with no significant variations in texture or color. There should be no sharp edges or corners where material has been removed, and the overall shape should be symmetrical and well-defined." 
101 describes["breakdown"] = "This is a photo of a blade for anomaly detection. A blade without breakdowns should maintain its structural integrity with no signs of cracking, chipping, or flaking. The blade should exhibit a uniform appearance, with no visible separations, detachments, or irregularities. The surface should be consistent across the entire length of the blade, with no significant variations in texture or color. There should be no sharp edges or corners where material has been removed, and the overall shape should be symmetrical and well-defined." 
102 describes["fracture"] = "This is a photo of a blade for anomaly detection. A blade without fractures should have a continuous, unbroken surface with no visible cracks or splits. The blade should be seamless and sturdy, with no signs of stress marks or propagation lines that indicate potential failure points. The surface should be smooth and free from any sharp edges or corners that could lead to further cracking." 
103 describes["groove"] = "This is a photo of a blade for anomaly detection. A blade without grooves should have a completely smooth and even surface with no indentations or carved-out lines. The blade's surface should be uniform and consistent, with no signs of unintended channeling or irregularities in the material." 
104
105 class MvtecDataset(Dataset):
106     """
107     # Old __init__ without changes
108     def __init__(self, root_dir: str):
109         self.root_dir = root_dir
110         # self.transform = transforms.Compose([
111             transforms.Resize((224, 224), interpolation=transforms.InterpolationMode.BICUBIC)
112         ])
113
114         self.norm_transform = transforms.Compose([
115             [
116                 transforms.ToTensor(),
117                 transforms.Normalize(
118                     mean=[0.48145466, 0.4578275, 0.408821073],
119                     std=[0.26862954, 0.26108258, 0.27577711]
120                 )
121             ]
122         ])
123
124
125         self.paths = []
126         self.x = []
127         for root, dirs, files in os.walk(root_dir):
128             for file in files:
129                 file_path = os.path.join(root, file)
130                 if "train" in file_path and "good" in file_path and ".png" in file:
131                     self.paths.append(file_path)
132                     self.x.append(self.norm_transform(Image.open(file_path).convert("RGB")))
133
134         self.prev_idx = np.random.randint(len(self.paths))
135
136     # New __init__ with changes including try/except block to skip bugged images
137     def __init__(self, root_dir: str):
138         self.root_dir = root_dir
139         # self.transform = transforms.Compose([
140             transforms.Resize(

```

Figure 12: Modification 3 - mvtec.py

```

1 class MvtecDataset(Dataset):
2     ...
3     # Old __init__ without changes
4     def __init__(self, root_dir='trt'):
5         self.root_dir = root_dir
6         # self.transform = transform
7         self.transform = transforms.Resize(
8             (224, 224), interpolation=transforms.InterpolationMode.BICUBIC
9         )
10
11     self.norm_transform = transforms.Compose(
12         [
13             transforms.ToTensor(),
14             transforms.Normalize(
15                 mean=[0.4814546, 0.4578275, 0.4882077],
16                 std=[0.2680254, 0.26130258, 0.27577713],),
17         ]
18     )
19
20     self.paths = []
21     self.x = []
22     for root, dirs, files in os.walk(root_dir):
23         for file in files:
24             file_path = os.path.join(root, file)
25             if 'good' in file_path and 'png' in file:
26                 self.paths.append(file_path)
27                 self.x.append(self.transform(Image.open(file_path).convert('RGB')))
28     self.prev_idx = np.random.randint(len(self.paths))
29
30     # New __init__ with changes including try/except block to skip bugged images
31     def __init__(self, root_dir='trt'):
32         self.root_dir = root_dir
33         # self.transform = transform
34         self.transform = transforms.Resize(
35             (224, 224), interpolation=transforms.InterpolationMode.BICUBIC
36         )
37
38         self.norm_transform = transforms.Compose(
39             [
40                 transforms.ToTensor(),
41                 transforms.Normalize(
42                     mean=[0.4814546, 0.4578275, 0.4882077],
43                     std=[0.2680254, 0.26130258, 0.27577713],),
44             ]
45         )
46
47         self.paths = []
48         self.x = []
49         for root, dirs, files in os.walk(root_dir):
50             for file in files:
51                 file_path = os.path.join(root, file)
52                 if 'good' in file_path and 'png' in file:
53                     try:
54                         image = Image.open(file_path).convert('RGB')
55                         self.paths.append(file_path)
56                         self.x.append(self.transform(image))
57                     except UnidentifiedImageError:
58                         print("Skipping corrupted image file: " + file_path)
59                     except:
60                         print("Unexpected error loading image " + file_path)
61
62     self.prev_idx = np.random.randint(len(self.paths))
63
64     def __len__(self):
65         return len(self.paths)

```

Figure 13: Modification 4 - mvtec.py

Introducing [these new changes](#) into the original scripts and adding [new samples](#) probably were going to [disrupt the loss function alongside with the results](#). To avoid this issue, Serena proposed to modify the loss function to make it more acquainted with the new data. Even though the modification of the loss function is quite complicated, the proposals that I prepared were, at least the first one, seemed quite logical.

My [proposals](#) consisted in the following [loss functions](#):

- Weighted Cross-Entropy Loss**: I suspect that by adding the new dataset, which include 1456 images of the same object (blade) during the training (separated in 4 different classes - ablation, breakdown, fracture and groove) will imbalance the dataset and might lead to overfitting.

$$L = \alpha \cdot L_{\text{W_CE}} + \beta \cdot L_{\text{focal}} + \delta \cdot L_{\text{dice}}$$

where the weighted cross-entropy loss function $L_{\text{W_CE}}$ depends on the term w_{y_i} representing the inverse frequency of each class in the training data, defined as follows:

$$L_{\text{W_CE}} = -\frac{1}{n} \cdot \sum_{i=1}^n w_{y_i} \cdot y_i \cdot \log(p_i)$$

- Combined Focal Loss with Adaptative Gamma**: This function was proposed by GPT4o, arguing that incorporating an adaptive version of the focal loss to dynamically adjust the focusing parameter γ based on class imbalance, might work.

$$L = \alpha \cdot L_{\text{CE}} + \beta \cdot L_{\text{adaptative_focal}} + \delta \cdot L_{\text{dice}}$$

where the adaptative focal loss would include the parameter $\gamma_{y_i} = \frac{\gamma_{\text{base}}}{w_{y_i}}$ and defined as follows:

$$L_{\text{adaptative_focal}} = -\frac{1}{n} \cdot \sum_{i=1}^n (1 - p_i)^{\gamma_{y_i}} \cdot \log(p_i)$$

The proposals were postponed in order to follow an easier path which would be applying [data augmentation](#) to balance all the classes, and that will be managed by [model V5](#). With all the previous considerations, the training

process can already start and shouldn't stop until its complete execution. Nonetheless, when it comes to testing the model, some modifications were once again needed to make it work. The [first issue](#) came with the names of the images, they had to be changed to ascending order starting from 000.png (for train and test) and 000_mask.png (ground_truth), and this renaming had to be performed for all new classes (ablation, breakdown, fracture and groove). The [second issue](#), was just to include the new classes following the same order as the defined structure as well as the descriptions for each class.

Figure 14: Modification 1 for Model V4- test_mvtec.py

Figure 15: Modification 2 for Model V4 - test_mvtec.py

5.3.1.5 Model V5

Finally, the latest model, model V5, builds on model V4 by using **data augmentation** to balance the object classes. There are two main ways data augmentation was applied to the original dataset. First, **the training samples** in the "train/good" folder from each original class (excluding the four new classes) were **increased from about 200 to roughly 1000 samples**. Second, data augmentation was also applied to the **test and ground_truth datasets** to boost the number of test samples, **duplicating their original size**, providing a more reasonable and realistic evaluation of the model's learning and inference capabilities. Let's explain each data augmentation process.

- **Training Data Augmentation**: For each class, there was conducted a meticulous study of which operations would suit better the characteristics of the corresponding object and there were applied mostly the following operations: **rotations, scaling, brightness tuning, horizontal and vertical flips**.

Operations such as translations, too wide rotations or blurring images **were discarded** due to obvious reasons, the original images would include black backgrounds and new anomalies which would not be represented in their respective labels (ground_truth images).

The different operations are included in the upcoming table, and **the percentage** shown (usually) under the different operations represents the **probability of applying such operation** to a given image. Finally, after the generation of the new training samples, we must rename the new images to adapt to the original script, otherwise when running the testing script, we will jump into different bugs. The code is implemented into the script `data_augmentation.py`.

