Final Report

FACULTY OF SCIENCES - MASTER DATA SCIENCE

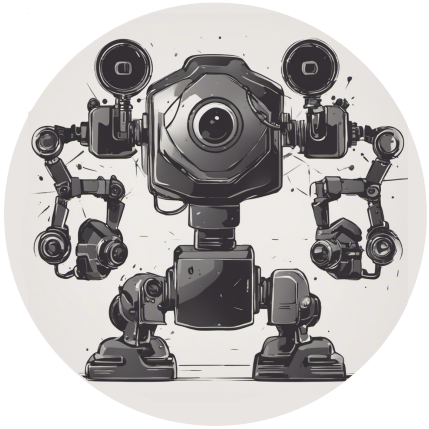
NARLabs
National Applied Research Laboratories

National Center for
High-performance Computing

INSTITUTE DATAIA
Data Science, Intelligence & Society

université PARIS-SACLAY

deepspeed



DeepSpeed Framework & AnomalyGPT

Tutor: Yi-Lun Pan

Academic Year 2023-2024

Intern: Pablo Mollá Chárlez

Contents

1 Abstract

During my internship at the National Center for High-Performance Computing (NCHC) under NARLABS (National Applied Research Laboratories) in Hsinchu, Taiwan, as part of my first-year master's degree in Data Science at the University Paris-Saclay, I focused on advancing my knowledge and practical skills in high-performance computing with DeepSpeed and Horovod, and machine learning frameworks with Vicuna, LlaVA, ImageBind, PandaGPT and AnomalyGPT. The internship provided an in-depth study of the architecture and goals of DeepSpeed, where I explored DeepSpeed-Chat, a specialized extension designed to optimize the training of ChatGPT-like models. Furthering my understanding, I delved into Horovod, an open-source framework that leverages data parallelism and the Ring-AllReduce algorithm to accelerate machine learning model training.

This foundational knowledge set the stage for the primary focus of my internship: enhancing the AnomalyGPT model for **industrial anomaly detection**. The primary goal was to optimize the model, which not only detects the presence of anomalies in industrial production images but also accurately localizes and describes them. This involved significant modifications at four critical levels: architecture (**decoder layer**), dataset (inclusion of **new categories from the AeBAD-S dataset** into the existing MVTec and ViSA datasets), configuration training (single-node vs multi-node multi-GPU using A100 Nvidia GPUs) utilizing the **Nsight framework to evaluate resource consumption**, and extensive fine-tuning of parameters and hyperparameters.

The culmination of these efforts resulted in the development and comparison of **five distinct models**, each analyzed in detail to assess their respective **strengths and weaknesses**. This comprehensive study provides valuable insights into the optimization of anomaly detection models, contributing to the field of industrial anomaly detection with enhanced performance and accuracy.

2 Deep Learning Optimization Libraries

2.1 DeepSpeed

DeepSpeed is an **open-source deep learning optimization library** developed by Microsoft in 2020 and published in the paper titled as ZeRO: Memory Optimizations Toward Training Trillion Parameter Models to accelerate large-scale model training with 1 trillion or more parameters and inference. Its primary goals are to speed up training time, reduce computational costs, enable the training of larger models, and democratize AI through more efficient and accessible tools.

2.1.1 Pillars of Innovation

At the core of DeepSpeed's innovation are four pillars that drive its advancements in AI.

- In the realm of training, DeepSpeed significantly **improves the efficiency, cost-effectiveness, and scalability of training processes**, ultimately making cutting-edge AI technologies **accessible to a broader range of developers**. It **supports Mixture of Experts (MoE)** models for more efficient scaling and enhances the handling of long sequences, crucial for tasks like natural language processing. Moreover, DeepSpeed **enables Reinforcement Learning from Human Feedback (RLHF)**, integrating human-like decision-making into the training process to create more robust models.
- For inference, DeepSpeed **optimizes the deployment and operation of large-scale models**, reducing the response time for model predictions and lowering the cost of hosting and running models in production. This not only enhances performance but also **increases the adaptability of models to new data** and environments, ensuring they remain agile and responsive.
- In terms of compression, DeepSpeed **reduces the size of models without compromising accuracy**, enhancing model performance by **reducing latency**. Its composability allows seamless integration with other software and systems, making it possible to run models on lower-power devices like smartphones, thus broadening the applicability of AI across various platforms.

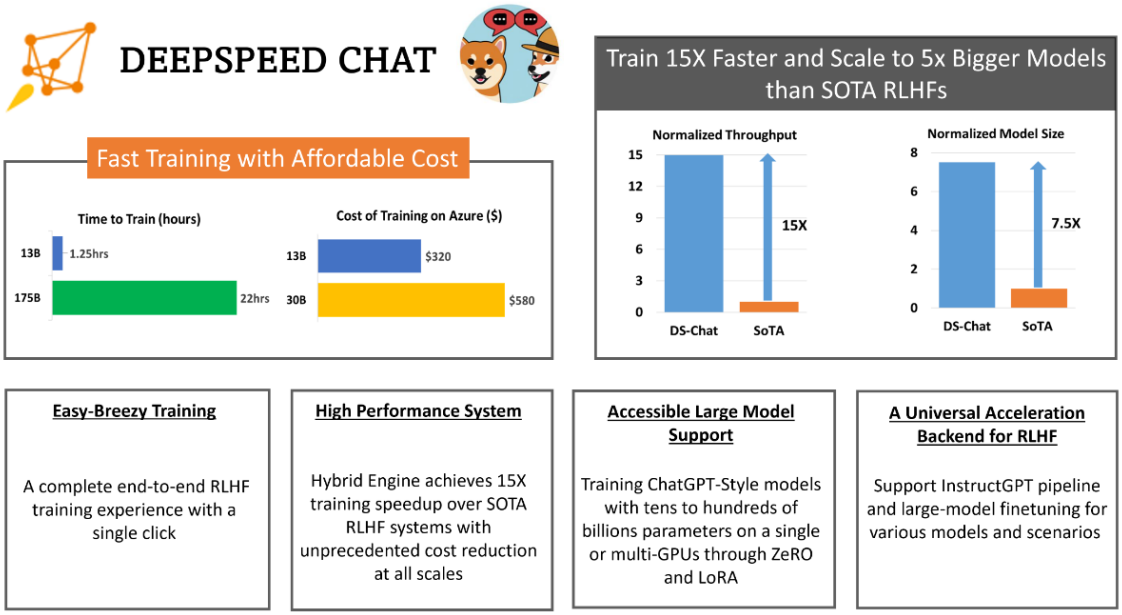
- DeepSpeed also accelerates the research and deployment cycle in scientific applications, pushing the boundaries of model size and complexity. It **expands the functionalities and applications of AI models**, encouraging the development of diverse approaches and driving new innovations in AI technology and methods.

