Universiteit Leiden

Efficient Anomaly Detection for Industrial Purposes: A Comparative Study of Transformers and CNNs

Enrico Luchini (s3622436)

Supervisors:
Prof. Dr. M.S.K. Lew & Dr. E.M. Bakker

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

July 30, 2026

Abstract

Industrial visual anomaly detection is a growing field aiming at replacing human manpower with deep neural network models to increase scalability and productivity. To perform such a task, the model needs not only to be accurate, but also to comply with the requirements of manufacturers. This can be a memory limitation or a time limitation. Models therefore need to be accurate, efficient, and fast. While individual models, or even group of models, have been benchmarked, modern literature lacks a unified comparison of CNN, transformer, and hybrid architecture on anomaly detection tasks measuring both accuracy and efficiency under identical conditions. We benchmarked nine models (four CNNs, two transformers, and three lightweight hybrids) on anomaly image datasets (MVTec AD and VisA), to find if lighter models can perform more efficiently than heavier models, and if combining CNNs and transformer strengths can help bridge the gap between efficiency and accuracy. We introduced a new model setup using the PatchCore framework with the FastViT hybrid model as the backbone network, and we added a parameter-free Cross-Scale Attention fusion mechanism to increase accuracy. We measured accuracy metrics such as Image-AUROC (Area Under the Receiver Operating Characteristic curve, a threshold-free measure of classification accuracy where 1.0 is perfect predictor and 0.5 is random) and Pixel-AUPRO (Area Under the Per-Region-Overlap curve, a pixel-level anomaly detection metric that applies a weighted score to avoid large anomalies dominating the score) and efficiency metrics such as the number of the model's parameters, FLOPs (FLOating Point operations per second, a measure of computational performance), model's inference latency, and peak GPU memory usage. From the results we found no efficiency increase in lightweight hybrid models over CNN-based methods nor transformer-based methods. MobileViT-PC, the smallest model at 2.5M parameters, showed ˜18% higher latency than PatchCore-R (75.3ms vs. 88.8ms on MVTec AD) despite being 10× lighter, confirming that parameter count is a poor measure of model's efficiency. Our CSA fusion mechanism allowed the FastViT-PC-CSA to achieve competitive results, reaching 95% pooled Image-AUROC, but multiplied GPU memory usage sevenfold. The transformer-based model Dinomaly achieved the most consistent performance across all metrics, with the highest accuracy and second-lowest latency in the benchmark, but it also has different preprocessing pipeline cropping images to 392px compared to every other model's 224px, which influences the outcome.

This thesis was carried out at LIACS, Leiden University, under supervision of Prof. Dr. M.S.K. Lew.

Contents

1 Introduction

1.1 Motivation

Production lines in recent times have focused on scaling up and increasing throughput, achieved by scaling and optimizing each stage of the line. One of these stages is quality control. Quality control has commonly been done by humans, who manually check each item going through the production line, but such labor is not easily scalable. To solve this problem, we turn to computers and automation.

Computers excel at repetitive tasks, such as quality control, and with the introduction of computer vision models able to process images, researchers have started developing anomaly detection models. Anomaly detection is a subfield of computer vision and consists of detecting if an item has a defect or is defect-free. The damage done by a defective item passing through the production line can vary, such as product recalls, material waste, financial loss, brand damage, etc. Therefore these models need to be accurate. But the speed of modern production lines is often counted in the hundreds or thousands of items per minute. With a production rate of 1000 items per minute, the inspection window for each item is 60ms. This means that the computer vision model has to perform inference and decide whether an item is defect-free or not in a millisecond-level time window. It is also the case that industries do not have the same hardware that researchers use to perform their experiments, which limits how heavy a model can be. A state-of-the-art model that is very accurate but slow, or too heavy, becomes useless in an industrial setting. Therefore, to replace the human force in quality control, these models not only need to be accurate, but also fast and efficient.

1.2 Problem Statement

There are two main architectures used for computer vision tasks: CNNs and vision transformers. CNN models are more efficient and are better able to extract local features, such as textures and patterns, and are the standard anomaly detection models. Meanwhile, vision transformers make use of attention heads and are able to capture global features, but due to their architecture, they are inherently heavy, with $O(n^2)$ complexity to process images, which can lead to slowdowns and heavy memory costs. To solve the transformer weakness, researchers have developed lightweight transformers [1, 2], which promise to close the gap between accuracy and efficiency.

All these models have been benchmarked and compared to similar ones, but there is no unified systematic comparison of CNNs and (lightweight-) transformers measuring both accuracy and efficiency under identical conditions. Such a comparison is an important step for an eventual manufacturer who wants to make an informed decision on which model to adopt for their own edge case.

1.3 Research Question

The goal of this bachelor thesis is to answer the following question:
Can lightweight transformer architectures outperform CNN-based anomaly detection methods in terms of accuracy, parameter efficiency, and inference speed on industrial image anomaly detection benchmarks?

To answer the above question, we will first answer these subquestions:

1. Which transformer achieves the best trade-off between accuracy and computational cost when used as the backbone of existing anomaly detection frameworks?

2. How do these transformers compare to specialized CNN-based anomaly detectors in per-category performance on datasets containing both structural and textural anomalies?

These will be answered by running a unified benchmark over different models of both families of architecture, on two anomaly detection datasets.

We expect lightweight models to trade some accuracy for a large decrease in latency time, since smaller models tend to drop in accuracy, but given the small number of parameters and FLOPs, it should result in lower latency and memory usage.
We also expect this latency reduction not to follow a linear pattern with the parameter reduction, as hardware-level factors such as memory bandwidth can dominate inference time independently of parameter count.

1.4 Thesis overview

This bachelor thesis was carried out at the Leiden Institute of Advanced Computer Science (LIACS), Leiden University, under the supervision of Prof. Dr. M.S.K. Lew. This section was an introduction to the topic of computer vision and anomaly detection. In Section 2 we will discuss related works and how different architectures work. In Section 3 we will explain the experimental setup for the comparison, and show the output of the simulations in Section 4. The final section, Section 5, will discuss the results and possible further research.

2 Background and Related Work

2.1 Anomaly Detection

Quality Control is a sector every industry needs, but it is slow, tedious, and repetitive, an optimal problem to solve using Computer Vision. Such a computer vision task is called Anomaly Detection, which consists of a machine learning model detecting when an item or product has a defect or not. Defects can range across a broad spectrum (scratch, bent, twist, missing piece, etc.), and as such, it would be impossible to show the models every possible one. However, classification tasks need examples for every class; that would mean having a dataset including all possible defects, which is

infeasible.

Computer vision models, therefore, need to rely on an unsupervised training framework: training only with defect-free samples. This flips the idea of showing models every class (and therefore defect) and instead makes the model understand what a "normal" item is and looks like, and then any deviation from normality is an anomaly. This approach gives two types of output granularity: image-level and pixel-level. Image-level output is the final score, whether an image has an anomaly or is defect-free, while pixel-level scores per-pixel anomalies, or how each pixel deviates from "normal", which generates a heatmap of the anomaly location. To evaluate pixel-level, a ground-truth mask is needed. We called the training of anomaly detection models unsupervised, but it could be called semi-supervised or one-class, since we know all training images are defect-free.

2.1.1 Anomaly detection paradigms

Anomaly detection paradigms can utilize different taxonomies to complete the desired task. The main ones are the memory bank mechanism, the reconstruction-based, the student-teacher approach, the hypersphere-based approach, and the normalizing flow.

Memory bank, or embedding-based, extracts features by sampling images from a dataset and stores them in a bank. The sampling is not done randomly but through a mathematical algorithm that selects subsets of the dataset in a way that preserves coverage. It starts with an empty set and iteratively adds the farthest point from the current set. This guarantees feature space diversity. It is a gradient-free approach, requiring no training, but it runs one initial epoch to populate the memory banks. To perform anomaly detection, it compares test features to stored ones using a nearest-neighbour approach, and if the distance is above a threshold, it will flag the image as an anomaly. The main drawback of memory-bank methods is that inference requires a nearest-neighbor search over the entire bank, which has a cost of $O(|M| \cdot d)$ per query, where $|M|$ is the coreset size and d the feature dimension.

Reconstruction-based is a paradigm that works using an encoder-decoder pair. An encoder, usually with frozen weights, generates a feature map of normal images. The decoder is then tasked with reconstructing the image from the feature map. The decoder is trained to be able to reconstruct normal image's feature maps, which forces the decoder to learn semantic relationships between features rather than pixel pattern. When encountering a defect image, the anomalies will disrupt the relationship learned by the decoder, producing a high reconstruction error and flagging it as anomalous.

The student-teacher paradigm works in a similar way. The teacher is a model with frozen weights, which produce feature pyramids using normal images. The student model is randomly initialized with the same architecture as the teacher model, and it is trained to match the teacher feature pyramids. During inference, the student is able to match the pyramid of normal images, but when presented with a defect image, it fails to reconstruct the areas where the anomaly is present, thus generating an anomaly heatmap.

The hypersphere-based paradigm makes use of a feature-space to cluster normal images and push away anomalous ones. It works by creating normalized vectors from images and then projecting all the vectors into a hypersphere. It then learns simultaneously a hyperplane to separate the hypersphere's origin from the projected normal features and a hypersphere boundary that encloses the projected normal features very tightly. It is trained to push normal features close together and push anomalous features away from the hypersphere boundary. At inference, a test image's feature is projected onto the hypersphere, and based on whether it falls inside or outside the boundary region, it is flagged as normal or an anomaly, respectively.

The normalizing flow paradigm, which is the only paradigm not benchmarked in this comparison, maps a real-world distribution to a more well-understood distribution, usually Gaussian, using reversible transformations. Like the name suggests, it normalizes the distribution (Gaussian distribution = normal distribution) in a flow, or sequence, of incremental transformations. During training, it extracts features from normal images and then passes them through the flow model. The flow model then learns how to map these features to a Gaussian distribution. At inference time, the test image passes through the same pipeline and is placed onto the Gaussian distribution. A normal image would end up near the middle of the Gaussian, while anomalous images are placed farther away in the tails of the distribution.

2.2 Feature Extraction Backbones

CNNs were popularized by AlexNet [3] as one of the first deep learning models capable of effectively processing images. More recent models perform significantly better than the first instances of CNNs, but the underlying mechanism that makes them work remains unchanged. It works by processing an input image by sliding a small filter (usually 3×3) and computing a weighted sum at each position. This happens multiple times over different filters, with each filter capturing a different pattern (edge, stripe, corner, etc). This process is called convolution (Figure 1). At each iteration, the image decreases in resolution, and the context the model is able to capture increases. This is the fundamental limitation of CNN architecture. It detects very early local patterns and textures, and only later in the feature pyramid it links global features together. And because of this gap, information can get diluted passing through layers.

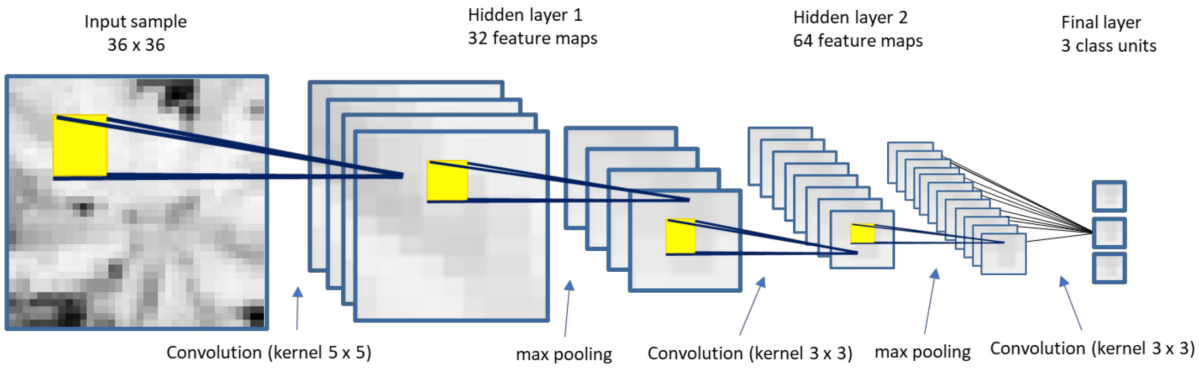


Figure 1: Convolution operation, with the filter (yellow) sliding over the image. Credits: [4]

Then came transformers [5]. They work by splitting the input data into tokens, which are then processed using the self-attention mechanism, the key mechanism introduced by this architecture. Each token computes three vectors: query (Q), key (K), and value (V). Then every token's Q is compared to every other token's K, which tells the transformer how relevant one token is compared to the other.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

This means that every token can interact with every other token independently from where they are located in the data, hence the strength of transformers at detecting global features. There is also a multiheaded mechanism, where h number of heads runs in parallel, and then their output is concatenated and projected. Unfortunately, comparing every token with each other produces a complexity $O(n^2)$, which means the bigger the data, the longer it takes.