- **Test & Ground Truth Data Augmentation**: As the training dataset grows, it is necessary to augment the testing dataset correspondingly. To generate new anomalous images, a technique known as **Poisson Image Editing** will be used. This method involves taking **a labeled image describing the location of the anomaly** (mask in ground_truth), **the anomalous picture**, and a **non-anomalous picture** to create a **new anomalous picture**. The defect from the original anomalous image is seamlessly blended into the new non-anomalous image, resulting in a new anomalous picture. The defect can be placed voluntarily into the non-anomalous picture generating therefore new samples as realistic as the original ones. Even though these new anomalous samples look real, in some cases they don't vary sufficiently from the original anomalous picture. Therefore, to solve such issue, after applying the **Poisson** technique, additional operations such as rotations or flips are applied. The code is implemented into the script `poisson_image_editing.py`. Some examples of the previously mentioned technique, based on a paper published in 2003, can be found below.

The operations implemented can be summarized in the following tables, grouping the different operations applied during the process for the training and testing samples augmentation. Notice that, in the table in which is compared the dataset used for **model V4** and **model V5**, the table contains the number of elements in the mentioned folders.

For instance, in the folder "/ablation/ground_truth/" we find a total of 151 elements (in this case, masks), in the folder "/ablation/test/" we find 151 elements contained in the same folders as in "/ablation/ground_truth/" (which corresponds to the folders background, illumination, same and view) and 109 images contained in the folder "/ablation/test/good/". Moreover, the increase in training samples is roughly 5 times the original size and a duplication for testing and ground_truth samples.

5.3.1.6 Metric: AUROC

The metric that is used to evaluate the model's performance is AUROC, which specifically refers to the area under the Receiver Operating Characteristic (ROC) curve. The ROC curve is a plot of the true positive rate (TPR or recall) against the false positive rate (FPR) at various threshold settings. There are **several reasons** that lead **to choose such metric**, including that such metric is commonly used to evaluate the performance of binary classification models and is quite useful for comparing different models or classifiers as it is independent of the classification threshold.

Class	Operation 1	Operation 2	Operation 3	Operation 4
Bottle	Rotate 180° 80%	Rotate 90° 60%	Horizontal Flip 50%	Flip vertical
Cable	Rotate 180° 80%	Rotate 90° 60%	Horizontal Flip	Brightness 90-110% 70%
Capsule	Rotate 270°	Rotate 180° 50%	Brightness 90-110% 70%	X
Carpet	Rotate 180° 50%	Rotate 270° 50%	Vertical Flip 50%	Brightness 90-110% 70%
Grid	Rotate 180° 50%	Rotate 270° 50%	Vertical Flip 50%	Brightness 90-110% 70%
Hazelnut	Rotate 90° 50%	Rotate 270° 50%	Horizontal Flip 50%	Brightness 90-110% 70%
Leather	Rotate 90° 50%	Rotate 180° 50%	Vertical Flip	Brightness 90%-110% 70%
Metal Nut	Rotate 90° 50%	Rotate 180° 50%	Vertical Flip	X
Pill	Rotate [-35°, 35°]	Scaling 0-20%	Flip horizontal 60%	Vertical Flip 70%
Screw	Rotate [-35°, 35°]	Scaling 20-50%	Horizontal Flip 60%	Vertical Flip 60%
Tile	Rotate [-35°, 35°]	Zoom 20-40%	X	X
Toothbrush	Vertical Flip	Rotate [-20°, 20°]	X	X
Transistor	Rotate [-15°, 15°]	Vertical Flip 60%	X	X
Wood	Rotate [-50°, 50°]	Scaling 50-80%	Vertical Flip 70%	X
Zipper	Rotate [-15°, 15°]	Scaling 40-60%	Vertical Flip 70%	X

Table 2: Data Augmentation: Training Dataset

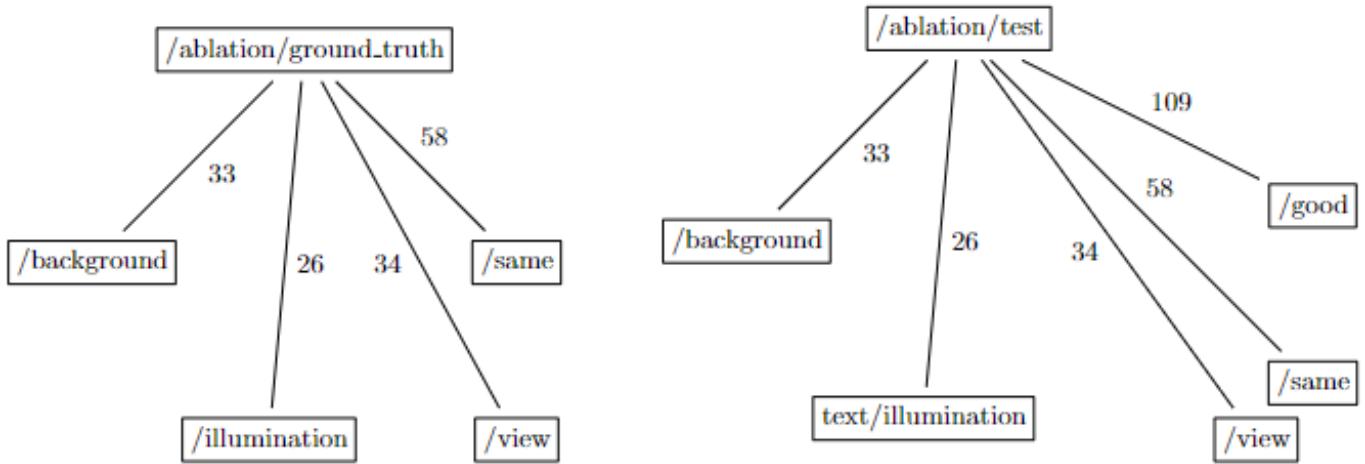


Figure 16: Ground_truth & Test folder structure: Ablation

		Class	V4	V5	V6	
GT	ablation	Class	V4	V5	V6	
		151	151	151	140	
		151+109	151+109	151+109	140+40	
	bottle	364	364	364	400	
		63	126	126	184	
		63+20	126+20	126+20	184+32	
	breakdown	209	1000	400	184+32	
		310	310	310	186	
		310+109	310+109	310+109	186+22	
	cable	364	364	364	186+22	
Test		92	184	184	282	
		92+58	184+58	184+58	282+26	
		224	1000	400	282+26	
Train	capsule	109	218	218	119	
		109+23	218+23	218+23	238	
		219	1000	400	238+41	
GT	carpet	89	178	178	320	
		89+28	178+28	178+28	1000	
		280	1000	400	400	
Test	fracture	370	370	370	84	
		370+109	370+109	370+109	84+33	
		364	364	364	168+33	
Train	grid	57	114	114	230	
		57+21	114+21	114+21	1000	
		264	1000	400	400	
GT	groove	241	241	241	40	
		241+109	241+109	241+109	80+60	
		364	364	364	1000	
Test	hazelnut	70	140	140	400	
		70+40	140+40	140+40	1000	
		391	1000	400	400	
Train	leather	92	184	184	119	
		92+32	184+32	184+32	238	
		245	1000	400	238+41	
GT	metal_nut	93	186	186	320	
		93+22	186+22	186+22	1000	
		220	400	400	400	
Test	pill	141	282	282	84	
		141+26	282+26	282+26	168	
		267	1000	400	168+33	
Train	screw	119	238	238	230	
		119+41	238+41	238+41	1000	
		320	400	400	400	
GT	tile	84	168	168	60	