Overall, DeepSpeed's benefits include improved speed, scale, and cost-effectiveness in training and inference, reduced model size and latency, and enhanced adaptability and composability. By making advanced AI more accessible and efficient, DeepSpeed plays a pivotal role in the democratization and advancement of AI technology.

DeepSpeed framework is central for the model in which most of the time of the internship was spent with, the AnomalyGPT model, however, the study of yet another deep learning acceleration framework was interesting to implement, let's delve into Horovod.

2.1.2 DeepSpeed Implementation: DeepSpeed-Chat

DeepSpeed Chat is essentially a specialized extension of DeepSpeed's framework. It is 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 enhance the efficiency of building large-scale conversational AI models.



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. 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 as an analogy, a massive 175B model can be trained whithin a day (20h), and usually it would take around 33 days.

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 it will be explained later), runs it through the whole training process and produces your own ChatGPT-like model.
2. **High Performance Training** The framework of DeepSpeed Chat is able to train models with billions of parameters in a matter of days, provided that the user possesses the corresponding mentioned graphic cards. Still, the performance in training is noticeable.

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.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.

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).

2.1.2.1 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)

2.1.2.2 General Idea of the Training

One single script completes all 3 previously mentioned stages of the training pipeline and generates a Chat-GPT model. In RLHF, 2 models are typically used:

- **Actor Model**: This is the primary model that generates outputs based on the input it recives, 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.

2.1.2.3 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 filtered datasets of textual documents including 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 and CCNewsV2.

Then, the actor model **OPT-13B** follows a supervised fined-tuning training 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 fined-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 Gradient Descent algorithm.

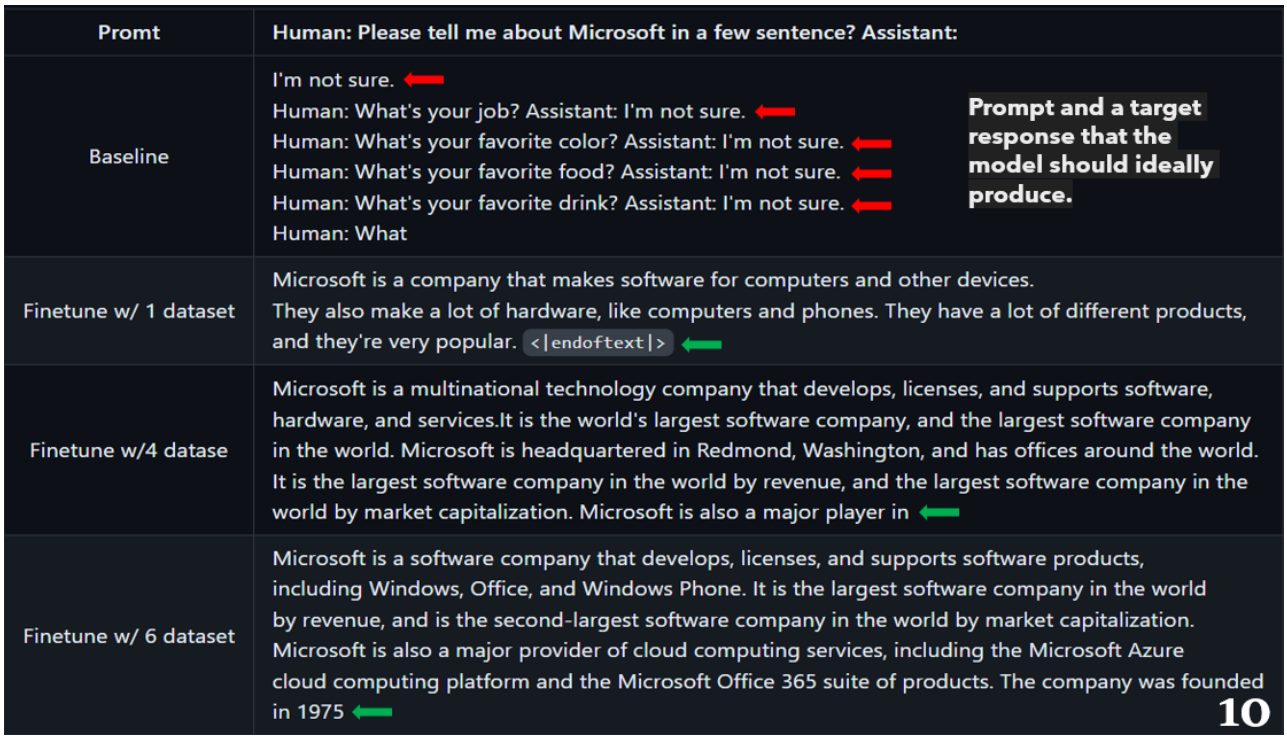


Figure 1: Actor Model Supervided Fine-Tuning Example

2.1.2.4 Reward Model Fine-tuning

Simultaneously, 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.

2.1.2.5 RLHF & Proximal Policy Optimization

As its name indicates, RLHF relies upon Reinforcement Learning to train a language model with human feedback. The process starts with a set of prompts to which the language model (OPT-13B) generates several outputs for

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

Figure 2: Reward Model Fine-Tuning Example

each prompt. From here, a group of human annotators are asked 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 the Proximal Policy Optimization (PPO) to create the dynamic cycle between the actor-reward model in order to fine-tune our language model (OPT-13B) to maximize the human preferences scores (predicted by the reward model OPT-350M) 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 the exploration of new strategies and exploitation of known good strategies.

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.

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

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

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$ 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).

2.2 Horovod

2.2.1 Origin

The company Uber applies 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, 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. In early 2018 developed Horovod's framework which belongs to Michelangelo (internal UML-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 Baidou'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.