At first, transformers were only used for natural language processing tasks, and the limitations given by the degree of complexity were negligible, but when increasing dimensionality and going from one-dimension to two-dimensions, a big computational problem arose. To process an image, a computer vision model has to translate each pixel value into numbers to then compute operations on those numbers, but an image pixel number increases quadratically with its resolution, going from $O(w*h)$ pixels to $O(w^2*h^2)$ naive pixel-level self-attention (given the transformers complexity).

Vision transformers [6] were introduced to tackle the problem. The core feature of these transformers is splitting the image into smaller patches (usually 14×14 or 16×16), flattening them, projecting them into a vector, and adding positional embedding so the transformer knows where each patch is located. This "transforms" an image into a sequence of tokens, like a natural language sentence, which makes the computational cost decrease. Unfortunately, patchification does not fix the high parameter count and heavy memory usage of transformers, and to try and solve it, lightweight transformers were introduced, such as MobileViT [1], EfficientFormer [2], and FastViT [7]. Lightweight transformers were developed to try and bridge the gap between accuracy and efficiency.

Therefore, CNN architecture has an inductive bias: it uses 3×3 convolution filters that only allow it to see the nine neighbouring pixels and it slides through the whole image, detecting the same pattern regardless of its location. A CNN model follows a hierarchical feature pyramid, where first it works on edges, followed by texture, parts, and then objects. It is also data-efficient; it doesn't need large amounts of training data to achieve good performance. On the other hand, the vision transformer's architecture has a global receptive field from the early layers, since every patch interacts with every other patch in all layers, but this leads to very high computational costs. They also require massive pretraining to learn what CNNs manage to learn thanks to their architecture design. And finally, hybrid architecture uses early convolution layers to easily process local features when the resolution is still high and transitions to attention blocks in deeper layers when resolution decreases. In summary, CNNs trade global context for data efficiency, transformers trade computational cost for global receptive fields, and hybrids attempt to combine the strengths of both: early convolutions for cheap local processing, late attention for global context.
Lightweight transformers make use of some clever strategies to reduce the computational cost needed to run them. There are three main approaches: convolutional token mixing, structural reparameterization, and pooling-based token mixing. Convolutional token mixing changes some

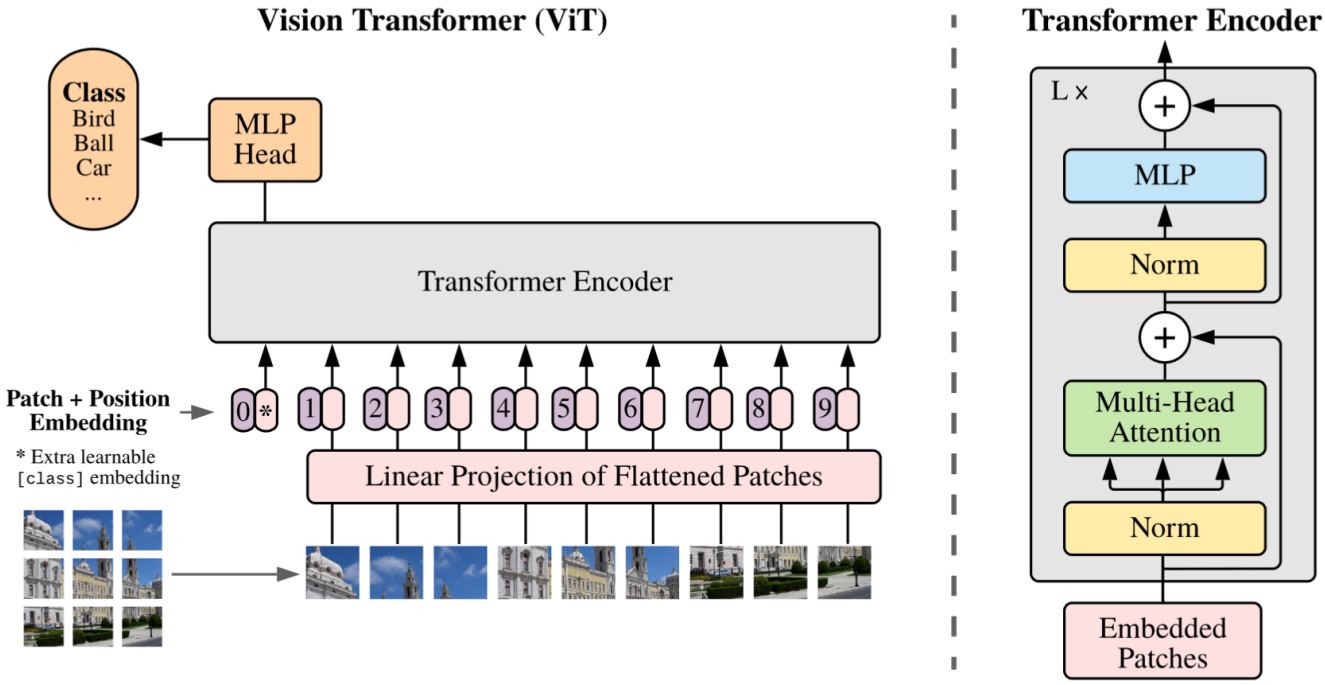


Figure 2: Patchification process, where an image is split into smaller non-overlapping patches and flattened into tokens. Credit: [6]

early attention blocks, which are then replaced with convolution layers, since global attention is not needed in early high-resolution stages but is essential in downsampled spatial grids. The second approach, structural reparameterization, splits blocks into multi-branch blocks that perform parallel convolutions during training, which are then algebraically fused together during inference. And lastly, pooling-based token mixing removes attention blocks in the early layers and instead makes use of a simple average pooling to mix spatial information in early high-resolution stages, while attention is only used for low-level resolution stages, where the token count is smaller.

The models used in this comparison, which will be detailed in the following two sections, are PatchCore, CFA, STFPM, Dinomaly, AnomalyDINO, and MobileViT. They all have a frozen backbone. The backbone is the pretrained feature extractor network, which takes the input images and processes them into feature maps, the numeric representation of the images. It is pretrained with a large dataset that contains various pictures with different subjects. Common networks are ResNet [8] and WideResNet [9].

The backbone is the structural core of the models; it is what makes them "see" the images, and on top of that we add an anomaly detection mechanism that is able to decide whether an image contains an anomaly or not. A frozen backbone means that the pretrained weights of the network are locked and cannot be updated. What changes is the anomaly detection mechanism, which varies between different models. Not all backbones are pretrained in the same way. The transformer-based models in this comparison, Dinomaly and AnomalyDINO, use DINOv2 [10] as their backbone, which is trained in a self-supervised manner. Unlike ImageNet pretraining, which relies on human-provided labels, self-supervised learning builds its own training signal directly from the images,

for example, by teaching the network to produce consistent features for two different crops of the same image. This removes the need for labeled data and, in the case of DINOv2, yields very strong general-purpose features that transfer well to dense tasks such as anomaly localization.

2.3 CNN-based Anomaly Detection

The CNN models that we compare in this thesis are PatchCore, CFA, and STFPM.

Patchcore [11] uses a frozen backbone and a memory bank mechanism to store patch features from the frozen backbone. It works by concatenating layer2 and layer3 of the frozen backbone, which helps have a multi-scale representation. Layer2 better captures fine textures and patterns (early stages), while layer3 better captures semantic structures (middle stages). This concatenation allows the model to have a multi-scale representation and be able to detect both small defects and large structural anomalies. PatchCore also concatenates features from neighbouring positions via average pooling to prevent errors in edge cases where the defect might be between the feature grid points. It is gradient-free, meaning it does not need a training phase. It starts with an empty set and runs one epoch to update the memory banks using coreset-subsampling (usually 1%, 10%, or 25%). Coreset-subsampling consists of adding the furthest training feature from the current set. This guarantees feature space's diversity and does not skew towards more dense features. It then scores test patches by nearest-neighbour distance using a weighted sum, to reduce the influence of possible outlier neighbours.

CFA [12] uses a coupled-hypersphere feature adaptation mechanism, which represents images in a space where similar images cluster together, marking everything that lands outside as an anomaly. The coupled-hypersphere mechanism works by learning simultaneously (coupling) a hyperplane that separates normal features from the origin and a hypersphere that sets the boundary of normal features. CFA uses a frozen backbone but needs to train its adaptation mechanism by pushing normal features closer to their assigned prototype and pushing away anomalous features.

The STFPM [13] model uses a student-teacher approach. The student network "learns" from the teacher network (the frozen backbone) by training to match the teacher feature pyramid on normal images. Both teacher and student have a three-layer backbone, each handling feature maps at different resolutions, and the student matching happens in all three layers simultaneously. As mentioned with PatchCore, having three layers allows to catch small texture defects in the early stages, as well as bigger structural anomalies in the deeper stages. The loss is calculated as $||feature\_teacher - feature\_student||^2$ summed over all pyramid levels, which during inference allows to produce pixel-level resolution heatmaps of eventual anomalies.

2.4 Transformer-based Anomaly Detection

On the transformer side, we will benchmark Dinomaly, AnomalyDINO, and MobileViT.

Dinomaly [14] uses an encoder-decoder combination to perform anomaly detection. The encoder it uses is DINOv2, which is the frozen backbone, and it trains the decoder using the normal images. The decoder, which usually reconstructs raw pixel images, has to reconstruct intermediate feature maps. This forces the decoder to understand and learn feature relationships that can be disrupted if an anomaly is present. During inference, when facing an anomaly, it is not able to reconstruct normal features, resulting in a high reconstruction error. Dinomaly only has a transformer encoder, the DINOv2 backbone, and uses a simple lightweight CNN to work as a decoder. The researchers who developed the model concluded that the frozen encoder is able to capture enough features, and another visual transformer would only increase the size and, potentially, the computational cost of the model. Furthermore, DINOv2 has a variant called DINOv2-Reg, with register tokens. Register tokens are extra learnable tokens that are able to reduce artifacts in feature maps, and without the help of these tokens, DINOv2 can end up producing artifacts especially in dense predictions.

AnomalyDINO [15] uses the same frozen backbone as Dinomaly, DINOv2, and uses memory banks with few-shot sampling, where a very limited amount of normal images is selected, and, like PatchCore, it scores the test patches by nearest-neighbour. The few-shot sampling only selects 8 normal images, which are not randomly selected but are specifically picked. These images are chosen to maximize feature diversity coverage. AnomalyDINO also runs PCA on all patch features to detect and remove the background (Figure 3). It works by computing the variance of a patch, and if it is below a certain threshold, which means the pixels in that patch are all similar to each other, it flags the patch as background. This mechanism prevents the background from influencing the anomaly score.