		84+33	168+33	168+33	60	
		230	400	400	60	
Test	toothbrush	30	60	60	30+12	
		60	1000	400	60+12	
		1000	400	400	60+12	
Train	transistor	40	80	80	40	
		40+60	80+60	80+60	213	
		1000	400	400	1000	
GT	wood	60	120	120	60+19	
		60+19	120+19	120+19	247	
		247	1000	400	1000	
Test	zipper	119	238	238	119	
		119+32	238+32	238+32	240	
		240	400	400	400	

Table 3: Data Augmentation: Model Comparison V4, V5 & V6

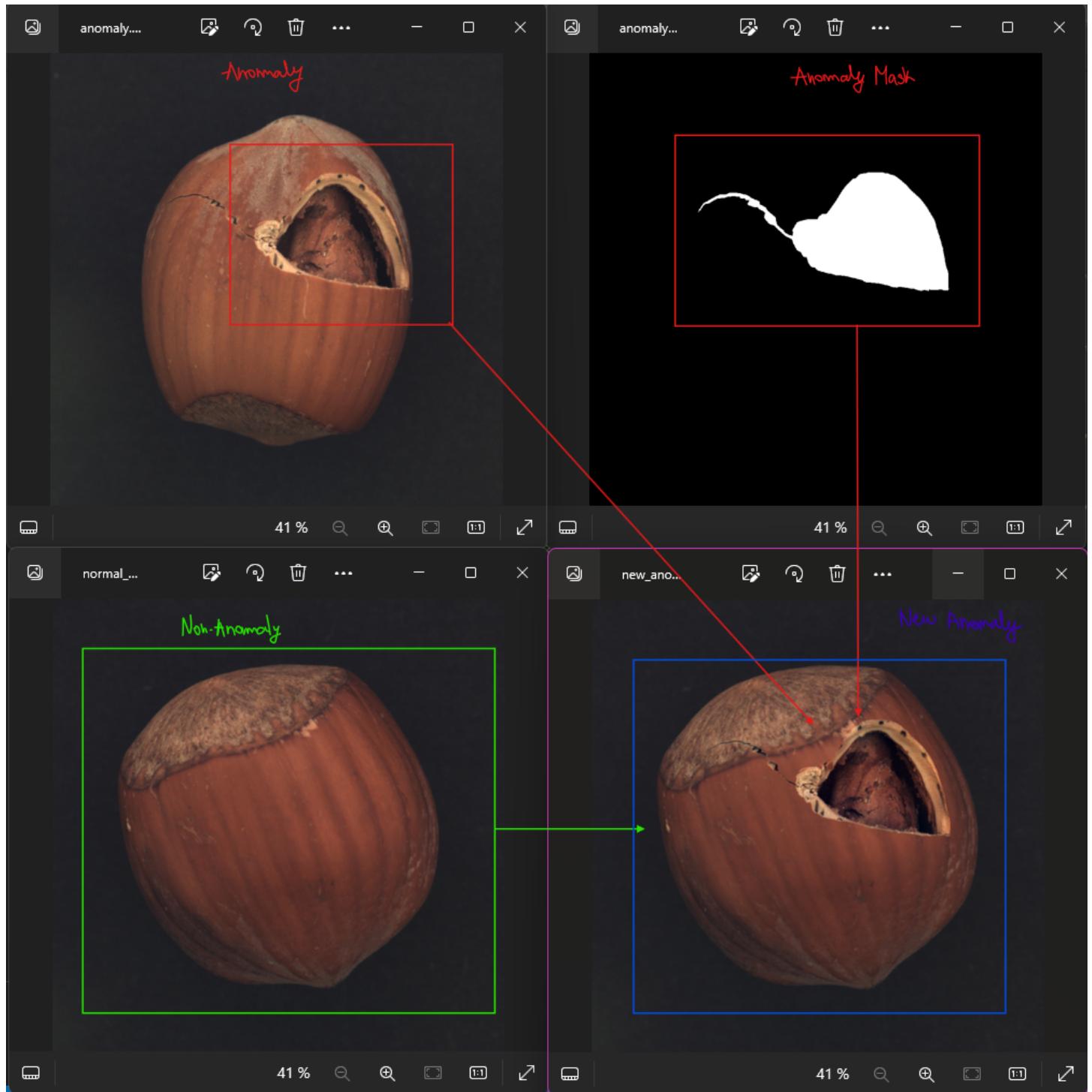


Figure 17: Poisson Editing Technique: Hazelnut

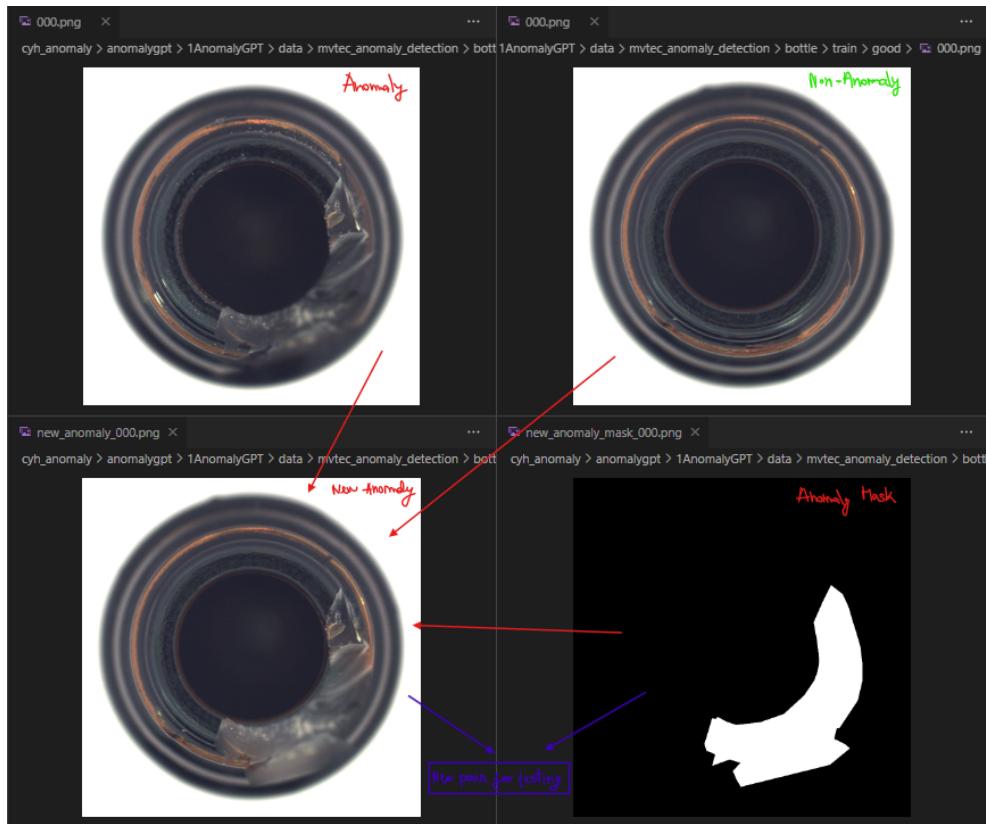


Figure 18: Poisson Editing Technique: Bottle

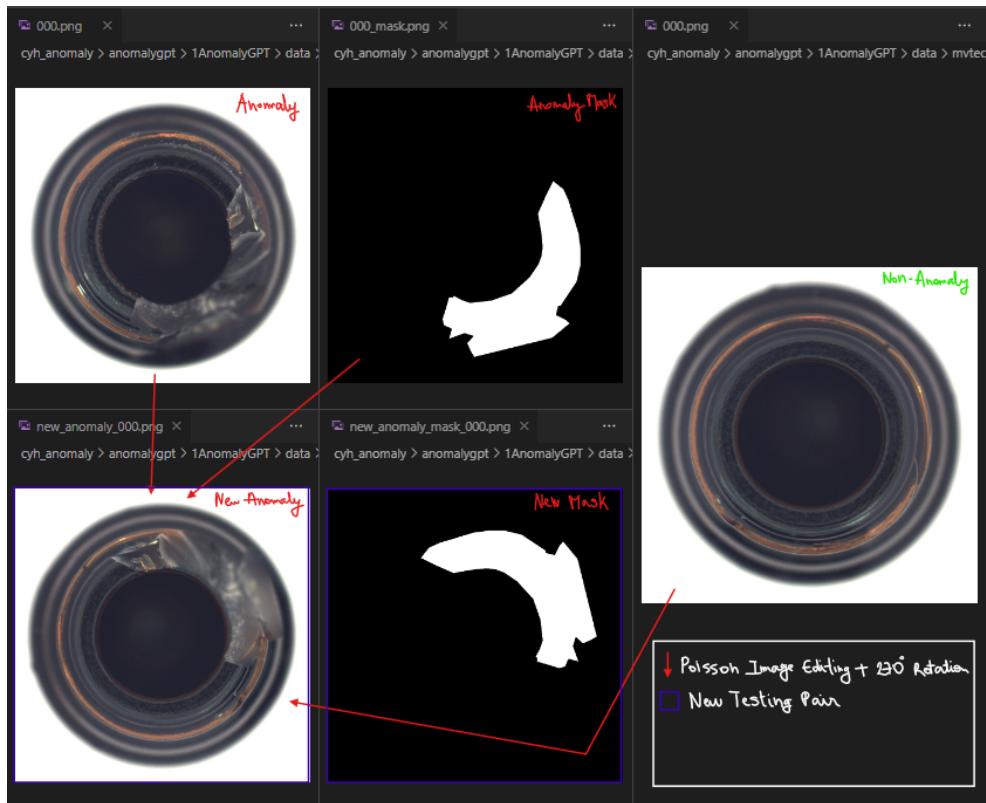


Figure 19: Improved Poisson Editing Example: Bottle

5.4 Results

To run the evaluation script corresponding to the chosen training dataset (`test_mvtec.py`), it simply needs to be executed in the correct folder after replacing the path from the trained model, as shown in the below picture:

```
File Edit Selection View Go Run Terminal Help < > work [SSH: 140.110.18.89]
EXPLORER
WORK [SSH: 140.110.18.89]
anomalygpt
  ▾ AnomalyGPT
    code
      ▾ cmd
        train_mvtec.py
      train_visa_2040518
      train_visa_on_github
      README.md
    config
    datasets
    docstring
    model
    scripts
    tests
    utils
    environment.yml
  fmm.png
  header.hzv
  loss_dpmg
  output.png
  README.txt
  test_mvtec.py
  test_visa.py
  train_mvtec_cnn.py
  train_mvtec.py
  train_visa.py
  visual_train_mvtec.py
  web_demo.py
  Dockerfile
  LICENSE
  README.md
  requirements.txt
  dataset
  22imagebind
  23cvcuna
  24cvuna-pandapt
  25cvuna-weight
  cv_anomaly
  cv_deepspeed
  OUTLINE
No symbols found in document Test_mvtec.py

... < - > Welcome train_mvtec.py M modeling.lamapay M openlam_pefany M train_mvtec.sh M agentpy M F train_Sat May 18 21:47:27 2024.log test_mvtec.py M
anomalygpt / AnomalyGPT / code > test_mvtec.py
  1 parser.add_argument('--round', type=int, default=3)
  2
  3 command_args = parser.parse_args()
  4
  5
  6
  7
  8
  9
  10
  11
  12
  13
  14
  15
  16
  17
  18
  19
  20
  21
  22
  23
  24
  25
  26
  27
  28
  29
  30
  31
  32
  33
  34
  35
  36
  37
  38
  39
  40
  41
  42
  43
  44
  45
  46
  47
  48
  49
  50
  51
  52
  53
  54
  55
  56
  57
  58
  59
  60
  61
  62
  63
  64
  65
  66
  67
  68
  69
  70
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
  100
  101
  102
  103
  104
  105
  106
  107
  108
  109
  110
  111
  112
  113
  114
  115
  116
  117
  118
  119
  120
  121
  122
  123
  124
  125
  126
  127
  128
  129
  130
  131
  132
  133
  134
  135
  136
  137
  138
  139
  140
  141
  142
  143
  144
  145
  146
  147
  148
  149
  150
  151
  152
  153
  154
  155
  156
  157
  158
  159
  160
  161
  162
  163
  164
  165
  166
  167
  168
  169
  170
  171
  172
  173
  174
  175
  176
  177
  178
  179
  180
  181
  182
  183
  184
  185
  186
  187
  188
  189
  190
  191
  192
  193
  194
  195
  196
  197
  198
  199
  200
  201
  202
  203
  204
  205
  206
  207
  208
  209
  210
  211
  212
  213
  214
  215
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240
  241
  242
  243
  244
  245
  246
  247
  248
  249
  250
  251
  252
  253
  254
  255
  256
  257
  258
  259
  260
  261
  262
  263
  264
  265
  266
  267
  268
  269
  270
  271
  272
  273
  274
  275
  276
  277
  278
  279
  280
  281
  282
  283
  284
  285
  286
  287
  288
  289
  290
  291
  292
  293
  294
  295
  296
  297
  298
  299
  300
  301
  302
  303
  304
  305
  306
  307
  308
  309
  310
  311
  312
  313
  314
  315
  316
  317
  318
  319
  320
  321
  322
  323
  324
  325
  326
  327
  328
  329
  330
  331
  332
  333
  334
  335
  336
  337
  338
  339
  340
  341
  342
  343
  344
  345
  346
  347
  348
  349
  350
  351
  352
  353
  354
  355
  356
  357
  358
  359
  360
  361
  362
  363
  364
  365
  366
  367
  368
  369
  370
  371
  372
  373
  374
  375
  376
  377
  378
  379
  380
  381
  382
  383
  384
  385
  386
  387
  388
  389
  390
  391
  392
  393
  394
  395
  396
  397
  398
  399
  400
  401
  402
  403
  404
  405
  406
  407
  408
  409
  410
  411
  412
  413
  414
  415
  416
  417
  418
  419
  420
  421
  422
  423
  424
  425
  426
  427
  428
  429
  430
  431
  432
  433
  434
  435
  436
  437
  438
  439
  440
  441
  442
  443
  444
  445
  446
  447
  448
  449
  450
  451
  452
  453
  454
  455
  456
  457
  458
  459
  460
  461
  462
  463
  464
  465
  466
  467
  468
  469
  470
  471
  472
  473
  474
  475
  476
  477
  478
  479
  480
  481
  482
  483
  484
  485
  486
  487
  488
  489
  490
  491
  492
  493
  494
  495
  496
  497
  498
  499
  500
  501
  502
  503
  504
  505
  506
  507
  508
  509
  510
  511
  512
  513
  514
  515
  516
  517
  518
  519
  520
  521
  522
  523
  524
  525
  526
  527
  528
  529
  530
  531
  532
  533
  534
  535
  536
  537
  538
  539
  540
  541
  542
  543
  544
  545
  546
  547
  548
  549
  550
  551
  552
  553
  554