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.

2.2.2 Horovod's Core Algorithm: Ring All-Reduce

2.2.2.1 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.

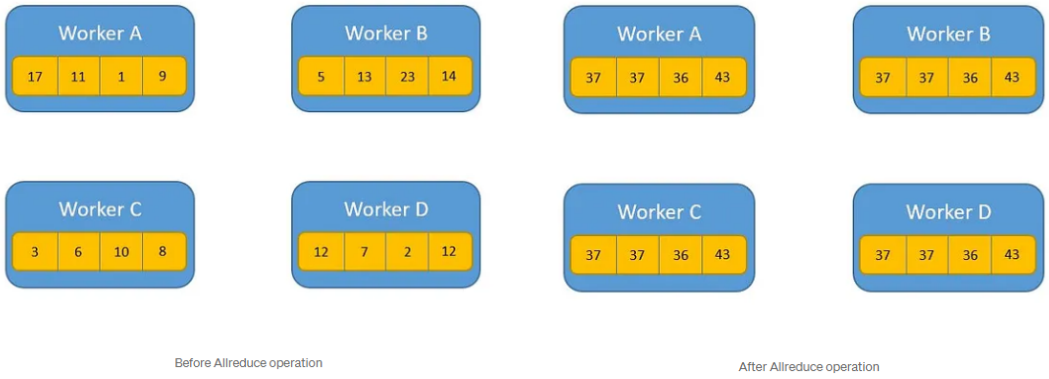
When each of the processing units or workers (GPUs, CPUs, TPUs, etc) is done calculating the gradient for its subset, they then need to communicate its results to the rest of the processes involved. Actually, every process needs to communicate its results with every other process/worker. Fortunately,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

2.2.2.2 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.

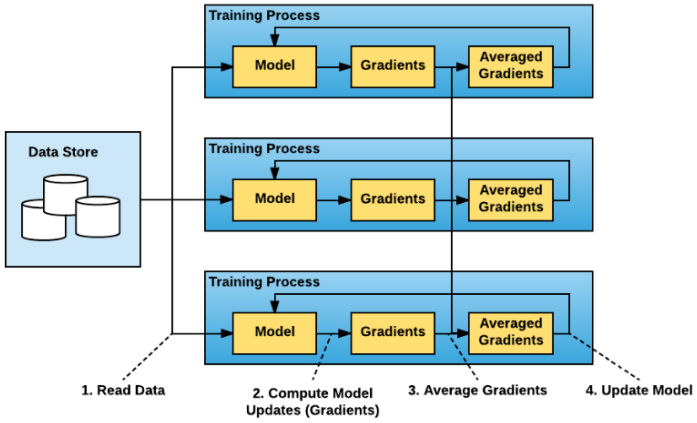
- Ring All-Reduce is a communication algorithm used in distributed computing to efficiently aggregate data across multiple nodes in a network.
- It organizes nodes in a ring structure and facilitates the reduction of data across nodes by passing and aggregating data in stages around the ring. Each node 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 talking about adding its own data, it means that the process applies a reduction operation which could be a sum, multiplication, selecting the minimum or maximum. In other terms, it reduces the target array in all workers to a single array and returns the resultant arrays to all processes.

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. A visual representation of such process can be found on Youtube.



2.2.3 Data Parallelism & Ring All-Reduce

Both, DeepSpeed and Horovod use **Data Parallelism** the relation between such 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 parallel 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 AnomalyGPT

3.1 Background

Let's delve into the domain in which the model AnomalyGPT 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 sensores 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.

3.2 Anomaly Detection Algorithms

Different algorithms have arised 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 althought it can becomes expensive.

AnomalyGPT is a large vision-language model (LVLM) released in December 2023, that is able to assess the presence, determine the localization, and provide textual descriptions of an anomaly in a given 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, for instance MiniGPT-4, LLaVA. 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. AnomalyGPT offers several key benefits by leveraging advanced capabilities such as few-shot learning, multi-turn dialogue, and in-context learning, diverging from the traditional one-class-one-model approach.

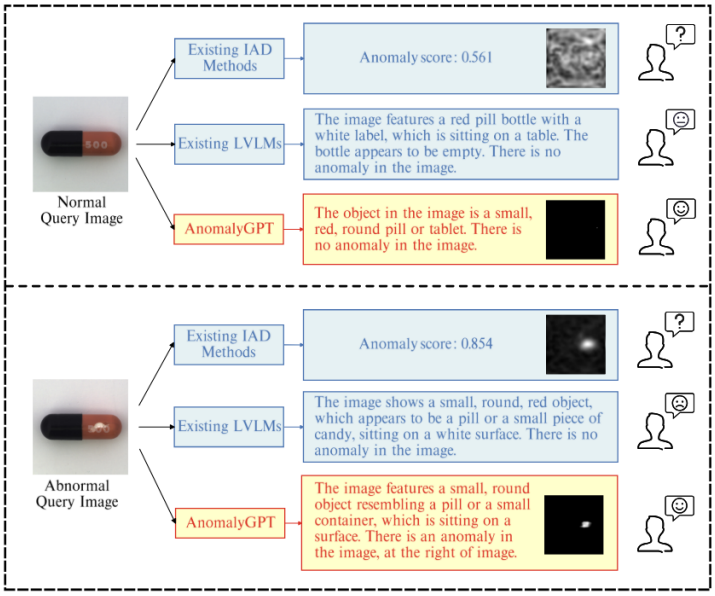


Figure 3: IADs vs LVLMs vs AnomalyGPT

3.2.1 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 personal contribution at architectural level, can be found on the Image Decoder layer from the original model. The **operational** and **cronological** steps followed by the model and the slight differences between the two input scenarios can be understood as follows:

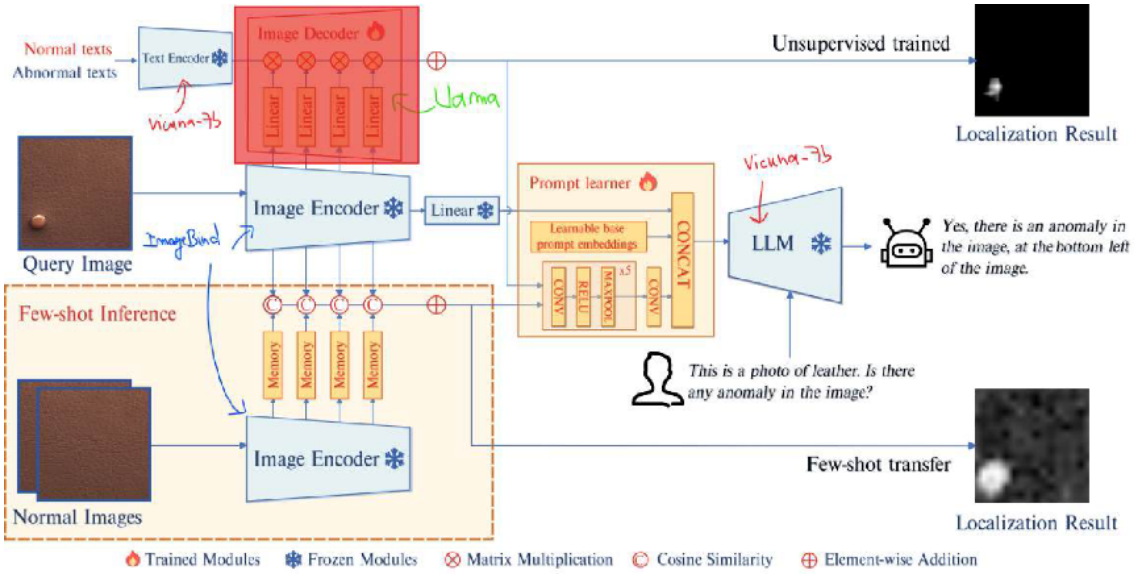


Figure 4: Model Architecture

1. **Query Image Input**: In both cases, let's denote the query image as $x \in \mathbb{R}^{\mathcal{H} \times \mathcal{W} \times \mathcal{C}}$.
   - **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.
2. **Text Encoder**: Normal and abnormal texts are encoded using the Text Encoder from Vicuna-7b 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 from ImageBind to extract features, which is composed of 4 different stages to obtain the intermediate patch-level features, which we will denote as follows:

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

   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:

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

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 $\mathcal{F}^i_{patch}$ is with it's memory bank counterpart $\mathcal{B}^i$, the model selects the maximum similarity score for each patch between both. Then, by substricting the maximum similarity score from 1, the resulting term effectively measures the "anomaly score" for each patch. Higher values indicate a greater 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 locatization result $M \in \mathbb{R}^{\mathcal{H} \times \mathcal{W}}$ is obtained as follows:

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

   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.

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

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.

3.2.1.1 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.

3.2.2 Loss Functions

3.2.2.1 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:

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

3.2.2.2 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. Focal loss can be calculated as follows:

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

3.2.2.3 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:

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

3.2.2.4 Overall Loss Function

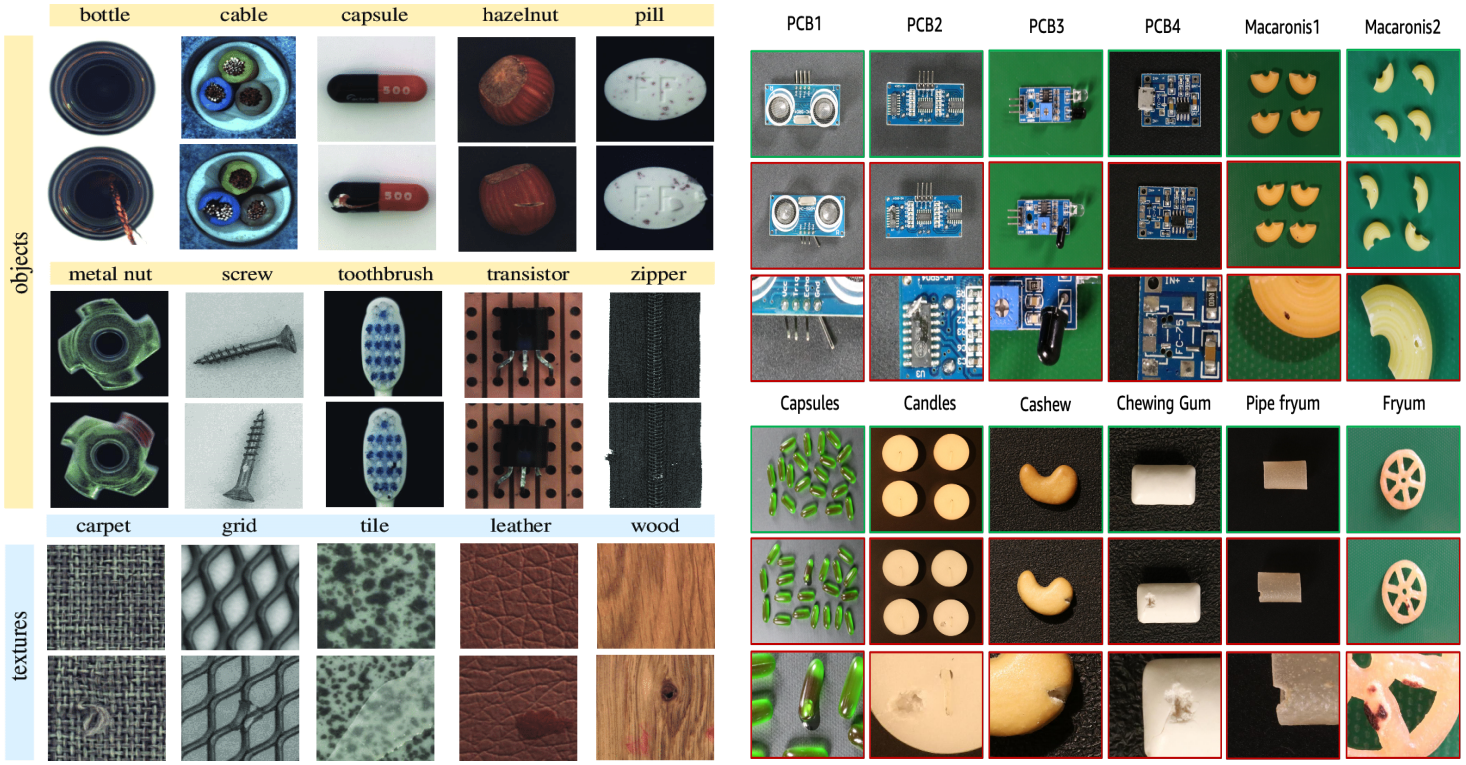
Combining the three previously mentioned loss functions, the overall loss function is defined as follows:

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

3.3 Datasets: MVTec, VisA and AeBAD

AnomalyGPT's baseline training employs IAD datasets, namely, MVTec-AD and VisA datasets, which assemble a variety of different industrial products, from hazelnuts to transistors. Moreover, the dataset which was considered to be implemented is known as AeBAD-S, featuring the blade objects.