Figure 3: PCA-based background removal, where low-variance patches are removed before anomaly scoring. Credit: [10]

MobileViT-PC is a re-adaptation of PatchCore, using a transformer backbone (MobileViT [1]) instead of a CNN one. The core mechanisms remain unchanged, using coreset memory banks and nearest-neighbour scoring, but the backbone is now a lightweight transformer: MobileViT-S, the smaller variant of the MobileViT transformer. This setup also allows to isolate the effect of the

backbone on the final results, showcasing the difference between a CNN-based PatchCore and a transformer-based PatchCore.

2.5 Related Comparisons and Gap

Other studies have done comparative benchmarks of CNNs against vision transformers [16, 17, 18] or benchmarks of different models for anomaly detection tasks [19], but they rarely measure efficiency alongside accuracy. This study's aim is to close the gap and benchmark the accuracy and efficiency of CNNs against lightweight vision transformers on anomaly detection.

3 Methodology

3.1 Experimental Setup

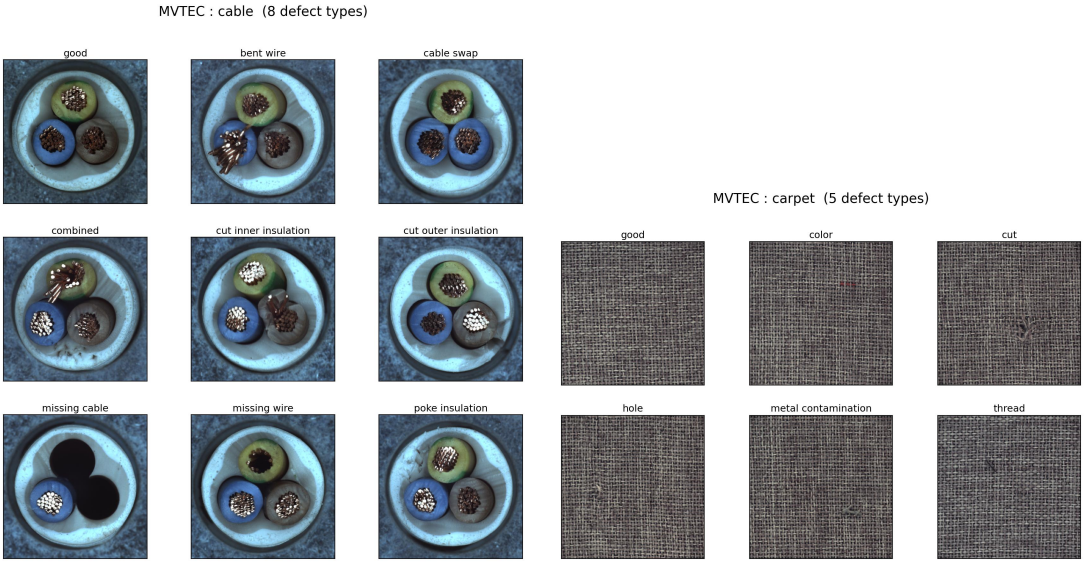
All experiments were run with the same datasets and hardware, and the same metrics were recorded after each run. Only preprocessing changed depending on how each model processed input images. The experiment was run using Python 3.12, the Anomalib unified framework [20], version 2.3.2, on a LIACS machine, hosting NVIDIA RTX 3090s, with 24GB of VRAM. To compute operations, we set `torch.set_float32_matmul_precision="high"`. The simulations were run on a single GPU for every model, and to avoid OutOfMemory errors and to keep memory measurements precise, at the end of every run we executed a memory cleanup. For each dataset's category, we ran a train phase, if required by the model, and test phase, and recorded the metrics that are explained in the evaluation metrics subsection below and then aggregated all category metrics into one mean value per model. Each model was run 20 times per category to ensure reliable mean and variance estimates.

3.2 Datasets

The datasets used were the MVTecAD [21] and VisA [22], commonly used for anomaly detection benchmarks.

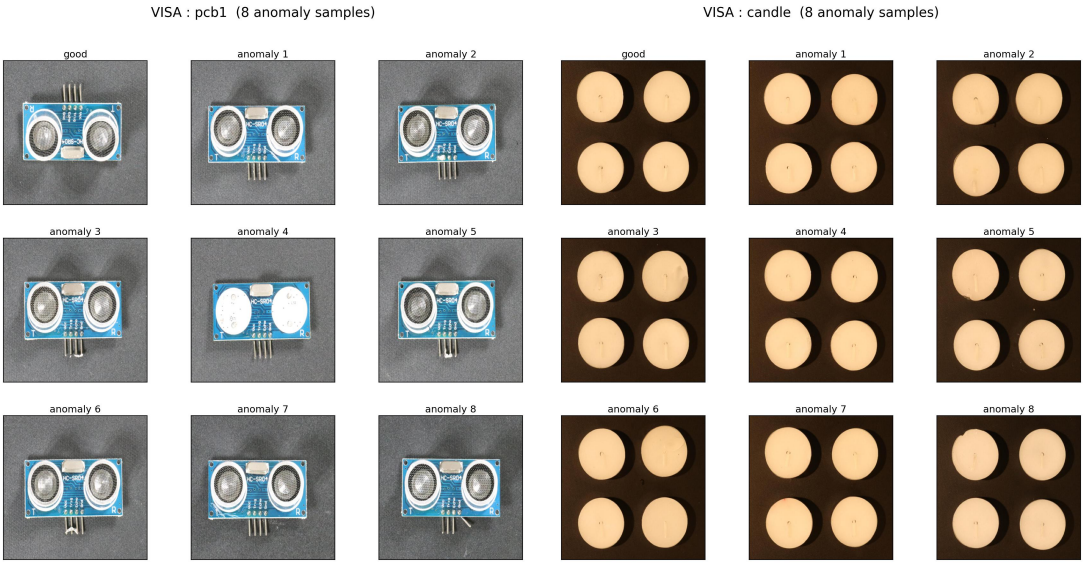
MVTecAD contains ≈5354 images spanning 15 categories. Each category is a class, and they could be split into textural classes (carpet, grid, leather, tile, toothbrush, wood) and object/structural classes (bottle, cable, capsule, hazelnut, metal nut, pill, screw, transistor, zipper). It has ≈3600 normal images and ≈1700 test images that include both normal and defect images. It has 73 different defect types, with pixel-precise masks.

VisA contains ≈10821 images, spanning 12 categories. It only has object classes, no textural ones, and the complexity of the object increases compared to the MVTecAD dataset. VisA has classes for PCBs, capsules, candle, macaroni, cashew, chewinggum, and fryum. Some classes have multiple instances, like PCB (pcb1, pcb2, pcb3) and macaroni (macaroni1, macaroni2). The dataset contains ≈9600 normal images and ≈1200 anomaly images. Beyond increased object complexity, the VisA dataset also has images with multiple instances of defects in one object that make the anomaly



(a) Cable (structural) (b) Carpet (textural)

Figure 4: Sample images of items in the MVTecAD dataset with their corresponding defects.



(a) PCB (b) Candle

Figure 5: Sample images of items in the VisA dataset with their corresponding defects.

detection task even harder.

MVTecAD was selected for being the standard dataset for anomaly detection benchmarks, and VisA was chosen for its increased complexity leading to better generalization. Both datasets have premade train and test splits, with the train set only containing defect-free images, while the test split has both defect-free and anomaly images.

3.3 Preprocessing

All models follow the same preprocessing pipeline, with some variation in resolution for the Dinomaly model. The raw image input is resized to 256×256 pixels and then center-cropped to 224×224 pixels. These settings are the original preprocessing steps from each model's paper.

The only difference from this general setup is Dinomaly, which resizes at 448×448 pixels and then center-crops at 392×392. We opted for keeping the preprocessing steps faithful to the original papers instead of running a unified preprocessing pipeline for all the models. To note that this gives an edge on Dinomaly, since it is processing higher resolution images. This means that it will be more accurate, but it also means it will use more GPU memory to process the bigger images compared to the rest of the models.

3.4 Models and Configuration

Each model's hyperparameters follow the model's original paper indications, and where missing, we would fall back to Anomalib defaults.

As mentioned in the Background section (Section 2), we selected models from existing literature. We then introduce our own variation of the PatchCore-FastViT hybrid model to try and bridge the gap between accuracy and efficiency. It uses the PatchCore framework with a CNN-transformer hybrid backbone: FastViT [7].

FastViT is a CNN-transformer hybrid that claims to have reached a state-of-the-art efficiency-accuracy tradeoff. It introduced a token mixer operator, RepMixer, that uses structural reparameterization, mixing two of the possible approaches at lightweight transformers. RepMixer uses three parallel branches, one 3×3 depthwise convolution, a 1×1 pointwise convolution, and an identity convolution during training, and at inference time it algebraically folds all branches into a single 3×3 convolution. Since convolutions are linear operations, the sum of all training branches can be transformed into one convolution with weights. This means that during training time, it is able to capture more expressive features using multiple branches but is then deployed with single-branch efficiency and with no accuracy loss.

FastViT makes use of RepMixer blocks in early stages, with structural reparameterization, for local processing, and sparse self-attention blocks only in the final stages where the spatial grid is smaller. A RepMixer block has $O(n)$ complexity (depthwise convolution), which greatly reduces computational cost compared to the self-attention block's complexity $O(n^2)$. Because it uses the same memory bank, coreset sampling, and nearest-neighbour mechanisms as PatchCore, this isolates the effect of the backbones, allowing us to easily compare a CNN backbone with a hybrid one.

We then tweaked the attention mechanism and added a cross-scale attention fusion (CSA). It replaces PatchCore's standard channel-wise concatenation of backbone features. The standard PatchCore pipeline concatenates layer2 and layer3 features channel-wise, which does capture multi-scale information but treats each scale independently. CSA fusion instead takes a coarse feature

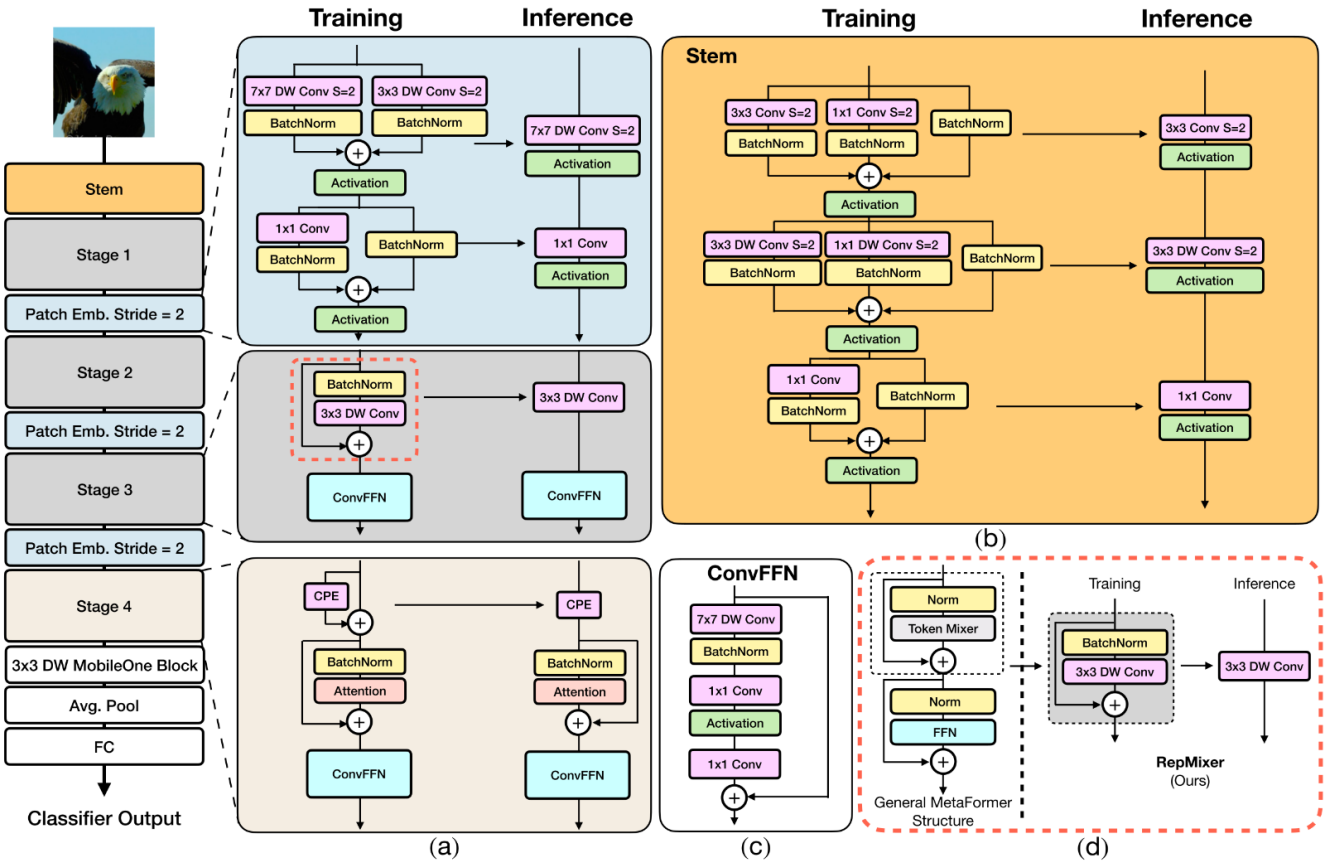


Figure 6: FastViT architecture. Credit: [7]