  555
  556
  557
  558
  559
  560
  561
  562
  563
  564
  565
  566
  567
  568
  569
  570
  571
  572
  573
  574
  575
  576
  577
  578
  579
  580
  581
  582
  583
  584
  585
  586
  587
  588
  589
  590
  591
  592
  593
  594
  595
  596
  597
  598
  599
  600
  601
  602
  603
  604
  605
  606
  607
  608
  609
  610
  611
  612
  613
  614
  615
  616
  617
  618
  619
  620
  621
  622
  623
  624
  625
  626
  627
  628
  629
  630
  631
  632
  633
  634
  635
  636
  637
  638
  639
  640
  641
  642
  643
  644
  645
  646
  647
  648
  649
  650
  651
  652
  653
  654
  655
  656
  657
  658
  659
  660
  661
  662
  663
  664
  665
  666
  667
  668
  669
  670
  671
  672
  673
  674
  675
  676
  677
  678
  679
  680
  681
  682
  683
  684
  685
  686
  687
  688
  689
  690
  691
  692
  693
  694
  695
  696
  697
  698
  699
  700
  701
  702
  703
  704
  705
  706
  707
  708
  709
  710
  711
  712
  713
  714
  715
  716
  717
  718
  719
  720
  721
  722
  723
  724
  725
  726
  727
  728
  729
  730
  731
  732
  733
  734
  735
  736
  737
  738
  739
  740
  741
  742
  743
  744
  745
  746
  747
  748
  749
  750
  751
  752
  753
  754
  755
  756
  757
  758
  759
  759
  760
  761
  762
  763
  764
  765
  766
  767
  768
  769
  770
  771
  772
  773
  774
  775
  776
  777
  778
  779
  779
  780
  781
  782
  783
  784
  785
  786
  787
  788
  789
  789
  790
  791
  792
  793
  794
  795
  796
  797
  798
  799
  800
  801
  802
  803
  804
  805
  806
  807
  808
  809
  809
  810
  811
  812
  813
  814
  815
  816
  817
  818
  819
  819
  820
  821
  822
  823
  824
  825
  826
  827
  828
  829
  829
  830
  831
  832
  833
  834
  835
  836
  837
  838
  839
  839
  840
  841
  842
  843
  844
  845
  846
  847
  848
  849
  849
  850
  851
  852
  853
  854
  855
  856
  857
  858
  859
  859
  860
  861
  862
  863
  864
  865
  866
  867
  868
  869
  869
  870
  871
  872
  873
  874
  875
  876
  877
  878
  879
  879
  880
  881
  882
  883
  884
  885
  886
  887
  888
  889
  889
  890
  891
  892
  893
  894
  895
  896
  897
  898
  899
  900
  901
  902
  903
  904
  905
  906
  907
  908
  909
  910
  911
  912
  913
  914
  915
  916
  917
  918
  919
  920
  921
  922
  923
  924
  925
  926
  927
  928
  929
  930
  931
  932
  933
  934
  935
  936
  937
  938
  939
  940
  941
  942
  943
  944
  945
  946
  947
  948
  949
  950
  951
  952
  953
  954
  955
  956
  957
  958
  959
  959
  960
  961
  962
  963
  964
  965
  966
  967
  968
  969
  969
  970
  971
  972
  973
  974
  975
  976
  977
  978
  979
  979
  980
  981
  982
  983
  984
  985
  986
  987
  988
  989
  989
  990
  991
  992
  993
  994
  995
  996
  997
  998
  999
  999
  1000
  1001
  1002
  1003
  1004
  1005
  1006
  1007
  1008
  1009
  1009
  1010
  1011
  1012
  1013
  1014
  1015
  1016
  1017
  1018
  1019
  1019
  1020
  1021
  1022
  1023
  1024
  1025
  1026
  1027
  1028
  1029
  1029
  1030
  1031
  1032
  1033
  1034
  1035
  1036
  1037
  1038
  1039
  1039
  1040
  1041
  1042
  1043
  1044
  1045
  1046
  1047
  1047
  1048
  1049
  1049
  1050
  1051
  1052
  1053
  1054
  1055
  1056
  1057
  1058
  1059
  1059
  1060
  1061
  1062
  1063
  1064
  1065
  1066
  1067
  1067
  1068
  1069
  1069
  1070
  1071
  1072
  1073
  1074
  1075
  1076
  1077
  1078
  1078
  1079
  1080
  1081
  1082
  1083
  1084
  1085
  1086
  1087
  1087
  1088
  1089
  1089
  1090
  1091
  1092
  1093
  1094
  1095
  1096
  1097
  1097
  1098
  1099
  1099
  1100
  1101
  1102
  1103
  1104
  1105
  1106
  1107
  1107
  1108
  1109
  1109
  1110
  1111
  1112
  1113
  1114
  1115
  1116
  1117
  1117
  1118
  1119
  1119
  1120
  1121
  1122
  1123
  1124
  1125
  1126
  1127
  1127
  1128
  1129
  1129
  1130
  1131
  1132
  1133
  1134
  1135
  1136
  1137
  1137
  1138
  1139
  1139
  1140
  1141
  1142
  1143
  1144
  1145
  1145
  1146
  1147
  1147
  1148
  1149
  1149
  1150
  1151
  1152
  1153
  1154
  1155
  1156
  1157
  1157
  1158
  1159
  1159
  1160
  1161
  1162
  1163
  1164
  1165
  1166
  1167
  1167
  1168
  1169
  1169
  1170
  1171
  1172
  1173
  1174
  1175
  1176
  1177
  1177
  1178
  1179
  1179
  1180
  1181
  1182
  1183
  1184
  1184
  1185
  1186
  1186
  1187
  1188
  1188
  1189
  1189
  1190
  1191
  1192
  1193
  1194
  1195
  1196
  1197
  1197
  1198
  1199
  1199
  1200
  1201
  1202
  1203
  1204
  1205
  1206
  1207
  1207
  1208
  1209
  1209
  1210
  1211
  1212
  1213
  1214
  1215
  1216
  1216
  1217
  1218
  1218
  1219
  1220
  1221
  1222
  1223
  1224
  1225
  1226
  1226
  1227
  1228
  1228
  1229
  1230
  1231
  1232
  1233
  1234
  1235
  1236
  1237
  1237
  1238
  1239
  1239
  1240
  1241
  1242
  1243
  1244
  1245
  1245
  1246
  1247
  1247
  1248
  1249
  1249
  1250
  1251
  1252
  1253
  1254
  1255
  1256
  1256
  1257
  1258
  1258
  1259
  1260
  1261
  1262
  1263
  1264
  1265
  1266