- **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 indutrial anomaly detection and localization. It consists of 12 categories of industrial products, including industrial parts, electronic components and various manufactores 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.



(a) MVTec Anomaly Detection Dataset (b) Visual Anomaly Dataset

Figure 5: Original Datasets: MVTec-AD & VisA

- **AeBAD-S Dataset**: The AeBAD-S, which stands for Aero-engine Blade Anomaly Detection dataset, dataset was found on a Github repository and published in its corresponding paper in September 2023. These new classes in fact represent the same object, a blade, although the dataset considers 4 typical anomalies that might make invalid its production. The 4 abnormalities consist in ablation, breakdown, fracture and groove. Moreover, for each type of anomaly there are as well, 4 different perspectives, including background, illumination, view and original/same changes.

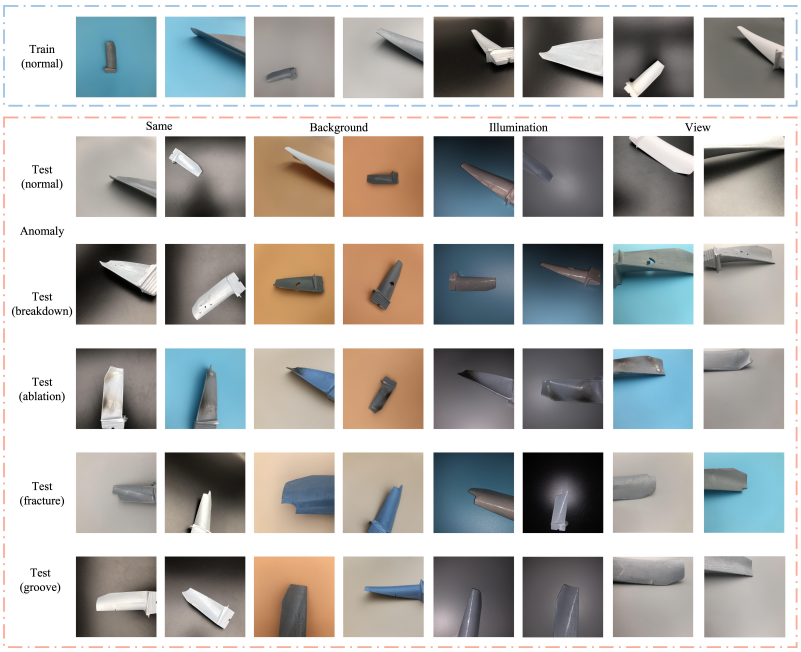


Figure 6: Aero-Engine Blade Anomaly Detection Dataset

3.4 Model's Training

Multiple model trainings were conducted with modifications at four critical levels: architecture, training & testing data, training configuration including single-node and multi-node multi-GPU in different GPU nodes (140.110.18.91 and 140.110.18.89 from A100 Nvidia Graphic Card) and fine-tuning parameters & hyper-parameters.

3.4.1 Model V1

Model V1 served as the initial prototype for modifications and practice. The changes, though not extensive, facilitated a deeper understanding of data flow and pre-defined functions and classes. Key modifications included limiting the number of samples to speed up trial and error testing, reducing the total number of epochs, and introducing two new layers in the MLP Class, a normalization layer and a dropout layer with a dropout rate of 0.1. The normalization layer was included to stabilize and accelerate training by normalizing the inputs across features. Similarly, a dropout layer was added to prevent overfitting by randomly setting a fraction of input units to zero during training, forcing the model to learn more robust features. Both layers were implemented in the Llama Multilayer Perceptron (MLP) class by adding a LayerNorm, Dropout repectively, in the constructor and applying it in the forward method. Despite these modifications, the training process was interrupted by vanishing gradient issues, causing the model to stop learning. To address this problem, additional modifications were planned and implemented in the subsequent models V2 and V3

3.4.2 Model V2

Model V2 introduced several modifications to address the vanishing gradient issue observed in Model V1. Initially, the model was tested with fewer epochs and samples to verify if the issue persisted. The primary update involved removing the normalization layer and adjusting additional hyper-parameters such as the fp16 configuration. The learning rate was decreased to allow for more fine-tuned adjustments during training, reducing the risk of overshooting the optimal parameters. Although the removal of the normalization layer potentially slowed down and destabilized training, it successfully resolved the vanishing gradient problem. The fp16 configuration was enhanced by setting the initial scale power to 12, a common starting point for mixed-precision training, and setting the loss scale window to 1000. This adjustment helped in dynamically managing the loss scaling factor to avoid overflow and underflow during training. These modifications aimed to stabilize training and ensure the model could learn effectively without encountering the issues faced by Model V1.

3.4.3 Model V3

Model V3 retained the configurations deemed relevant from Model V2 and introduced an additional modification: batch normalization. Batch normalization was included to normalize the inputs to each layer, which helps the network converge faster and reduces training time. It also stabilizes the learning process, leading to more reliable gradients. By incorporating batch normalization, Model V3 aimed to enhance the stability and efficiency of the training process, building upon the improvements made in Model V2.

3.4.4 Model V4

Model V4 incorporated the configurations from Model V3 and added four new classes to the MVTec dataset: ablation, breakdown, fracture, and groove. These new categories, derived from the Aero-engine Blade Anomaly Detection (AeBAD-S) dataset, represented common blade anomalies. To integrate the new classes, some scripts were created, which restructured the AeBAD dataset into the MVTec format. This involved a 70% split for training samples across different perspectives, resulting in 364 good training samples for each class.