map from the deep layers, where resolution is low, and upsamples it to match the earlier layer's fine grid, where resolution is high. The coarse map is then L2-normalized per spatial position of the fine map and then re-aggregated its own features, using a softmax cosine-affinity matrix via non-local self-attention [23]. This results in a context-enriched coarse feature map, which is then concatenated with the fine features. Possible intermediate layers are plain-concatenated. This injects global semantic context from the deepest low-res layer into the finer layer's features. During this process the weights of the model remain unchanged.

CSA is a parameter-free method, only using L2-normalization and matrix multiplication, preserving the gradient-free properties of PatchCore. We then performed a sweep of different layer configurations to identify the optimal one. The configurations evaluated were: s0-s3, s1-s2, s1-s3, s2-s3, s1-s2-s3. Layer and stage can be used interchangeably (s0-s3 is the same as layer0-layer3).

Model		**Backbone**	**Method**	**Params**	**GFLOPs**
CNN baselines					
PatchCore-R [11]	2022	WideResNet-50-2 [9]	Memory bank (10% coreset)	24.9M	18.4
PatchCore-C	2022	ConvNeXt-v2-nano [24]	Memory bank (25% coreset)	7.5M	4.2
STFPM [13]	2021	ResNet-18 [8]	Student–teacher	5.6M	3.7
CFA [12]	2022	WideResNet-50-2	Hypersphere adaptation	31.3M	18.4
Transformer baselines					
Dinomaly [14]	2024	DINOv2-Reg ViT-S/14 [10]	Feature reconstruction	37.4M	33.9
AnomalyDINO [15]	2024	DINOv2 ViT-S/14	Memory bank (8-shot)	22.1M	N/A
Hybrid backbone					
MobileViT-PC	2022	MobileViT-S [1]	Memory bank (25% coreset)	2.5M	2.7
FastViT-PC	2023	FastViT-SA12 [7]	Memory bank (25% coreset)	10.4M	3.0
FastViT-PC-CSA	2026	FastViT-SA12	Memory bank (25%) + CSA fusion	10.4M	3.0

Table 1: The nine anomaly detection models evaluated in this thesis, grouped by family. FastViT-PC-CSA (in bold) is our proposed model. N/A: FlopCounterMode incompatible with AnomalyDINO.

To note is how in Dinomaly's paper the researchers trained the model using all categories together, while we trained each category standalone for consistency and to reflect real-world scenarios where a model might only be used for a single category of products.

3.5 Evaluation metrics

Two types of metrics were recorded for each model on each run: accuracy and efficiency metrics.

3.5.1 Accuracy metrics

For accuracy, we recorded Image AUROC and Pixel AUPRO.

Image AUROC is the area under the ROC curve and is what detects an anomaly. It plots the true positive rate against the false positive rate across all thresholds. It is threshold-free and showcases

whether the model is able to correctly identify both the normal items and the anomalies, where 0.5 is random guessing and 1.0 is a perfect classifier.

Pixel AUPRO is the area under the per-region-overlap curve, and is what localizes the anomaly. It localizes where the anomaly is by weighting each defect region so that bigger defects do not drown out smaller defects. Ranges from 0 to 1, where a higher value indicate a more accurate anomaly localization.

3.5.2 Efficiency metrics

Meanwhile, for efficiency metrics, we recorded the number of parameters, FLOPs, latency from raw image to output, and peak GPU memory usage.

The number of parameters is a broad way to measure how heavy a model is. The more parameters it has, the heavier it becomes. Lower parameter counts are preferable on resource-constrained devices.

FLOPs are the floating-point operations performed during the forward pass on the backbone feature extractor. It is a device-agnostic way to measure computation efficiency, where lower FLOPs indicate less computational work per inference, and was recorded using PyTorch FlopCounterMode. AnomalyDINO uses a kind of operation incompatible with FlopCounterMode, and for this reason, this model does not have a FLOP metric.

Peak GPU memory usage is recorded to keep track of the memory expenditure of the model during inference. Before each run we reset the PyTorch *torch.cuda.peak_memory_stats*, and after inference we called *torch.cuda.max_memory_allocated*() to record the memory usage, which includes model weights, activation, and memory bank.

The latency times how long it takes for a model to take the raw image as input, preprocess it, perform inference, and return the output. Lower latency mean faster inference. We record the latency for every step, which helps reveal bottlenecks, and compute the sum of all latencies to find the total timing of each model.

3.5.3 Warm-up protocol

To properly measure latency for real-world scenarios, we run a warm-up protocol, where we run 20 simulations that are then discarded. This is helpful for multiple reasons: it gives time for the CUDA kernel to compile, avoiding a cold start; it allows the GPU memory allocator to stabilize its memory caching; and it boosts the GPU clock speed from an idle power-saving state to a working state. Without such a protocol, the early measurements could end up more than ten times slower than what a warmed-up machine can achieve, therefore skewing the results.

After the warm-up protocol we run 200 iterations and record the aforementioned metrics. For these runs, we use a batch size of one to simulate real production lines where there is one product at a time.

Another key function that was set before the test run was *torch.cuda.synchronize*(). GPU kernels run asynchronously, which means that depending on the operations they complete, they queue the result to the CPU runtime, which then proceeds with its computation. Without synchronization, the CPU will not wait for the GPU to complete its workload and would time the GPU performance without waiting for the GPU to finish its job. We also put the synchronize function before starting the timer to complete any pending GPU job from prior operations. After every run, a cleanup was executed to release GPU memory. This was to prevent memory accumulation over multiple runs, which would lead to possible incorrect measurements or cross-contamination between different category runs.

Of the metrics recorded, we report the mean latency and the 99th percentile latency. Mean latency gives an idea of how each model performs on average, but in production lines, like in computational complexity, it is often the worst-case scenario that has the most impact and is taken as a baseline.

4 Results and Discussion

First, we check which layer configuration for FastViT-PC-CSA was the most performant one, and we can see that the s0-s3 manages to achieve the best accuracy with a mean of 95% Img-AUROC over both datasets (Table 2). It is also true that the same configuration uses a mean of 3986MB peak memory usage, 4.6× more than the amount of memory used by second most resource-intensive configuration, s1-s2-s3 with a peak of 875MB. Configuration s1-s2 performs slightly worse than s0-s3, 89.9% against 95.4% on the MVTec AD dataset, but has much shallower GPU memory peaks, with a 6.4× smaller peak usage. It is a good contender for the FastViT configuration, but we opted to use s0-s3 given higher accuracy and assuming most modern GPUs are able to handle the resource usage. If manufacturers cannot afford such resource expenditures, then s1-s3 is the second-best alternative. From now on, when mentioning FastViT-PC-CSA, we imply the s0-s3 layer configuration.

			MVTec AD		**VisA**		
Config	**Stages**	**Ch**	**AUROC**	**AUPRO**	**AUROC**	**AUPRO**	**GPU (MB)**
s0-s3	0, 3	576	95.4	80.0	94.6	85.6	3485 / 4487
s1-s2	1, 2	384	89.9	78.2	91.6	74.6	545 / 751
s1-s3	1, 3	640	91.5	51.5	89.4	60.7	620 / 913
s2-s3	2, 3	768	87.7	64.4	84.1	57.1	536 / 597
s1-s2-s3	1, 2, 3	896	90.9	72.3	86.5	65.4	692 / 1058

Table 2: Cross-Scale Attention layer-configuration sweep for FastViT-PC-CSA (image AUROC, pixel AUPRO, %). GPU column reports peak memory usage on MVTec AD/ VisA.

After selecting the layer configuration for FastViT and running the simulations, the results are as follows (Table 3). All metrics are displayed as mean values over all runs, with the associated

standard deviation.

	MVTec AD		VisA	
Model	**Img AUROC**	**Pixel AUPRO**	**Img AUROC**	**Pixel AUPRO**
PatchCore-R	98.8 ± 1.2	90.5 ± 4.4	92.2 ± 8.6	83.6 ± 7.3
PatchCore-C	96.4 ± 3.9	88.5 ± 4.9	86.4 ± 12.2	82.8 ± 11.5
CFA	91.5 ± 12.2	88.3 ± 6.9	82.0 ± 15.0	80.0 ± 11.2
STFPM	86.6 ± 12.7	51.5 ± 21.7	69.5 ± 12.3	27.1 ± 10.2
Dinomaly	**99.6 ± 0.7**	**95.5 ± 3.4**	**97.9 ± 2.7**	**94.5 ± 3.0**
AnomalyDINO	97.9 ± 3.1	90.5 ± 6.2	91.5 ± 7.2	83.1 ± 8.6
MobileViT-PC	81.0 ± 17.1	64.3 ± 13.8	81.6 ± 11.7	45.5 ± 18.4
FastViT-PC	87.8 ± 11.6	68.6 ± 10.4	79.5 ± 14.2	65.9 ± 18.3
FastViT-PC-CSA	95.4 ± 4.5	80.1 ± 12.1	94.6 ± 6.0	85.7 ± 8.8

Table 3: Per-dataset accuracy (mean ± s.d. across categories), % image AUROC and pixel AUPRO.

	MVTec AD			VisA		
Model	**Lat. (ms)**	**p99 (ms)**	**GPU (MB)**	**Lat. (ms)**	**p99 (ms)**	**GPU (MB)**
PatchCore-R	75.3 ± 3.4	186.8 ± 9.6	546 ± 13	76.9 ± 3.4	190.6 ± 11.8	587 ± 19
PatchCore-C	111.5 ± 13.4	169.3 ± 21.7	935 ± 145	198.2 ± 51.2	244.2 ± 13.3	3126 ± 1406
CFA	64.8 ± 3.0	125.7 ± 9.0	3199 ± 10	62.4 ± 3.5	117.5 ± 5.6	3217 ± 7
STFPM	**24.7 ± 0.9**	**64.3 ± 1.6**	434 ± 32	**25.5 ± 0.9**	**64.2 ± 2.0**	**545 ± 69**
Dinomaly	40.6 ± 2.3	82.0 ± 5.3	943 ± 178	42.4 ± 2.9	87.3 ± 6.1	1334 ± 163
AnomalyDINO	92.5 ± 19.8	279.1 ± 128.0	1188 ± 431	127.1 ± 8.6	458.0 ± 37.6	2046 ± 1250
MobileViT-PC	88.8 ± 2.6	172.8 ± 8.6	**351 ± 52**	96.1 ± 5.3	180.9 ± 10.7	680 ± 170
FastViT-PC	106.4 ± 2.9	216.3 ± 7.3	537 ± 19	107.1 ± 4.0	218.6 ± 12.2	629 ± 44
FastViT-PC-CSA	78.7 ± 17.5	85.8 ± 18.6	3278 ± 129	210.3 ± 61.8	224.9 ± 62.1	4460 ± 533

Table 4: Per-dataset efficiency (mean ± s.d. across categories): end-to-end latency, 99th-percentile latency, peak GPU memory. Params and GFLOPs are architecture-fixed (see Table 1).