  1266
  1267
  1268
  1268
  1269
  1270
  1271
  1272
  1273
  1274
  1275
  1275
  1276
  1277
  1277
  1278
  1279
  1279
  1280
  1281
  1282
  1283
  1284
  1285
  1285
  1286
  1287
  1287
  1288
  1289
  1289
  1290
  1291
  1292
  1293
  1294
  1295
  1295
  1296
  1297
  1297
  1298
  1299
  1299
  1300
  1301
  1302
  1303
  1304
  1305
  1306
  1306
  1307
  1308
  1308
  1309
  1310
  1311
  1312
  1313
  1314
  1315
  1315
  1316
  1317
  1317
  1318
  1319
  1319
  1320
  1321
  1322
  1323
  1324
  1325
  1325
  1326
  1327
  1327
  1328
  1329
  1329
  1330
  1331
  1332
  1333
  1334
  1335
  1336
  1336
  1337
  1338
  1338
  1339
  1340
  1341
  1342
  1343
  1344
  1344
  1345
  1346
  1346
  1347
  1348
  1348
  1349
  1350
  1351
  1352
  1353
  1354
  1355
  1355
  1356
  1357
  1357
  1358
  1359
  1359
  1360
  1361
  1362
  1363
  1364
  1365
  1365
  1366
  1367
  1367
  1368
  1369
  1369
  1370
  1371
  1372
  1373
  1374
  1375
  1375
  1376
  1377
  1377
  1378
  1379
  1379
  1380
  1381
  1382
  1383
  1384
  1385
  1385
  1386
  1387
  1387
  1388
  1389
  1389
  1390
  1391
  1392
  1393
  1394
  1394
  1395
  1396
  1396
  1397
  1398
  1398
  1399
  1400
  1401
  1402
  1403
  1403
  1404
  1405
  1405
  1406
  1407
  1407
  1408
  1409
  1409
  1410
  1411
  1412
  1413
  1413
  1414
  1415
  1415
  1416
  1417
  1417
  1418
  1419
  1419
  1420
  1421
  1422
  1423
  1424
  1425
  1425
  1426
  1427
  1427
  1428
  1429
  1429
  1430
  1431
  1432
  1433
  1434
  1435
  1435
  1436
  1437
  1437
  1438
  1439
  1439
  1440
  1441
  1442
  1443
  1444
  1444
  1445
  1446
  1446
  1447
  1448
  1448
  1449
  1450
  1451
  1452
  1453
  1453
  1454
  1455
  1455
  1456
  1457
  1457
  1458
  1459
  1459
  1460
  1461
  1462
  1463
  1464
  1464
  1465
  1466
  1466
  1467
  1468
  1468
  1469
  1470
  1471
  1472
  1473
  1474
  1474
  1475
  1476
  1476
  1477
  1478
  1478
  1479
  1480
  1481
  1482
  1483
  1483
  1484
  1485
  1485
  1486
  1487
  1487
  1488
  1489
  1489
  1490
  1491
  1492
  1493
  1493
  1494
  1495
  1495
  1496
  1497
  1497
  1498
  1499
  1499
  1500
  1501
  1502
  1503
  1503
  1504
  1505
  1505
  1506
  1507
  1507
  1508
  1509
  1509
  1510
  1511
  1512
  1513
  1513
  1514
  1515
  1515
  1516
  1517
  1517
  1518
  1519
  1519
  1520
  1521
  1522
  1523
  1524
  1524
  1525
  1526
  1526
  1527
  1528
  1528
  1529
  1530
  1531
  1532
  1533
  1534
  1534
  1535
  1536
  1536
  1537
  1538
  1538
  1539
  1540
  1541
  1542
  1543
  1543
  1544
  1545
  1545
  1546
  1547
  1547
  1548
  1549
  1549
  1550
  1551
  1552
  1553
  1554
  1554
  1555
  1556
  1556
  1557
  1558
  1558
  1559
  1560
  1561
  1562
  1563
  1563
  1564
  1565
  1565
  1566
  1567
  1567
  1568
  1569
  1569
  1570
  1571
  1572
  1573
  1574
  1574
  1575
  1576
  1576
  1577
  1578
  1578
  1579
  1580
  1581
  1582
  1583
  1583
  1584
  1585
  1585
  1586
  1587
  1587
  1588
  1589
  1589
  1590
  1591
  1592
  1593
  1593
  1594
  1595
  1595
  1596
  1597
  1597
  1598
  1599
  1599
  1600
  1601
  1602
  1603
  1603
  1604
  1605
  1605
  1606
  1607
  1607
  1608
  1609
  1609
  1610
  1611
  1612
  1613
  1613
  1614
  1615
  1615
  1616
  1617
  1617
  1618
  1619
  1619
  1620
  1621
  1622
  1623
  1624
  1624
  1625
  1626
  1626
  1627
  1628
  1628
  1629
  1630
  1631
  1632
  1633
  1634
  1634
  1635
  1636
  1636
  1637
  1638
  1638
  1639
  1640
  1641
  1642
  1643
  1643
  1644
  1645
  1645
  1646
  1647
  1647
  1648
  1649
  1649
  1650
  1651
  1652
  1653
  1654
  1654
  1655
  1656
  1656
  1657
  1658
  1658
  1659
  1660
  1661
  1662
  1663
  
```

Figure 20: Modifications: test_mvtec.py

In our case, the test considers a **1-Shot** setup and as shown in the original comparison table, we are comparing the highlighted scores with the ones obtained in the training. At the end of the execution, the results found for the trainings of each model are the following:

Model	Type	Score	Precision
Baseline	Image_AUROC	94.232	91.655
	Pixel_AUROC	95.4573	
V2	Image_AUROC	94.2320	89.5601
	Pixel_AUROC	95.4573	
V3	Image_AUROC	94.232	0
	Pixel_AUROC	95.4573	
V4	Image_AUROC	91.9205	85.8738
	Pixel_AUROC	95.3716	
V5	Image_AUROC	93.0984	86.2155
	Pixel_AUROC	94.7474	
V6	Image_AUROC	93.0984	87.8672
	Pixel_AUROC	94.7474	

Table 4: Model's Results

As it can be noticed, the model V3 reaches the same performance as the previous model V2 in terms of both, Image and Pixel AUC however, the accuracy is 0. There must be some kind of mistake. Moreover, model's V2 performance compared to the baseline model in which includes the 3 models and the optimizer LoRA (like V2), improves the Pixel-AUC, although worsening the Image-AUC.

The following tables include the results class per class of all model trainings. A noticeable remark is that model V3 produces a null precision for all classes due to the batch_normalization operation during training.