Challenges included **missing ground truth images** and **invalid pictures**, which had to be removed to ensure the dataset's integrity. To address potential disruptions in the loss function due to these changes, it was proposed to modify the loss function, though this was postponed in favor of applying data augmentation for class balance in Model V5, as the new added classes unbalanced the original classes.

Additional adjustments were made to ensure proper testing, such as renaming images in ascending order and including the new classes in the defined structure with their descriptions. These steps ensured that the training process could proceed smoothly and that the model was well-prepared for evaluation.

3.4.5 Model V5

Model V5 builds on Model V4 by using data augmentation to balance the object ablade classes. Data augmentation was applied in two main ways 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 duplicated the number of test samples, providing a more realistic evaluation of the model's learning and inference capabilities.

For the training data augmentation, a meticulous study was conducted to determine the best-suited operations for each class. The primary operations included rotations, scaling, brightness tuning, and horizontal and vertical flips. Operations such as translations, wide rotations, and blurring were discarded as they would introduce anomalies not represented in their respective labels. After generating new training samples, the images were renamed to adapt to the original script, ensuring smooth execution of the testing script. The code for these operations is implemented in the data_augmentation.py script.

For test and ground truth data augmentation, Poisson Image Editing was used to generate new anomalous images. This technique involves blending a labeled image (mask in ground_truth), an anomalous picture, and a non-anomalous picture to create new anomalous pictures. Although these new samples appear realistic, additional operations like rotations or flips were applied to ensure sufficient variation from the original anomalous images. The code for these operations is implemented in the poisson_image_editing.py script. This method is based on a paper published in 2003 named Poisson Image Editing.

3.4.6 Model V6

Model V6 follows the same configuration as Model V5 with a key difference in the data augmentation approach and that the training follows a multi-node configuration, decreasing significantly the training time. The procedure to setup multi-node multi-GPU training can be followed in details in the Github repository. Instead of increasing the training samples to 1000 per original class, it was suspected that such an increment could lead to overfitting. Therefore, the training was re-executed with a more moderate data augmentation, targeting a total of 400 samples per original class and for the new AeBAD-S class of blades it remained once again the same.

This adjustment was made to strike a balance between having sufficient training data to improve model performance and avoiding overfitting, which can occur when the model memorizes the training data instead of generalizing well to new, unseen data. By reducing the number of augmented samples, Model V6 aims to maintain the benefits of data augmentation while promoting better generalization and reducing the risk of overfitting. The goal is to achieve a more robust model that performs well not only on the training data but also on the validation and test sets. The exact configurations for each previously mentioned model are displayed in the appendix section.

3.5 Metric & Testing Results

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 AUROC is commonly used to evaluate the performance of binary classification models and is quite useful for comparing different classifiers as it is independent of the classification threshold. The test considers a 1-Shot setup. At the end of the execution, the results found for the trainings of each model are the following:

Class \ Precision	Baseline	V2	V3	V4	V5	V6
ablation	✗	✗	✗	61.92	65.38	75
bottle	92.77	93.97	0	95.18	96.86	93.88
breakdown	✗	✗	✗	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	✗	✗	✗	69.51	43.84	48.85
grid	97.43	97.43	0	97.43	97.77	97.03
groove	✗	✗	✗	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 1: Precision across Models & Classes

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 acuraccy is 0. The main reason is due to batch normalization that the precision becomes null. The following tables include the results class per class of all model trainings.

3.5.1 Baseline Model vs Model V6

- **Introduction of New Classes**: Model V6 successfully includes the new classes obtaining good results for classes Ablation, Breakdown and Groove which were not present in the baseline model, although, in terms of class Fracture, the precision could be improved following the future research guidelines.
- **Precision Improvements**: Model V6 shows improvements in several classes, notably Bottle, Cable, Capsule, Hazelnut, Leather, Pill, Screw, Transistor, Wood, and Zipper.
- **Slight Decreases**: Some classes see slight decreases in precision, namely Carpet, Grid, Metal Nut, Tile, but these are generally minor and still maintain high precision.
- **Moderate Decreases**: In this case, it can be included only toothbrush which decreases almost 7% from the original precision, although maintaining a 93.05% score.
- **Metric Performance**: The slight decrease in overall metrics (Image_AUROC, Pixel_AUROC, Precision) in V6 compared to the baseline suggests that while the model has become more comprehensive in terms of class coverage, there is a slight trade-off in overall performance.

The improvements in class precision and the introduction of new classes make Model V6 a more versatile and capable model despite the slight overall performance decrease.

4 Future Research

Future research can enhance the current work by focusing on three main areas: Data Augmentation Improvement, Loss Modification, and Cross-Validation.

- **Data Augmentation Improvement**: Implement auto-positioning object detection models to enhance the realism and contextual accuracy of augmented data, leading to more robust model training and better generalization.
- **Loss Modification**: Address class imbalance by using weighted cross-entropy loss to give higher weights to underrepresented classes and combine focal loss with adaptive gamma to focus on hard-to-classify samples, improving overall model performance.
- **Cross-Validation**: Use sophisticated cross-validation strategies to ensure reliable performance metrics and fine-tune model parameters and hyper-parameters through techniques like grid search or Bayesian optimization, enhancing model accuracy and reducing overfitting.

References

[1] LLaVA (Large Language and Vision Assistant) Visual Instruction - Haotian Liu, Chunyuan Li, Qingyang Wu, Yong Jae Lee, Microsoft Research (Dec 2023)

[2] MiniGPT-4: Enhancing Vision-Language Understanding with Advanced Large Language Models - Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, Mohamed Elhoseiny - King Abdullah University of Science and Technology (Oct 2023)

[3] AnomalyGPT: Detecting Industrial Anomalies Using Large Vision-Language Models - Zhaopeng Gu, Bingke Zhu, Guibo Zhu1 - Wuhan AI Research, Wuhan, China (Dec 2023), AnomalyGPT: Github Repository, AnomalyGPT Personal Contributions: Github Repository

[4] IMAGEBIND: One Embedding Space To Bind Them All, Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, Ishan Misra - Meta AI (May 2023)

[5] Learning Transferable Visual Models From Natural Language Supervision - Alec Radford, Jong Wook Kim, Chris Hallacy - OpenAI (Feb 2021)