From Figure 7 we can better visualize the performance of the models for the accuracy-efficiency trade-offs. The ideal model occupies the top-left corner (high accuracy, low latency). The Pareto frontier lies between Dinomaly and STFPM, with the highest pooled Img-AUROC at 98.8% and the lowest latency at mean 25ms respectively, and we can see how the hybrid lightweight models place themselves further to the right of it. Models on the Pareto frontier cannot be improved in one dimension without sacrificing the other.

Between the hybrid models, FastViT-PC-CSA wins in accuracy, with a mean of 95% Img-AUROC, while FastViT-PC and MobileViT-PC scored 83.65% and 81.3% Img-AUROC. MobileViT-PC has the lowest latency, at 92.5ms and is the smallest model of the whole benchmark with only 2.5M parameters, displaying how in the same family of architecture the lightest model is also the fastest.

The CNN model PatchCore-R scored +0.5% Img-AUROC compared to our FastViT-PC-CSA over both datasets and is faster than MobileViT-PC by 16.1ms, making PatchCore-R both more accurate and faster than any lightweight hybrid benchmarked in this comparison, achieving a better accuracy-efficiency trade-off. It is worth noting that although PatchCore-R has a higher mean Img-AUROC score, it actually performs worse than FastViT-PC-CSA on the VisA dataset, scoring -2.4% Img-AUROC.

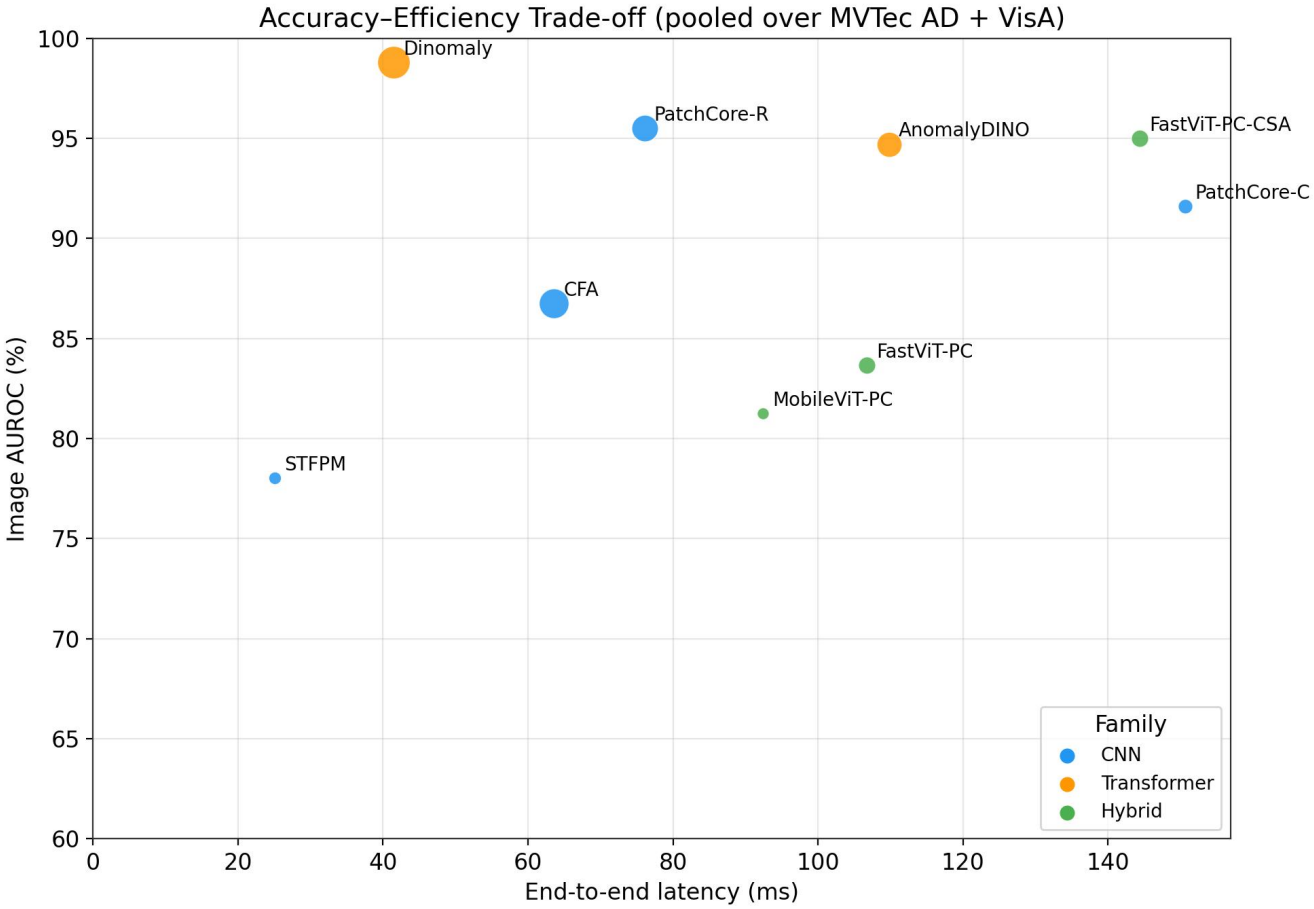


Figure 7: Accuracy-efficiency trade-off. Mean Image AUROC vs mean latency

Efficiency-wise, the most performing model is STFPM, with 25ms latency, since it only has to compute L2 distance between the student and teacher feature difference and uses no memory bank. It is also one of the smallest models in this comparison, with only 5.6M parameters. The second fastest model is Dinomaly, with a mean of 40ms, 64.4% slower than STFPM, despite being the largest model in the comparison with 37.4M parameters. Such speed might be possible due to the architecture and methods of the model. The reconstruction-based anomaly scoring is inherently $O(1)$ per patch, since it only has to compute cosine distance between encoder features and decoder output, and the nearest-neighbour mechanism has $O(N \times d)$ per patch. This might help the model avoid long computation times.

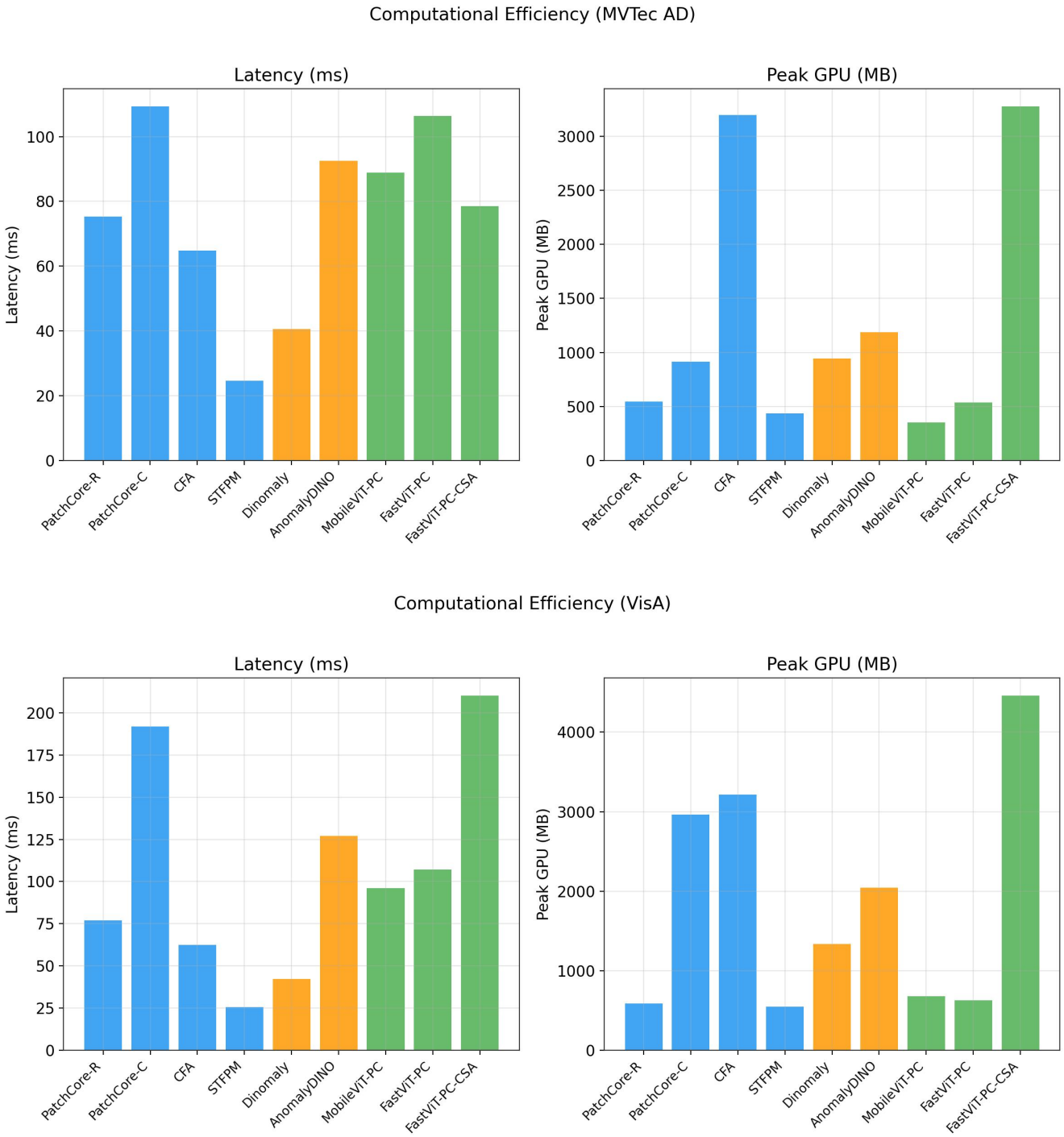


Figure 8: Latency and GPU usage barplots. Each color is a different architecture family: CNNs (blue), transformers (orange), hybrids (green)

The highest latency variance is in the AnomalyDINO model, which has $\pm 128ms$ on the 99th percentile. This suggests that the few-shot memory bank wildly varies between categories, with some categories's features easily represented in a few images, while more complex structures cannot

be extracted into generalizable features with only 8 images.