Setup	Method	MVTec-AD			VisA		
		Image-AUC	Pixel-AUC	Accuracy	Image-AUC	Pixel-AUC	Accuracy
1-shot	SPADE	81.0 ± 2.0	91.2 ± 0.4	-	79.5 ± 4.0	95.6 ± 0.4	-
	PaDiM	76.6 ± 3.1	89.3 ± 0.9	-	62.8 ± 5.4	89.9 ± 0.8	-
	PatchCore	83.4 ± 3.0	92.0 ± 1.0	-	79.9 ± 2.9	95.4 ± 0.6	-
	WinCLIP	93.1 ± 2.0	95.2 ± 0.5	-	83.8 ± 4.0	96.4 ± 0.4	-
AnomalyGPT (ours)		94.1 ± 1.1	95.3 ± 0.1	86.1 ± 1.1	87.4 ± 0.8	96.2 ± 0.1	77.4 ± 1.0
2-shot	SPADE	82.9 ± 2.6	92.0 ± 0.3	-	80.7 ± 5.0	96.2 ± 0.4	-
	PaDiM	78.9 ± 3.1	91.3 ± 0.7	-	67.4 ± 5.1	92.0 ± 0.7	-
	PatchCore	86.3 ± 3.3	93.3 ± 0.6	-	81.6 ± 4.0	96.1 ± 0.5	-
	WinCLIP	94.4 ± 1.3	96.0 ± 0.3	-	84.6 ± 2.4	96.8 ± 0.3	-
AnomalyGPT (ours)		95.5 ± 0.8	95.6 ± 0.2	84.8 ± 0.8	88.6 ± 0.7	96.4 ± 0.1	77.5 ± 0.3
4-shot	SPADE	84.8 ± 2.5	92.7 ± 0.3	-	81.7 ± 3.4	96.6 ± 0.3	-
	PaDiM	80.4 ± 2.5	92.6 ± 0.7	-	72.8 ± 2.9	93.2 ± 0.5	-
	PatchCore	88.8 ± 2.6	94.3 ± 0.5	-	85.3 ± 2.1	96.8 ± 0.3	-
	WinCLIP	95.2 ± 1.3	96.2 ± 0.3	-	87.3 ± 1.8	97.2 ± 0.2	-
AnomalyGPT (ours)		96.3 ± 0.3	96.2 ± 0.1	85.0 ± 0.3	90.6 ± 0.7	96.7 ± 0.1	77.7 ± 0.4

Table 2. Few-shot IAD results on MVTec-AD and VisA datasets. Results are listed as the average of 5 runs and the best-performing method is in **bold**. The results for SPADE, PaDiM, PatchCore and WinCLIP are reported from [11].

Figure 21: Baseline Results

Precision Class \	Baseline	V2	V3	V4	V5	V6
ablation	X	X	X	61.92	65.38	75
bottle	92.77	93.97	0	95.18	96.86	93.88
breakdown	X	X	X	78.28	87.35	93.07
cable	83.33	86	0	84	89.25	88.84
capsule	85.60	68.18	0	81.81	82.15	86.72
carpet	100	100	0	100	98.54	98.05
fracture	X	X	X	69.51	43.84	48.85
grid	97.43	97.43	0	97.43	97.77	97.03
groove	X	X	X	71.14	60.28	62.28
hazelnut	96.36	97.27	0	97.27	96.66	98.88
leather	97.58	98.38	0	98.38	99.07	99.07
metal_nut	99.13	100	0	100	98.55	98.07
pill	87.42	85.62	0	87.42	89.93	90.58
screw	76.25	75	0	76.25	82.43	81
tile	99.14	99.14	0	94.01	96.51	97.01
toothbrush	100	95.23	0	90.47	86.11	93.05
transistor	84	82	0	83	85.71	86.42
wood	93.67	94.93	0	94.93	97.12	96.40
zipper	82.11	68.18	0	74.17	87.03	85.18
Image_AUROC	94.232	94.232	94.232	91.920	93.0984	93.0984
Pixel_AUROC	95.457	95.4573	95.457	95.3716	94.7474	94.7474
Precision	91.655	89.5601	0	85.8738	86.2155	87.8672

Table 5: Precision across Models & Classes

Class	Type	Score	Right	Wrong	Precision
bottle	Image_AUROC	99.52	78	5	93.97
	Pixel_AUROC	98.03			
cable	Image_AUROC	89.54	129	21	86
	Pixel_AUROC	93.59			
capsule	Image_AUROC	65.46	90	42	68.18
	Pixel_AUROC	89.8			
carpet	Image_AUROC	100.0	117	0	100
	Pixel_AUROC	99.4			
grid	Image_AUROC	99.79	76	2	97.43
	Pixel_AUROC	97.28			
hazelnut	Image_AUROC	99.46	107	3	97.27
	Pixel_AUROC	98.88			
leather	Image_AUROC	99.97	122	2	98.38
	Pixel_AUROC	99.31			
metal_nut	Image_AUROC	100.0	115	0	100
	Pixel_AUROC	90.43			
pill	Image_AUROC	95.34	143	24	85.62
	Pixel_AUROC	95.96			
screw	Image_AUROC	77.43	120	40	75
	Pixel_AUROC	96.27			
tile	Image_AUROC	99.82	116	1	99.14
	Pixel_AUROC	95.93			
toothbrush	Image_AUROC	97.78	40	2	95.23
	Pixel_AUROC	98.44			
transistor	Image_AUROC	91.85	82	18	82
	Pixel_AUROC	87.85			
wood	Image_AUROC	99.61	75	4	94.93
	Pixel_AUROC	95.95			
zipper	Image_AUROC	97.91	106	45	68.18
	Pixel_AUROC	94.74			

Table 6: Model's V2 Results

Class	Type	Score	Right	Wrong	Precision
bottle	Image_AUROC	99.52	0	83	0
	Pixel_AUROC	98.03			
cable	Image_AUROC	89.54	0	150	0
	Pixel_AUROC	93.59			
capsule	Image_AUROC	65.46	0	132	0
	Pixel_AUROC	89.8			
carpet	Image_AUROC	100.0	0	117	0
	Pixel_AUROC	99.4			
grid	Image_AUROC	99.79	0	78	0
	Pixel_AUROC	97.28			
hazelnut	Image_AUROC	99.46	0	110	0
	Pixel_AUROC	98.88			
leather	Image_AUROC	99.97	0	124	0
	Pixel_AUROC	99.31			
metal_nut	Image_AUROC	100.0	0	115	0
	Pixel_AUROC	90.43			
pill	Image_AUROC	95.34	0	167	0
	Pixel_AUROC	95.96			
screw	Image_AUROC	77.43	0	160	0
	Pixel_AUROC	96.27			
tile	Image_AUROC	99.82	0	117	0
	Pixel_AUROC	95.93			
toothbrush	Image_AUROC	97.78	0	42	0
	Pixel_AUROC	98.44			
transistor	Image_AUROC	91.85	0	100	0
	Pixel_AUROC	87.85			
wood	Image_AUROC	99.61	0	79	0
	Pixel_AUROC	95.95			
zipper	Image_AUROC	97.91	0	151	0
	Pixel_AUROC	94.74			

Table 7: Model's V3 Results

Class	Type	Score	Right	Wrong	Precision
ablation	Image_AUROC	85.94	161	99	61.92
	Pixel_AUROC	94.26			
bottle	Image_AUROC	99.52	79	4	95.18
	Pixel_AUROC	98.03			
breakdown	Image_AUROC	98.73	328	91	78.28
	Pixel_AUROC	99.3			
cable	Image_AUROC	89.54	126	24	84
	Pixel_AUROC	93.59			
capsule	Image_AUROC	65.46	108	24	81.81
	Pixel_AUROC	89.8			
carpet	Image_AUROC	100.0	117	0	100
	Pixel_AUROC	99.4			
fracture	Image_AUROC	68.48	333	146	69.51
	Pixel_AUROC	89.84			