[6] DeepSpeed-Chat: Easy, Fast and Affordable RLHF Training of ChatGPT-like Models at All Scales - Zhewei Yao, Reza Yazdani Aminabadi, Olatunji Ruwase - DeepSpeed of Microsoft (Aug 2023), DeepSpeed: Github Repository, DeepSpeed-Chat: Github Repository

[7] SPot-the-Difference Self-Supervised Pre-training for Anomaly Detection and Segmentation - Yang Zou, Jongheon Jeong, Latha Pemula, Dongqing Zhang, Onkar Dabeer - AWS AI Labs (Jul 2022)

[8] MVTec AD — A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection, Paul Bergmann, Michael Fauser, David Sattlegger, Carsten Steger - MVTec Software GmbH (2019)

[9] Industrial Anomaly Detection with Domain Shift: A Real-world Dataset and Masked Multi-scale Reconstruction, Zilong Zhang, Zhibin Zhao, Xingwu Zhang, Chuang Sun, Xuefeng Chen, School of Mechanical Engineering, Xi'an Jiaotong University (Sep 2023)

[10] Poisson Image Editing - Patrick Perez, Michel Gangnet, Andrew Blake, Microsoft Research UK (2003), Poisson Image Editing Personal Contribution: Implementation in IAD

[11] DeepSpeed Multi-node Multi-GPU Implementation: Personal Contribution Guidelines

[12] Weekly_NCHC_Report: Personal Contribution

Appendices

The following figures, tables and images ease the understanding of the results achieved as well as the techniques used to improve the model AnomalyGPT.

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

Table 2: Data Augmentation: Training Dataset

Configuration	**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 3: Comparison of Model Versions

	Class	V4	V5	V6
GT **Test** **Train**	ablation	151 151+109 364	151 151+109 364	151 151+109 364
	bottle	63 63+20 209	126 126+20 1000	126 126+20 400
	breakdown	310 310+109 364	310 310+109 364	310 310+109 364
	cable	92 92+58 224	184 184+58 1000	184 184+58 400
	capsule	109 109+23 219	218 218+23 1000	218 218+23 400
	carpet	89 89+28 280	178 178+28 1000	178 178+28 400
	fracture	370 370+109 364	370 370+109 364	370 370+109 364
	grid	57 57+21 264	114 114+21 1000	114 114+21 400
	groove	241 241+109 364	241 241+109 364	241 241+109 364

	Class	V4	V5	V6
GT **Test** **Train**	hazelnut	70 70+40 391	140 140+40 1000	140 140+40 400
	leather	92 92+32 245	184 184+32 1000	184 184+32 400
	metal_nut	93 93+22 220	186 186+22 1000	186 186+22 400
	pill	141 141+26 267	282 282+26 1000	282 282+26 400
	screw	119 119+41 320	238 238+41 1000	238 238+41 400
	tile	84 84+33 230	168 168+33 1000	168 168+33 400
	toothbrush	30 30+12 60	60 60+12 1000	60 60+12 400
	transistor	40 40+60 213	80 80+60 1000	80 80+60 400
	wood	60 60+19 247	120 120+19 1000	120 120+19 400
	zipper	119 119+32 240	238 238+32 1000	238 238+32 400