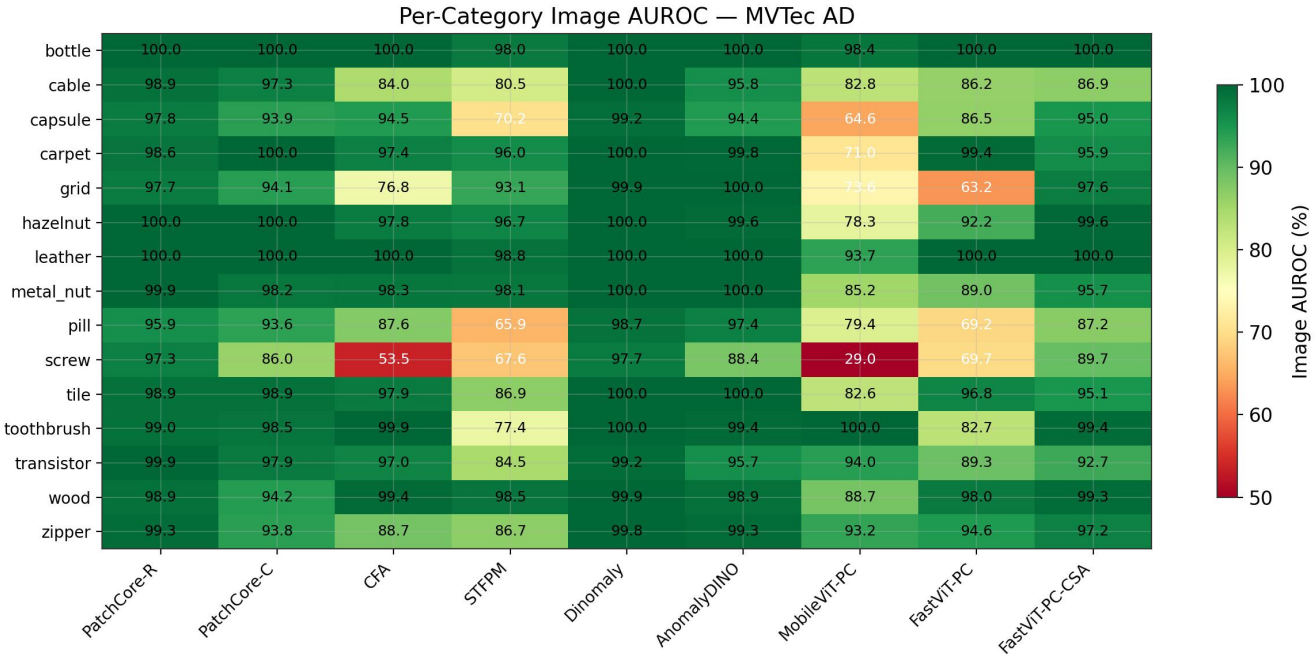
The most GPU-intensive model is FastViT-PC-CSA with a peak usage of 4460MB on the VisA dataset, followed by PatchCore-C with 3126MB peak usage on the VisA dataset, and CFA with 3217MB peak usage. FastViT-PC-CSA peak memory usage is 7× that of FastViT-PC, showing how the CSA mechanism greatly influences the resources needed to compute inference. The CSA attention maps for high-resolution feature maps, which require more compute power, especially when mapping early layers. Since layer0 has the highest resolution, it will lead to high memory usage. Also shown in Table 2 we can see how when using a stage0 memory usage reaches higher peaks compared to all other configurations that start from a lower resolution layer.

Between the CNN PatchCore variants, PatchCore-C is 48.1% slower than PatchCore-R on MVTec AD and 157.7% slower on the VisA dataset. It also reaches higher memory usage peaks, 71.2% higher than PatchCore-R on MVTec AD and 432.5% higher on VisA, more than 5× PatchCore-R peak. This is possibly due to the backbone design. Compared to WideResNet, ConvNeXt makes use of higher resolution feature maps per stage, which inflates the memory bank, increasing the memory usage, and also leads to slowdowns of the nearest-neighbour mechanism. So despite being 3.3× lighter than PatchCore-R, PatchCore-C performs slower and uses more resources, showing how a light model does not imply a fast and efficient one. This counter-intuitive result, a 3.3× lighter model being both slower and more memory-intensive, shows how parameter count is an unreliable predictor of model's efficiency.

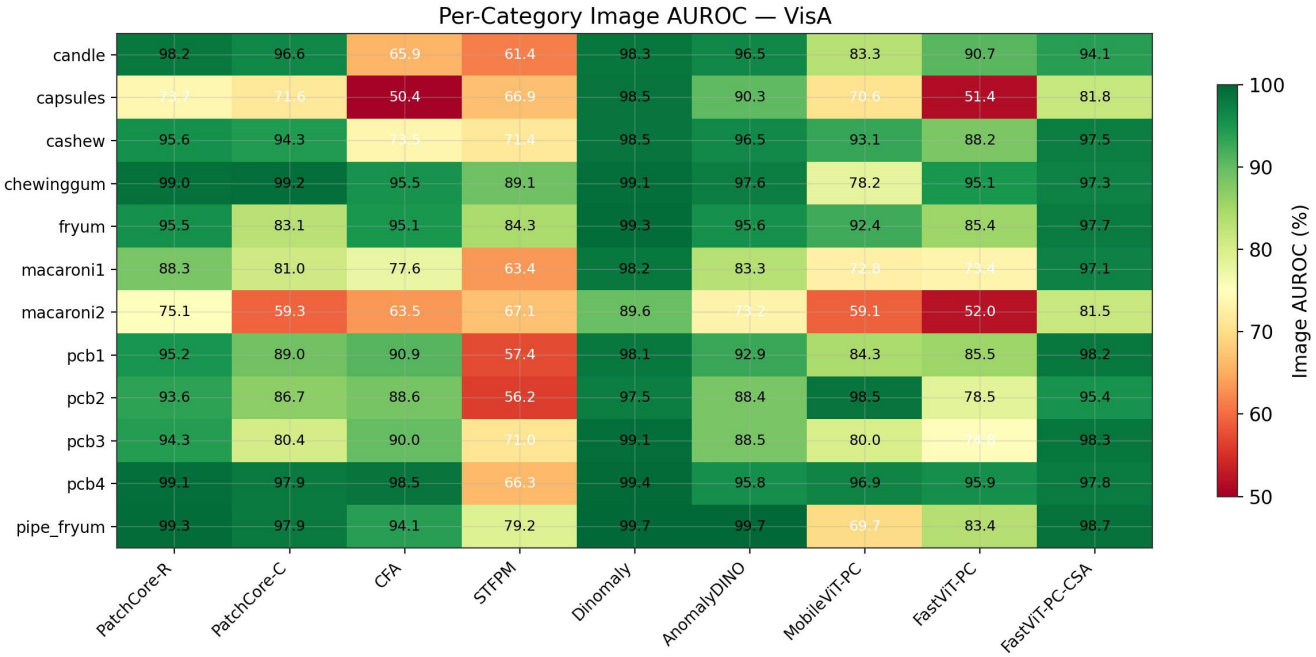
While comparing models, the mean score is as important as the standard deviation of said score. The variance of a model can reflect the reliability of its performance, which, for manufacturers, matters more than absolute accuracy. An unreliable model that is able to score 100% accuracy only half the time is much less useful than a model that is able to consistently score ≈90% accuracy. The most consistent model is Dinomaly, with only 3% of variance at most, while the least reliable models are CFA and MobileViT-PC, with 15.0% and 17.1% respectively. Such large variance means that on some items they will be able to score 100%, while on others it would be an almost random choice.

When comparing structural or textural anomalies, we can see from Figure 9 how all models interact with objects and how capable they are at extracting features. We can see how every model struggles to detect structural anomalies (screw, pcb, macaroni, capsule), while textural anomalies (carpet, grid, leather, tile) are universally easy, with even the weakest models reaching 90% Img-AUROC or higher. The more extreme case is MobileViT-PC, which reaches 100% Img-AUROC on toothbrush, a texture, but in the lowest case reaches 29% Img-AUROC on screw, structural features, performing 3.4× worse. It is the largest accuracy swing between all models.

Our model performance was on par with the most performing ones and scored significantly better than FastViT-PC, its counterpart. It had a 13.6% increase in Img-AUROC and a 23.3% increase in Pxl-AUPRO. The CSA mechanism helps the model increase its accuracy, particularly on structural defects, where FastViT-PC struggles a lot. To further analyze the impact of introducing the CSA fusion mechanism, we compared the standard FastViT with our implementation, and we can see how our model is more accurate but also more resource-intensive than the standard variant. The



(a) MVTec AD



(b) VisA

Figure 9: Anomaly localization heatmaps across all models for representative samples.

CSA fusion mechanism allows for the model to better detect anomalies, but it also increases the amount of computation it has to perform during inference, since it has a new added mechanism to the pipeline, and that results in a ≈30% increase in latency and a 7× peak GPU memory usage. From Table 5 we can see how FastViT-PC-CSA is specifically better in Pxl-AUPRO scorings,

implying that with the CSA fusion mechanism, the model is able to better localize the anomaly. This is consistent with how the CSA fusion works, providing more spatial context that helps place anomaly boundaries with more precision. It also improves performance for diverse types of objects and scales, where multi-scale context matters more. This is especially true in the VisA dataset (9), where both objects and defects might have different scales (from a tiny texture scratch to a large structural deformation). The tradeoff for such improvements is very heavy GPU memory usage.

Metric	**FastViT-PC**	**FastViT-PC-CSA**	**Δ**
MVTec AD			
Image AUROC	87.8	95.4	+7.6
Pixel AUPRO	68.6	80.1	+11.5
VisA			
Image AUROC	79.5	94.6	+15.1
Pixel AUPRO	65.9	85.7	+19.8
Efficiency (pooled)			
Latency (ms)	106.7	137.2	+30.5
Peak GPU (MB)	578	3803	+3225

Table 5: Effect of Cross-Scale Attention fusion: FastViT-PC (plain channel-wise concatenation) versus FastViT-PC-CSA (s0-s3 CSA fusion).

5 Conclusions and Further Research

Quality control is a required procedure in every industrial setting, where accuracy and efficiency are key factors for a successful implementation. In this work we evaluated 9 computer vision models with the goal of creating a unified comparison between different architectures, and we then proposed a novel cross-scale attention fusion mechanism that injects global context into fine-grained features. This allows manufacturers to more appropriately select the model that fits their edge case.

From the results, we saw how memory-bank methods, such as PatchCore-R (95.5% Img-AUROC) or PatchCore-C (91.4% Img-AUROC), match or outperform methods that require training while remaining gradient-free, such as CFA (86.8% Img-AUROC) or STFPM (78.1% Img-AUROC), displaying how strong frozen backbone features can compensate for the absence of training. We also saw how lightweight backbones do not necessarily translate into faster or more efficient models, with MobileViT-PC showing ˜18% higher latency than PatchCore-R (88.8ms vs 75.3ms on MVTec AD), despite being 10× smaller (2.5M parameters vs 24.9M parameters).

The best trade-off between accuracy and efficiency between hybrid models is FastViT-PC-CSA, which, at 10.4M parameters, reaches 95% pooled Img-AUROC with 137ms inference latency. This is thanks to our CSA fusion implementation, without which neither FastViT-PC (83.65% Img-AUROC) nor MobileViT-PC (81.3% Img-AUROC) would achieve competitive accuracy. This improvement is nonetheless not enough for hybrid models to reach the CNN PatchCore-R's performance, which

manages to reach a higher accuracy and a lower latency than all other hybrid models in the comparison. Among transformer-based models, Dinomaly performs competitively on both accuracy (98.8% Img-AUROC) and efficiency (41.4ms latency) benchmarks, outperforming lightweight models despite having 37.4M parameters.

To answer our research question, we have found that, on the MVTec AD and VisA datasets, lightweight transformer architecture does not outperform CNN-based models in terms of efficiency. With our CSA fusion mechanism implementation, we pushed the lightweight model FastViT-PC-CSA's accuracy to match CNN's accuracy, at the cost of 7× GPU memory usage. The most consistent architecture across different categories and datasets was transformer-based, with lightweight hybrids being the most unreliable. Once again, with our CSA fusion implementation, the FastViT model significantly improved its accuracy variance (±14.2% vs. ±6.0% Img-AUROC on the VisA dataset), showcasing how this method could improve the model's performance if enough GPU memory is available.

Several limitations must be taken into account when interpreting these results. Firstly, the training regimes of each model. CFA, STFPM, and Dinomaly have training components (hypersphere adaptation mechanism, student-teacher training, and decoder model, respectively), while all other models, which have memory banks, require no training and rely on the frozen backbone features. This comparison focuses on deployment-ready configurations rather than isolating architecture from the training method.Secondly, the transformer models (Dinomaly and AnomalyDINO) used the DINOv2 backbone, a self-supervised backbone trained on massive amounts of unlabelled data, while other models rely on a ResNet or ViT backbone. This difference makes it difficult to discern if a model's performance is due to its backbone features or its architecture alone. Furthermore, Dinomaly preprocessing differs from all other models, using a 448×448 resolution compared to everyone else's 256×256, giving the model an advantage in accuracy. Thirdly, the benchmark was run on a single Nvidia RTX 3090 GPU. Depending on the changes in hardware, the model's efficiency might vary, depending on the specific hardware memory bandwidth.