grid	Image_AUROC	99.79	76	2	97.43
	Pixel_AUROC	97.28			
groove	Image_AUROC	79.86	249	101	71.14
	Pixel_AUROC	96.8			
hazelnut	Image_AUROC	99.46	107	3	97.27
	Pixel_AUROC	98.88			
leather	Image_AUROC	99.97	122	2	98.38
	Pixel_AUROC	99.31			
metal_nut	Image_AUROC	100.0	115	0	100
	Pixel_AUROC	90.43			
pill	Image_AUROC	95.34	146	21	87.42
	Pixel_AUROC	95.96			
screw	Image_AUROC	77.43	122	38	76.25
	Pixel_AUROC	96.27			
tile	Image_AUROC	99.82	110	7	94.01
	Pixel_AUROC	95.93			
toothbrush	Image_AUROC	97.78	38	4	90.47
	Pixel_AUROC	98.44			
transistor	Image_AUROC	91.85	83	17	83
	Pixel_AUROC	87.85			
wood	Image_AUROC	99.61	75	4	94.93
	Pixel_AUROC	95.95			
zipper	Image_AUROC	97.91	112	39	74.17
	Pixel_AUROC	94.74			

Table 8: Model's V4 Results

Class	Type	Score	Right	Wrong	Precision
ablation	Image_AUROC	85.94	170	90	65.38
	Pixel_AUROC	94.26			
bottle	Image_AUROC	99.52	216	13	94.32
	Pixel_AUROC	97.83			
breakdown	Image_AUROC	98.73	366	53	87.35
	Pixel_AUROC	99.3			
cable	Image_AUROC	92.27	216	26	89.25
	Pixel_AUROC	94.3			
capsule	Image_AUROC	72.46	198	43	82.15
	Pixel_AUROC	88.37			
carpet	Image_AUROC	99.34	203	3	98.54
	Pixel_AUROC	99.23			
fracture	Image_AUROC	68.48	210	269	43.84
	Pixel_AUROC	89.84			
grid	Image_AUROC	99.81	132	3	97.77
	Pixel_AUROC	97.19			
groove	Image_AUROC	79.86	211	139	60.28
	Pixel_AUROC	96.8			
hazelnut	Image_AUROC	99.68	174	6	96.66
	Pixel_AUROC	98.67			
leather	Image_AUROC	99.64	214	2	99.07
	Pixel_AUROC	99.4			
metal_nut	Image_AUROC	99.94	205	3	98.55
	Pixel_AUROC	89.96			
pill	Image_AUROC	96.99	277	31	89.93
	Pixel_AUROC	94.53			
screw	Image_AUROC	83.22	230	49	82.43
	Pixel_AUROC	96.77			
tile	Image_AUROC	99.83	194	7	96.51
	Pixel_AUROC	96.01			
toothbrush	Image_AUROC	98.19	62	10	86.11
	Pixel_AUROC	97.6			
transistor	Image_AUROC	95.93	120	20	85.71
	Pixel_AUROC	84.86			
wood	Image_AUROC	99.8	135	4	97.12
	Pixel_AUROC	94.58			
zipper	Image_AUROC	98.96	235	35	87.03
	Pixel_AUROC	90.70			

Table 9: Model's V5 Results

Class	Type	Score	Right	Wrong	Precision
ablation	Image_AUROC	85.94	195	65	75
	Pixel_AUROC	94.26			
bottle	Image_AUROC	99.52	215	14	93.88
	Pixel_AUROC	97.83			
breakdown	Image_AUROC	98.73	390	29	93.07
	Pixel_AUROC	99.3			
cable	Image_AUROC	92.27	215	27	88.84
	Pixel_AUROC	94.3			
capsule	Image_AUROC	72.46	209	32	86.72
	Pixel_AUROC	88.37			
carpet	Image_AUROC	99.34	202	4	98.05
	Pixel_AUROC	99.23			
fracture	Image_AUROC	68.48	234	245	48.85
	Pixel_AUROC	89.84			
grid	Image_AUROC	99.81	131	4	97.03
	Pixel_AUROC	97.19			
groove	Image_AUROC	79.86	218	132	62.28
	Pixel_AUROC	96.8			
hazelnut	Image_AUROC	99.68	178	2	98.88
	Pixel_AUROC	98.67			
leather	Image_AUROC	99.94	214	2	99.07
	Pixel_AUROC	99.4			
metal_nut	Image_AUROC	99.94	204	4	98.07
	Pixel_AUROC	89.96			
pill	Image_AUROC	96.99	279	29	90.58
	Pixel_AUROC	94.53			
screw	Image_AUROC	83.22	226	53	81
	Pixel_AUROC	96.77			
tile	Image_AUROC	99.83	195	6	97.01
	Pixel_AUROC	96.01			
toothbrush	Image_AUROC	98.19	67	5	93.05
	Pixel_AUROC	97.6			
transistor	Image_AUROC	95.93	121	19	86.42
	Pixel_AUROC	84.86			
wood	Image_AUROC	99.8	134	5	96.40
	Pixel_AUROC	94.58			
zipper	Image_AUROC	98.96	230	40	85.18
	Pixel_AUROC	90.70			

Table 10: Model's V6 Results

5.4.1 Loss Evolution

The plots of the loss evolution from each model are displayed below:



Figure 22: Model V2 - Loss Evolution

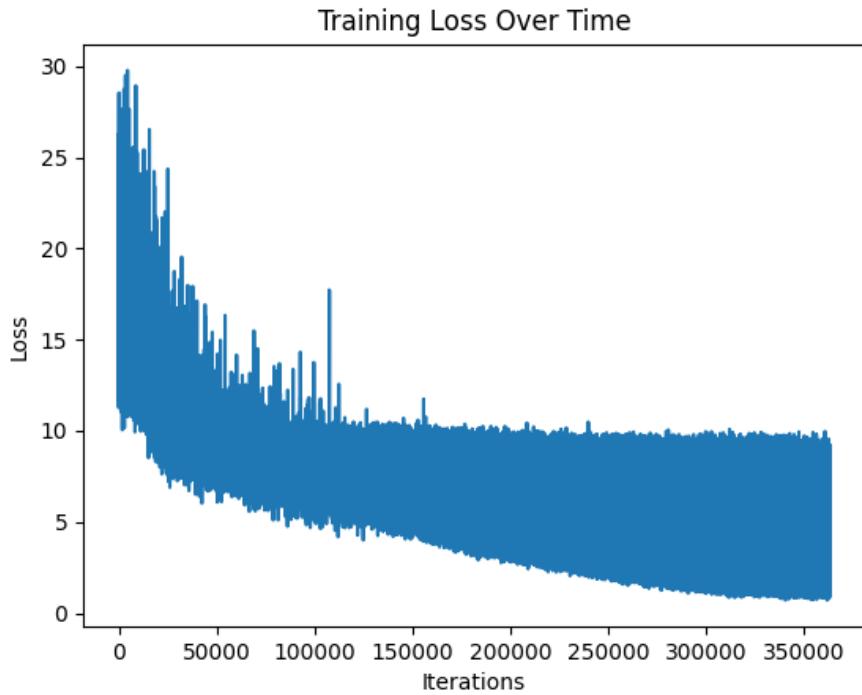


Figure 23: Model V3 - Loss Evolution



Figure 24: Model V4 - Loss Evolution

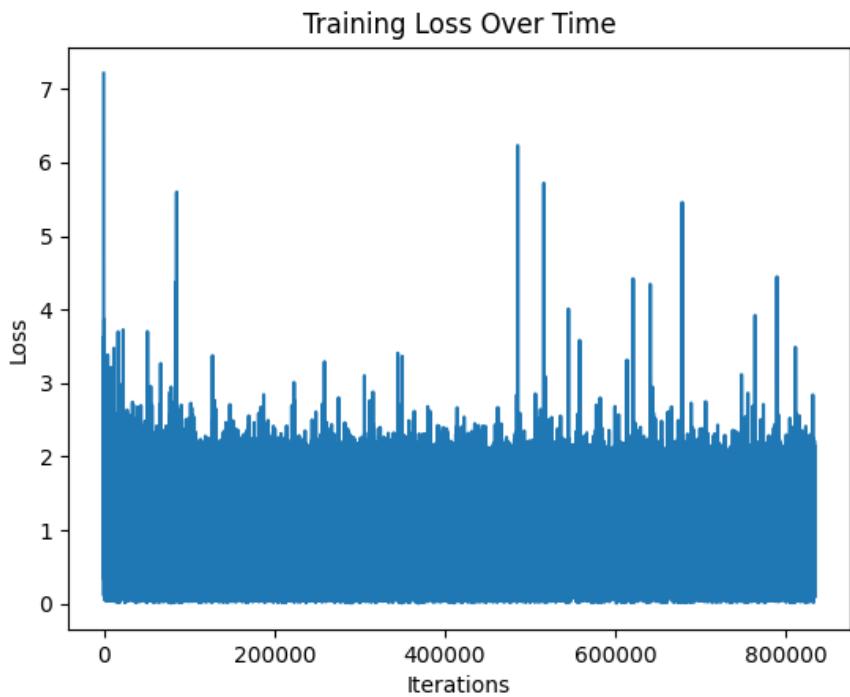


Figure 25: Model V5 - Loss Evolution



Figure 26: Model V6 - Loss Evolution

5.5 Extra-Information

5.5.1 PEFT

PEFT which stands for **Parameter Efficient Fine-Tuning**, is a technique used in the field of machine learning and natural language processing to **fine-tune large pre-trained models efficiently**. The **main goal** of PEFT is to **reduce the number of parameters that need to be updated during the fine-tuning process**, thus making the process more efficient in terms of both computation and memory usage.

PEFT methods aim to update only a small subset of the model's parameters during fine-tuning. This can be achieved through various techniques, such as:

- **Adapter Layers:** Adding small adapter modules to the pre-trained model. Only these adapter modules are trained, while the rest of the model parameters remain fixed.
- **Low-Rank Adaptation (LoRA):** Decomposing weight matrices into low-rank matrices and fine-tuning only these low-rank matrices.
- **Sparse Fine-Tuning:** Updating only a sparse subset of the model's parameters based on certain criteria.