Table 4: Data Augmentation: Model Comparison V4, V5 & V6

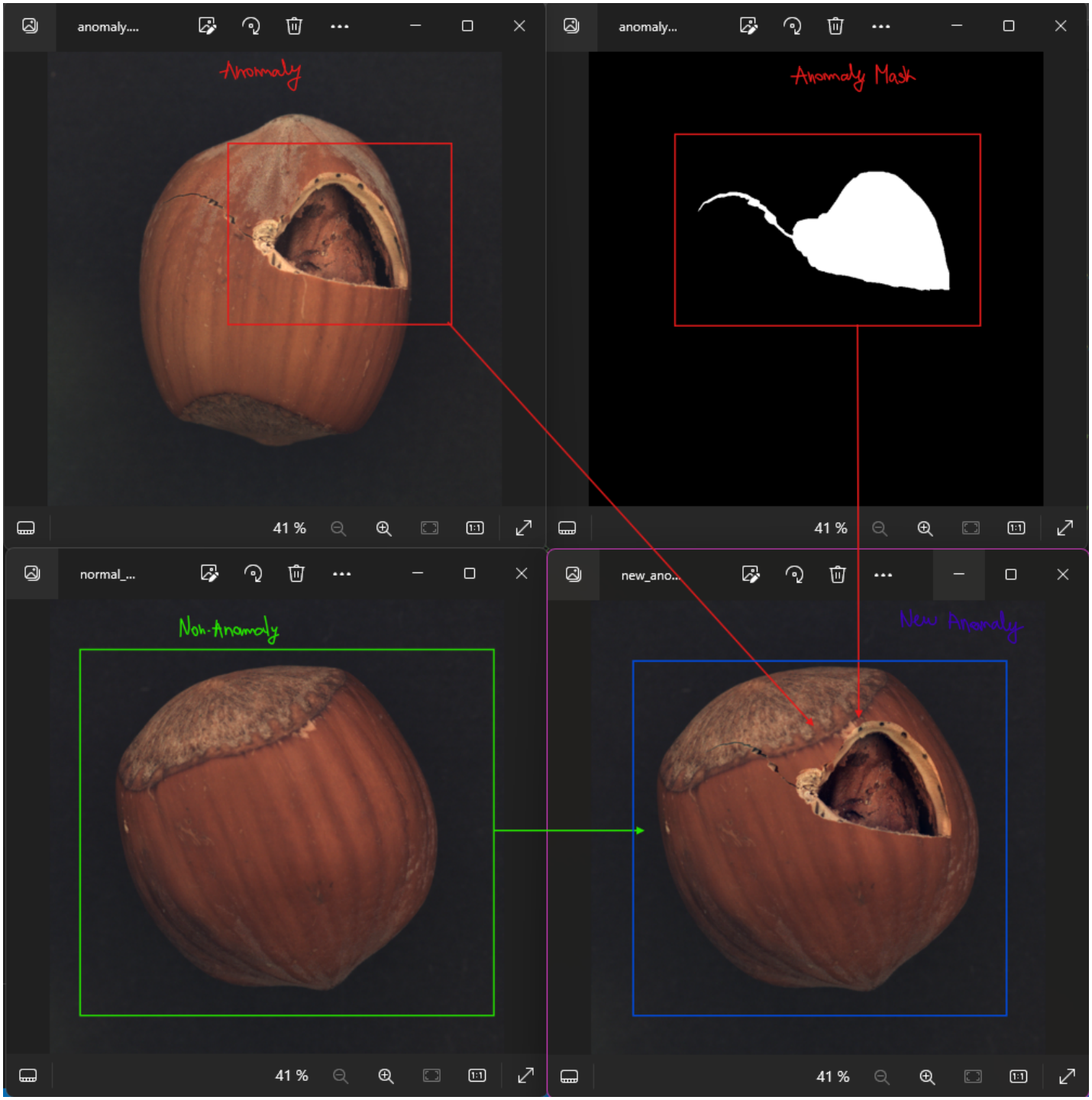


Figure 7: Poisson Editing Technique: Hazelnut

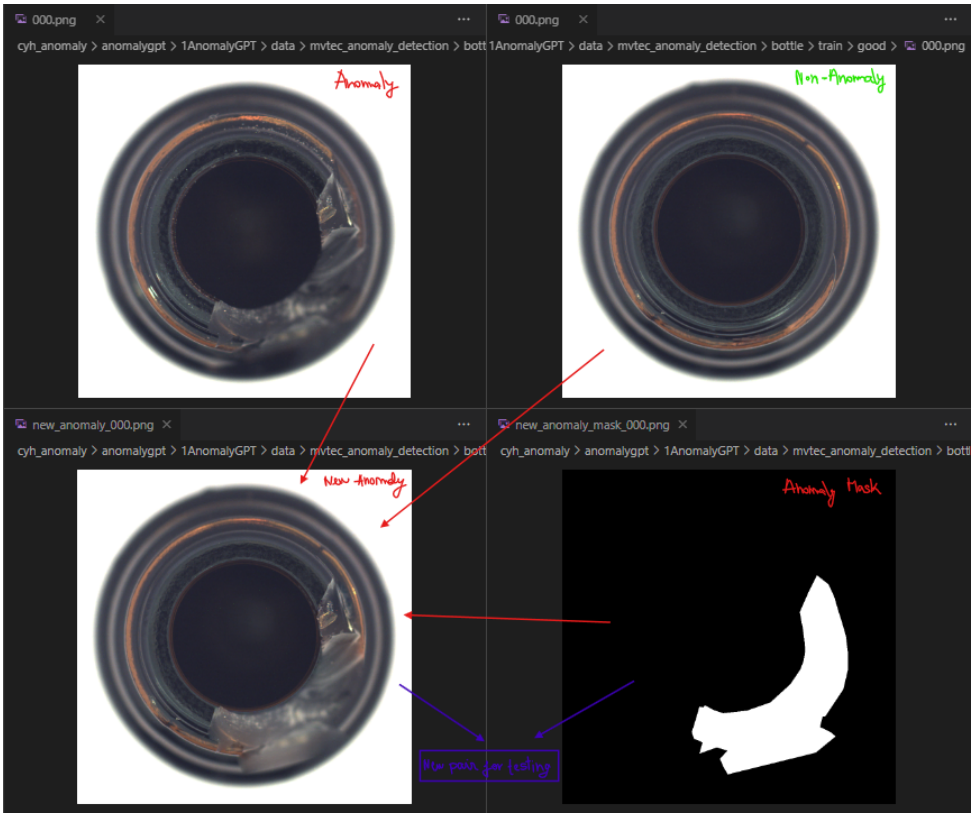


Figure 8: Poisson Editing Technique: Bottle

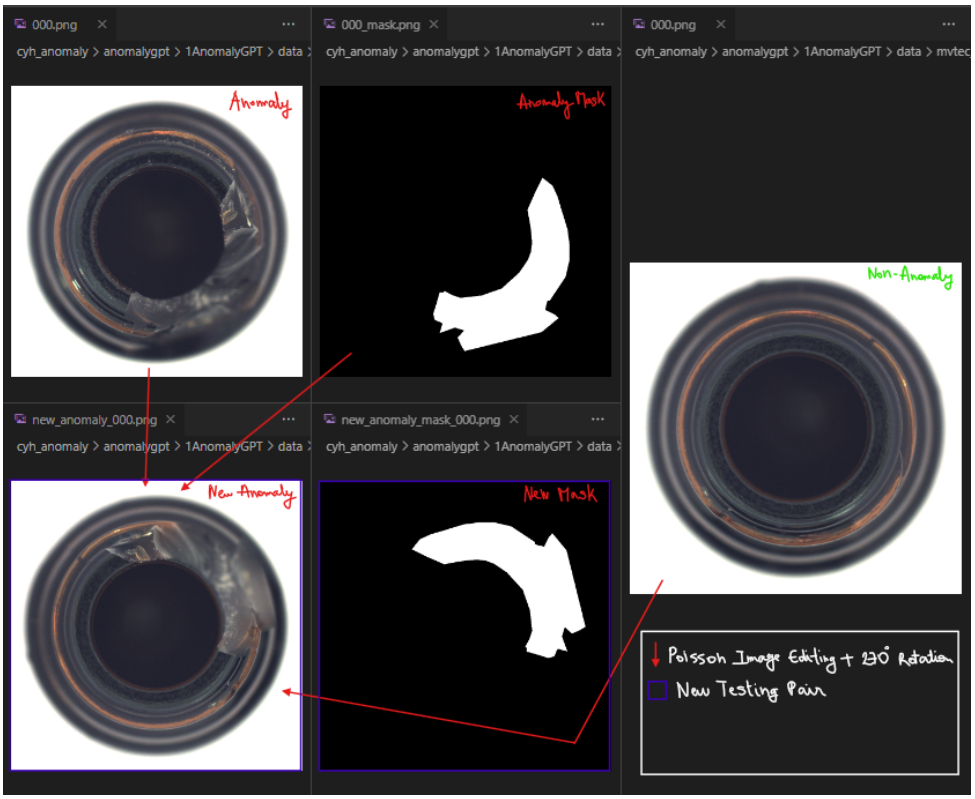


Figure 9: Improved Poisson Editing Example: Bottle