The scope of this study is limited to one-class unsupervised anomaly detection. The frameworks in which the models operate are specifically designed for anomaly detection tasks and would not be suitable for other types of computer vision tasks. Moreover, both MVTec AD and VisA are datasets containing controlled factory images; real-world deployment performance remains untested.

To expand the benchmark and explore alternative models, the CSA fusion mechanism could be implemented in all models and, with a configuration sweep, find the most performing configuration for each one. Additional datasets with more complex objects could also show model performance under different circumstances and their ability to generalize.

References

[1] Sachin Mehta and Mohammad Rastegari. "MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=vh-0sUt8HlG.

[2] Yanyu Li et al. "EfficientFormer: Vision Transformers at MobileNet Speed". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 12934–12949. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/5452ad8ee6ea6e7dc41db1cbd31ba0b8-Paper-Conference.pdf.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Communications of the ACM* 60 (May 2012), pp. 84–90.

[4] Aaron Aeberli et al. "Detection of Banana Plants Using Multi-Temporal Multispectral UAV Imagery". In: *Remote Sensing* 13.11 (2021). ISSN: 2072-4292. DOI: 10.3390/rs13112123. URL: https://www.mdpi.com/2072-4292/13/11/2123.

[5] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[6] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[7] Pavan Kumar Anasosalu Vasu et al. "FastViT: A Fast Hybrid Vision Transformer Using Structural Reparameterization". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2023, pp. 5785–5795.

[8] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[9] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks". In: *Proceedings of the British Machine Vision Conference (BMVC)*. Ed. by Edwin R. Hancock Richard C. Wilson and William A. P. Smith. BMVA Press, Sept. 2016, pp. 87.1–87.12. ISBN: 1-901725-59-6. DOI: 10.5244/C.30.87. URL: https://dx.doi.org/10.5244/C.30.87.

[10] Maxime Oquab et al. "DINOv2: Learning Robust Visual Features without Supervision". In: *Transactions on Machine Learning Research* (2024). Featured Certification. ISSN: 2835-8856. URL: https://openreview.net/forum?id=a68SUt6zFt.

[11] Karsten Roth et al. "Towards Total Recall in Industrial Anomaly Detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 14318–14328.

[12] Sungwook Lee, Seunghyun Lee, and Byung Cheol Song. "CFA: Coupled-Hypersphere-Based Feature Adaptation for Target-Oriented Anomaly Localization". In: *IEEE Access* 10 (2022), pp. 78446–78454. DOI: 10.1109/ACCESS.2022.3193699.

[13] Guodong Wang et al. “Student-teacher feature pyramid matching for anomaly detection”. In: *Proceedings of the British Machine Vision Conference 2021*. British Machine Vision Association, 2021.

[14] Jia Guo et al. “Dinomaly: The Less Is More Philosophy in Multi-Class Unsupervised Anomaly Detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2025, pp. 20405–20415.

[15] Simon Damm et al. “AnomalyDINO: Boosting Patch-based Few-Shot Anomaly Detection with DINOv2”. In: *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2025, pp. 1319–1329. DOI: 10.1109/WACV61041.2025.00136.

[16] Manqi Yang. “Efficiency and Adaptability of Deep Learning Architectures: A Performance Comparison of CNN, Transformer, and Hybrid Models”. In: *2025 5th International Conference on Computer Vision, Application and Algorithm (CVAA)* (Aug. 2025), pp. 1–4. DOI: 10.1109/cvaa66438.2025.11193320. URL: https://ieeexplore.ieee.org/abstract/document/11193320.

[17] Tobias Christian Nauen et al. “Which Transformer to Favor: A Comparative Analysis of Efficiency in Vision Transformers”. In: *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2025, pp. 6955–6966. DOI: 10.1109/WACV61041.2025.00676.

[18] Yunusa Haruna et al. “Exploring the synergies of hybrid convolutional neural network and Vision Transformer architectures for computer vision: A survey”. In: *Engineering Applications of Artificial Intelligence* 144 (2025), p. 110057. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2025.110057. URL: https://www.sciencedirect.com/science/article/pii/S0952197625000570.

[19] Alessandro David. “Patchcore Transformed: A Vision Transformer Approach To Anomaly Detection”. MA thesis. Università di Bologna, 2024. URL: https://amslaurea.unibo.it/id/eprint/31435/.

[20] Samet Akcay et al. “Anomalib: A Deep Learning Library for Anomaly Detection”. In: *2022 IEEE International Conference on Image Processing (ICIP)*. 2022, pp. 1706–1710. DOI: 10.1109/ICIP46576.2022.9897283.

[21] Paul Bergmann et al. “MVTec AD – A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[22] Yang Zou et al. “SPot-the-Difference Self-supervised Pre-training for Anomaly Detection and Segmentation”. In: *Computer Vision – ECCV 2022*. Ed. by Shai Avidan et al. Cham: Springer Nature Switzerland, 2022, pp. 392–408. ISBN: 978-3-031-20056-4.

[23] Xiaolong Wang et al. “Non-local Neural Networks”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7794–7803. DOI: 10.1109/CVPR.2018.00813.

[24] Sanghyun Woo et al. “ConvNeXt V2: Co-designing and Scaling ConvNets with Masked Autoencoders”. In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, pp. 16133–16142. DOI: 10.1109/CVPR52729.2023.01548.

A Appendix

Model	Pre	H2D	Forward	Post	Total
PatchCore-R	49.0	0.6	20.9	4.7	75.3
PatchCore-C	15.5	0.4	77.1	18.5	111.5
CFA	35.4	0.6	23.2	5.8	65.0
STFPM	5.0	0.4	10.3	9.0	24.7
Dinomaly	11.5	0.7	19.9	8.4	40.6
AnomalyDINO	52.0	0.7	32.2	7.7	92.5
MobileViT-PC	50.9	0.6	33.5	3.7	88.8
FastViT-PC	49.7	0.6	40.6	15.4	106.4
FastViT-PC-CSA	4.4	0.2	73.4	0.7	78.7

Table 6: End-to-end latency decomposition in milliseconds (mean over MVTec AD categories): preprocessing, host-to-device transfer, backbone forward pass, and postprocessing (memory-bank kNN / scoring). Components may not sum exactly to the total due to per-category averaging. Note: FastViT-PC-CSA has an inverted profile, the CSA non-local attention runs inside the forward pass, and its pipeline measures preprocessing differently.

Table 7: **PatchCore-R** – MVTec AD metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
bottle	100.0	92.4±0.2	535±0	73.9±24.4	186.2±128.6	47.8±21.0	0.6±0.2	20.6±4.3	4.9±4.0
cable	98.9±0.3	90.0±0.4	552	75.6±21.1	188.6±103.6	49.2±19.0	0.6±0.2	21.1±3.4	4.7±3.7
capsule	97.8±0.5	91.5±0.5	552	75.1±27.9	187.8±131.4	48.9±23.1	0.6±0.2	20.9±4.8	4.6±3.7
carpet	98.6±0.2	93.1±0.2	546	76.3±28.4	196.2±144.1	49.6±22.6	0.6±0.2	21.2±5.2	4.8±4.0
grid	97.7±0.3	90.6±0.9	539±1	73.1±30.8	182.1±153.8	47.1±24.3	0.6±0.1	20.6±5.6	4.8±4.0
hazelnut	100.0	93.9±0.2	554±1	75.9±30.6	189.0±151.4	48.9±24.1	0.6±0.2	21.3±6.1	5.0±4.3
leather	100.0	96.6±0.0	546±0	72.7±24.4	174.4±122.2	47.1±21.6	0.6±0.2	20.3±3.8	4.7±3.7
metal_nut	99.9±0.1	91.7±0.2	544±1	69.2±28.8	170.2±128.4	44.2±22.9	0.6±0.2	19.8±5.5	4.7±4.1
pill	95.9±0.6	93.4±0.3	563±1	72.2±24.7	179.6±116.7	46.8±20.2	0.6±0.2	20.1±4.7	4.6±3.8
screw	97.3±0.9	90.3±0.9	565±1	71.1±26.6	178.0±118.9	45.9±21.2	0.6±0.2	20.0±5.1	4.6±3.7
tile	98.9±0.1	79.3±0.2	546±1	78.4±34.4	195.2±160.0	51.6±28.1	0.6±0.2	21.4±6.5	4.7±3.9
toothbrush	99.0±0.7	84.2±0.8	511	81.2±33.8	204.3±139.0	54.0±27.9	0.6±0.2	21.8±5.7	4.9±4.1
transistor	99.9±0.1	94.1±0.2	544±0	78.0±34.0	186.6±127.1	51.4±27.1	0.6±0.2	21.4±6.6	4.5±3.4
wood	98.9±0.2	84.6±0.3	538±1	75.8±34.6	182.3±103.5	49.5±28.1	0.6±0.2	21.2±6.6	4.5±3.6
zipper	99.3±0.1	91.8±0.1	554±0	80.4±35.8	201.1±145.6	53.2±28.1	0.6±0.2	22.0±7.0	4.6±4.0
Mean	**98.8±1.2**	**90.5±4.4**	**546±12**	**75.3±29.6**	**186.8±131.8**	**49.0±24.1**	**0.6±0.2**	**20.9±5.5**	**4.7±3.8**

Table 8: **PatchCore-R** – VisA metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
candle	98.2±0.3	94.6±0.3	602±1	73.4±37.0	171.9±131.4	47.2±29.1	0.6±0.2	20.9±7.0	4.6±4.3
capsules	73.7±1.8	67.5±1.9	575±0	77.0±32.7	193.9±121.7	49.8±25.2	0.6±0.2	21.6±6.6	5.0±4.3
cashew	95.6±0.5	89.0±0.5	564±0	79.7±31.8	193.7±141.7	52.3±25.1	0.6±0.2	21.8±5.8	5.0±4.5
chewinggum	99.0±0.3	84.4±0.5	563±0	73.3±28.4	176.0±101.9	47.1±22.8	0.6±0.2	20.7±5.4	5.0±4.2
fryum	95.5±0.8	78.4±0.9	563±0	80.7±45.9	207.8±179.6	52.7±33.6	0.6±0.2	22.3±9.6	5.0±4.8
macaroni1	88.3±0.7	87.9±0.9	602±0	75.9±31.5	198.7±148.0	48.7±24.4	0.6±0.3	21.7±6.7	4.8±3.8
macaroni2	75.1±1.5	80.3±2.0	602±0	78.7±34.8	197.4±139.4	50.7±27.1	0.6±0.2	22.5±7.3	5.0±4.1
pcb1	95.2±0.4	86.5±0.9	602±0	77.9±38.7	189.5±132.2	49.8±30.6	0.6±0.2	22.5±7.9	5.0±4.2
pcb2	93.6±0.6	80.8±0.9	602±0	82.0±61.6	197.0±158.2	53.4±46.9	0.6±0.2	23.0±13.0	5.0±4.3
pcb3	94.3±0.5	76.3±1.0	602±0	79.6±51.4	200.4±164.7	50.6±38.0	0.6±0.2	23.2±12.3	5.1±4.7
pcb4	99.1±0.2	84.2±0.8	602±1	71.8±25.2	170.5±103.4	45.3±21.0	0.6±0.2	21.1±5.1	4.7±3.9
pipe_fryum	99.3±0.2	92.9±0.2	563±0	73.0±21.8	189.7±107.8	46.9±19.2	0.6±0.2	20.9±3.6	4.6±3.8
Mean	**92.2±8.6**	**83.6±7.3**	**587±18**	**76.9±38.1**	**190.6±137.0**	**49.5±29.4**	**0.6±0.2**	**21.8±8.0**	**4.9±4.2**