5.5.2 LORA

LoRA, or Low-Rank Adaptation, is a method used in the context of fine-tuning large language models efficiently. It aims **to reduce the number of parameters that need to be adjusted during the fine-tuning process**, making it more computationally efficient and reducing the memory footprint. Let's see a detailed explanation of what LoRA is and how it works:

5.5.2.1 What is LoRA?

LoRA (Low-Rank Adaptation) is a technique that **adapts pre-trained language models by adding low-rank matrices to the existing model parameters**. The key idea is **to decompose the weight updates into low-rank matrices** during

fine-tuning, significantly reducing the number of parameters that need to be learned. The key concepts are the following:

1. Low-Rank Decomposition:

- Instead of fine-tuning the entire large model, LoRA introduces two low-rank matrices A and B such that $W' = W + AB$, where W is the original weight matrix, and AB is the low-rank approximation.
- Here, A and B are much smaller matrices (low-rank), which means they have fewer parameters compared to the original weight matrix W .

2. Parameter Efficiency:

- By decomposing the weight updates into low-rank matrices, the number of parameters that need to be trained during fine-tuning is significantly reduced.
- This makes the fine-tuning process much more efficient in terms of computational resources and memory usage.

3. Maintaining Model Performance:

- Despite the reduction in the number of trainable parameters, LoRA can maintain or even improve the performance of the model on specific tasks because the low-rank matrices can effectively capture the necessary adaptations without modifying the entire weight matrix.

5.5.2.2 How LoRA Works?

1. **Initialization:** Start with a pre-trained language model. The existing weight matrices W of the model are kept frozen (i.e., not updated during fine-tuning).
2. **Adding Low-Rank Matrices:** For each weight matrix W that needs to be fine-tuned, introduce two trainable low-rank matrices A and B .
3. **Fine-Tuning:**
 - During fine-tuning, only the low-rank matrices A and B are updated. The original weight matrix W remains unchanged.
 - The effective weight matrix during fine-tuning becomes $W' = W + AB$.
4. **Forward Pass:** In each forward pass, the model uses the effective weight matrix W' , which incorporates the low-rank adaptation.
5. **Parameter Update:** Only the parameters in matrices A and B are updated during backpropagation.

5.5.2.3 Advantages of LoRA

1. **Efficiency:** Reduces the number of trainable parameters, leading to faster and more memory-efficient fine-tuning.
2. **Flexibility:** Can be applied to various parts of the model without altering the original architecture.
3. **Performance:** Maintains or even improves the model's performance on specific tasks by effectively capturing necessary adaptations.

5.5.2.4 Example in Practice

Here's a conceptual example using PyTorch-like pseudocode to illustrate the application of LoRA:

```
import torch
import torch.nn as nn

class LoRAAdaptedLayer(nn.Module):
    def __init__(self, original_layer, rank):
        super(LoRAAdaptedLayer, self).__init__()
        self.original_layer = original_layer
        self.rank = rank
        self.A = nn.Parameter(torch.randn(original_layer.weight.size(0), rank))
        self.B = nn.Parameter(torch.randn(rank, original_layer.weight.size(1)))

    def forward(self, x):
        W = self.original_layer.weight
        W_prime = W + torch.matmul(self.A, self.B)
        return nn.functional.linear(x, W_prime, self.original_layer.bias)

# Example usage
original_linear = nn.Linear(512, 512)
lora_layer = LoRAAdaptedLayer(original_linear, rank=4)

# During fine-tuning, only lora_layer.A and lora_layer.B will be updated
```

5.5.3 ML Concepts

5.5.3.1 Pad Token

The **pad_token** is a special token used to pad sequences to ensure they are all of the same length. This is particularly useful when batching sequences together for processing, as many neural network models require input sequences to be of uniform length. Suppose you have the following sequences of tokens:

- Sequence 1: ["The", "quick", "brown", "fox"]
- Sequence 2: ["jumps", "over", "the", "lazy", "dog"]

When batching these sequences together, you need them to have the same length. If the maximum length is 5, you might pad Sequence 1 to match this length:

- Padded Sequence 1: ["The", "quick", "brown", "fox", "< pad >"]
- Sequence 2: ["jumps", "over", "the", "lazy", "dog"]

5.5.3.2 Eos Token

The **eos_token** (end-of-sequence token) is a special token that indicates the end of a sequence. It is used by the model to know where the sequence ends, which is particularly important during generation tasks or when processing variable-length sequences. If you have a sentence, such as the following, you might append an eos_token to signify the end of this sequence:

- Sentence: ["The", "quick", "brown", "fox"]
- With EOS: ["The", "quick", "brown", "fox", "< eos >"]

5.5.3.3 Padding Side

The `padding_side` specifies where the padding should be applied in a sequence: either at the beginning (left) or the end (right). Using the previous sequences:

- Sequence 1: [”The”, ”quick”, ”brown”, ”fox”]
- Sequence 2: [”jumps”, ”over”, ”the”, ”lazy”, ”dog”]

When padding these sequences to a length of 6 and setting `padding_side` to ”right”, the result will be:

- Padded Sequence 1: [”The”, ”quick”, ”brown”, ”fox”, ”< pad >”, ”< pad >”]
- Sequence 2: [”jumps”, ”over”, ”the”, ”lazy”, ”dog”]

5.5.3.4 Residual Connection

A residual connection adds the input of a layer to its output, helping gradients flow more easily through the network. Let’s consider the following simplified example:

- Input: $x = [1, 2, 3]$
- Layer’s transformation $\mathcal{F}(x)$: Multiply by 2
- Output of the layer without residual connection: $\mathcal{F}(x) = [2, 4, 6]$

Now, with residual connection, it would output:

$$\text{output} = \mathcal{F}(x) + x = [2, 4, 6] + [1, 2, 3] = [3, 6, 9]$$

5.5.3.5 Batch Normalization

Batch normalization normalizes the input to each layer so that the inputs have a mean μ of 0 and a standard deviation σ of 1. This normalization is performed on a batch-by-batch basis. For instance, if the input batch is

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

For each feature (column):

1. Compute the mean and variance across the batch:

- Feature 1: mean = $\frac{1+4+7}{3} = 4$, variance = $\frac{(1-4)^2+(4-4)^2+(7-4)^2}{3} = \frac{18}{3} = 6$
- Feature 2: mean = $\frac{2+5+8}{3} = 5$, variance = $\frac{(2-5)^2+(5-5)^2+(8-5)^2}{3} = \frac{18}{3} = 6$
- Feature 3: mean = $\frac{3+6+9}{3} = 6$, variance = $\frac{(3-6)^2+(6-6)^2+(9-6)^2}{3} = \frac{18}{3} = 6$

2. Normalize:

- Feature 1: normalized = $\frac{X[:,0]-4}{\sqrt{6}} = \left[\frac{1-4}{\sqrt{6}}, \frac{4-4}{\sqrt{6}}, \frac{7-4}{\sqrt{6}} \right] = \left[-\frac{3}{\sqrt{6}}, 0, \frac{3}{\sqrt{6}} \right]$
- Feature 2: normalized = $\frac{X[:,1]-5}{\sqrt{6}} = \left[-\frac{3}{\sqrt{6}}, 0, \frac{3}{\sqrt{6}} \right]$
- Feature 3: normalized = $\frac{X[:,2]-6}{\sqrt{6}} = \left[-\frac{3}{\sqrt{6}}, 0, \frac{3}{\sqrt{6}} \right]$

The normalized batch is:

$$\text{normalized_}X = \begin{bmatrix} -\frac{3}{\sqrt{6}} & -\frac{3}{\sqrt{6}} & -\frac{3}{\sqrt{6}} \\ 0 & 0 & 0 \\ \frac{3}{\sqrt{6}} & \frac{3}{\sqrt{6}} & \frac{3}{\sqrt{6}} \end{bmatrix}$$

Now, let's say we want to combine Residual Connection and Batch Normalization at the same time.

1. Input to the Layer:

$$x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

2. Layer's Transformation $\mathcal{F}(x)$: Assume the layer simply multiplies the input by 2.

$$\mathcal{F}(x) = 2 \times x = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

3. Add Residual Connection:

$$\text{Residual Output} = \mathcal{F}(x) + x = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \\ 21 & 24 & 27 \end{bmatrix}$$

4. Apply Batch Normalization: For each feature compute mean and variance, then normalize.

- Mean: $\mu = [12 \ 15 \ 18]$
- Variance: $\sigma^2 = [54 \ 54 \ 54]$

$$\bullet \text{ Normalized Output} = \frac{\text{Residual Output} - \mu}{\sqrt{\sigma^2}} = \frac{\begin{bmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \\ 21 & 24 & 27 \end{bmatrix} - [12 \ 15 \ 18]}{\begin{bmatrix} \sqrt{54} & \sqrt{54} & \sqrt{54} \end{bmatrix}} = \begin{bmatrix} -\sqrt{\frac{3}{2}} & -\sqrt{\frac{3}{2}} & -\sqrt{\frac{3}{2}} \\ 0 & 0 & 0 \\ \sqrt{\frac{3}{2}} & \sqrt{\frac{3}{2}} & \sqrt{\frac{3}{2}} \end{bmatrix}$$

In practice, integrating these techniques into the neural network can significantly improve its performance and stability.