Table 9: **PatchCore-C** – MVTec AD metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
bottle	100.0	89.7±0.1	833±149	109.7±39.7	164.4±49.1	15.3±14.8	0.4±0.2	75.2±35.1	18.8±12.0
cable	97.3±0.2	86.7±0.2	888±167	113.7±26.4	169.8±54.8	17.9±19.6	0.4±0.2	76.9±22.5	18.5±12.0
capsule	93.9±0.5	90.6±0.1	875±159	109.8±23.3	165.8±50.5	13.9±13.4	0.4±0.2	75.9±21.7	19.7±11.8
carpet	100.0	95.1±0.0	1022±235	118.7±30.3	168.5±47.7	13.2±13.5	0.4±0.2	87.2±28.7	17.8±10.5
grid	94.1±1.4	82.0±0.3	1021±248	123.2±32.4	182.1±48.7	13.5±13.8	0.4±0.2	91.5±33.3	17.9±10.4
hazelnut	100.0	93.5±0.1	1238±303	135.0±18.9	214.5±48.3	16.6±14.5	0.4±0.2	101.2±14.6	16.7±9.2
leather	100.0	97.1±0.0	933±194	117.5±26.0	177.0±54.5	16.2±16.4	0.4±0.2	81.4±23.3	19.5±11.2
metal_nut	98.2±0.2	90.2±0.1	870±162	107.7±23.2	164.2±59.9	16.6±15.8	0.4±0.2	71.8±11.1	18.8±13.3
pill	93.6±0.2	91.9±0.1	1004±217	116.2±24.6	179.1±63.3	15.8±14.8	0.4±0.2	82.5±14.6	17.5±12.5
screw	86.0±1.7	89.9±0.2	1138±281	126.4±22.5	188.8±60.0	17.2±16.4	0.4±0.2	91.1±15.8	17.7±12.6
tile	98.9±0.2	84.1±0.1	895±175	106.5±26.3	167.0±66.2	16.5±15.8	0.4±0.2	72.0±13.8	17.6±13.2
toothbrush	98.5±0.6	80.3±0.1	600±0	76.1±9.9	109.6±56.1	9.8±15.3	0.3±0.2	37.9±4.2	28.0±11.5
transistor	97.9±0.3	82.5±1.3	851±153	100.1±22.1	154.1±58.7	16.0±15.0	0.4±0.2	66.9±11.1	16.8±12.4
wood	94.2±0.5	85.9±0.1	929±194	105.8±24.8	167.6±63.1	18.3±17.2	0.4±0.2	72.6±11.2	14.5±9.8
zipper	93.8±0.8	88.4±0.2	928±187	105.8±24.4	167.2±70.2	16.2±15.9	0.4±0.2	72.2±13.4	17.1±12.9
Mean	**96.4±3.9**	**88.5±4.9**	**935±145**	**111.5±13.4**	**169.3±21.7**	**15.5±2.2**	**0.4±0.0**	**77.1±14.3**	**18.5±2.9**

Table 10: **PatchCore-C** – VisA metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
candle	96.6±0.6	93.4±0.2	4315±946	240.4±56.4	256.6±49.0	0.7±0.8	0.2±0.1	237.8±57.1	1.6±0.8
capsules	71.6±1.7	51.4±0.9	1793±628	146.2±23.2	239.8±55.5	20.7±13.2	0.5±0.2	117.5±26.4	7.5±4.8
cashew	94.3±0.8	89.8±0.1	1505±479	134.4±24.0	218.3±53.9	19.1±14.3	0.4±0.2	103.9±25.4	10.8±9.8
chewinggum	99.2±0.1	85.3±0.1	1513±483	139.1±23.6	230.7±61.6	20.7±15.2	0.4±0.2	106.4±24.6	11.5±10.3
fryum	83.1±2.2	74.4±0.1	1505±479	140.1±22.7	226.5±60.4	22.2±16.9	0.4±0.2	106.0±23.8	11.4±10.1
macaroni1	81.0±2.4	89.1±0.5	4315±946	237.0±58.1	248.6±56.7	0.5±0.1	0.2±0.0	235.0±58.0	1.3±0.6
macaroni2	59.3±0.6	79.1±0.3	4330±952	243.4±64.6	257.5±72.2	0.6±0.5	0.2±0.0	240.9±64.0	1.6±0.9
pcb1	89.0±2.0	87.4±0.2	4349±956	238.0±63.7	246.3±65.3	0.4±0.1	0.2±0.0	236.1±63.1	1.3±0.9
pcb2	86.7±0.6	83.8±0.3	4335±953	237.8±64.1	248.0±67.3	0.5±0.1	0.2±0.0	235.8±63.4	1.4±0.9
pcb3	80.4±1.5	79.0±0.4	4290±1033	239.6±72.3	256.6±80.6	0.5±0.2	0.2±0.0	237.4±71.5	1.5±1.0
pcb4	97.9±0.8	86.7±0.1	3861±1302	240.5±55.6	259.9±57.7	2.0±3.7	0.2±0.1	232.9±57.9	5.4±9.6
pipe_fryum	97.9±0.2	93.9±0.0	1400±384	141.7±23.6	241.7±51.7	22.1±13.4	0.4±0.2	105.6±25.1	13.6±9.9
Mean	**86.4±12.2**	**82.8±11.5**	**3126±1406**	**198.2±51.2**	**244.2±13.3**	**9.2±10.4**	**0.3±0.1**	**182.9±66.3**	**5.7±4.9**

Table 11: **CFA** – MVTec AD metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
bottle	100.0	91.8±0.1	3191±41	70.7±46.5	142.1±118.3	39.9±36.5	0.6±0.1	24.2±9.4	6.0±6.3
cable	84.0±1.0	90.7±0.2	3209±41	67.1±35.2	126.2±71.2	37.6±26.0	0.5±0.1	23.3±8.1	5.6±6.1
capsule	94.5±0.4	91.1±0.1	3204±40	67.2±35.8	130.6±67.7	37.8±27.1	0.6±0.1	23.5±8.1	5.4±5.7
carpet	97.4±0.2	90.2±0.2	3199±40	66.3±32.8	121.9±62.9	35.5±23.9	0.6±0.1	23.6±7.4	6.6±6.7
grid	76.8±4.0	75.1±12.4	3187±41	67.1±55.9	128.2±137.0	36.4±41.3	0.6±0.1	23.9±13.6	6.1±7.6
hazelnut	97.8±0.2	94.1±0.1	3195±41	65.8±39.1	129.4±91.0	35.8±28.6	0.6±0.1	23.6±9.4	5.8±6.5
leather	100.0	95.6±0.1	3200±41	60.2±28.6	113.2±45.6	31.4±22.5	0.6±0.1	22.6±6.2	5.6±5.6
metal_nut	98.3±1.0	88.4±0.2	3198±42	59.6±25.5	112.2±40.7	30.9±20.2	0.6±0.1	22.4±5.6	5.8±6.2
pill	87.6±1.0	91.1±0.2	3213±41	67.0±35.5	139.0±101.6	36.9±26.1	0.6±0.1	23.7±8.0	5.8±6.2
screw	53.5±0.9	90.9±0.5	3212±41	64.6±35.2	134.3±114.8	36.0±26.3	0.5±0.1	23.0±8.0	5.1±6.5
tile	97.9±0.3	74.4±0.4	3200±41	63.1±31.2	117.3±58.0	33.6±23.5	0.5±0.1	22.9±7.1	6.2±6.7
toothbrush	99.9±0.1	86.2±0.2	3179±41	62.0±31.9	119.7±65.7	34.5±23.6	0.6±0.1	22.6±7.3	4.4±4.2
transistor	97.0±0.7	93.2±0.1	3196±41	62.9±33.3	130.5±108.5	33.3±24.7	0.6±0.1	22.7±7.7	6.4±7.4
wood	99.4±0.2	81.9±0.2	3187±41	65.1±40.4	124.8±100.3	34.8±29.2	0.6±0.1	23.6±9.6	6.2±6.5
zipper	88.7±1.7	89.2±0.4	3209±41	62.8±34.8	116.1±61.3	33.0±24.9	0.6±0.1	22.8±8.1	6.5±7.5
Mean	**91.5±12.2**	**88.3±6.9**	**3199±42**	**64.8±36.5**	**125.7±87.0**	**35.2±27.2**	**0.6±0.1**	**23.2±8.4**	**5.8±6.4**

Table 12: **CFA** – VisA metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
candle	65.9±1.9	89.6±0.5	3221±41	61.8±26.8	115.5±47.9	32.9±20.2	0.6±0.1	22.5±6.0	5.7±6.6
capsules	50.4±4.1	55.9±8.6	3213±41	58.8±26.4	114.8±62.6	29.8±17.6	0.6±0.1	22.0±6.2	6.4±7.0
cashew	73.5±0.6	86.4±0.3	3209±40	60.1±27.5	112.5±51.1	31.4±20.2	0.6±0.1	22.3±6.4	5.9±6.4
chewinggum	95.5±0.3	86.0±0.2	3207±42	61.3±25.4	113.3±40.5	31.9±19.0	0.6±0.1	22.5±6.0	6.4±7.6
fryum	95.1±0.3	78.8±0.3	3208±40	63.6±28.8	119.0±48.3	33.6±20.7	0.6±0.1	22.8±7.0	6.6±7.6
macaroni1	77.6±0.7	93.4±0.3	3221±41	58.1±32.8	109.2±56.5	29.3±23.5	0.5±0.1	22.1±7.9	6.1±6.7
macaroni2	63.5±0.8	60.0±1.4	3223±40	57.4±27.0	111.7±60.5	28.1±17.2	0.6±0.1	22.0±6.4	6.7±6.9
pcb1	90.9±1.7	78.6±1.3	3221±41	63.3±32.3	120.9±66.4	33.6±21.8	0.6±0.1	22.8±7.8	6.3±8.5
pcb2	88.6±0.5	79.0±0.4	3223±40	65.9±34.5	123.7±66.0	36.4±24.6	0.6±0.1	23.1±7.7	5.8±6.5
pcb3	90.0±0.4	81.7±0.5	3223±40	63.2±39.3	118.1±74.0	32.7±26.3	0.6±0.1	23.1±9.4	6.8±7.9
pcb4	98.5±0.1	81.7±0.6	3223±40	68.3±44.9	124.2±80.9	37.2±32.3	0.6±0.1	24.1±10.2	6.5±8.4
pipe_fryum	94.1±0.7	89.3±0.5	3210±39	67.1±37.4	127.0±75.5	36.5±25.4	0.6±0.1	23.6±8.8	6.5±8.9
Mean	**82.0±15.0**	**80.0±11.2**	**3217±41**	**62.4±32.3**	**117.5±61.6**	**32.8±22.7**	**0.6±0.1**	**22.7±7.6**	**6.3±7.4**

Table 13: **STFPM** – MVTec AD metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
bottle	98.0±3.5	67.3±13.5	410±3	25.0±22.5	64.6±40.3	5.2±7.3	0.4±0.1	10.5±6.9	8.9±11.0
cable	80.5±5.5	33.4±5.7	462±1	24.8±22.2	62.2±36.9	5.3±6.8	0.3±0.1	10.5±6.6	8.7±11.1
capsule	70.2±4.0	29.2±4.1	431±1	24.1±21.6	62.7±36.6	4.9±6.6	0.4±0.1	10.2±6.3	8.6±10.7
carpet	96.0±1.9	78.1±4.5	421±1	24.0±21.6	63.9±37.5	4.3±6.1	0.3±0.1	9.9±6.3	9.5±11.4
grid	93.1±2.8	55.6±7.8	411±0	23.6±20.3	64.6±42.9	4.8±6.9	0.3±0.1	10.2±6.3	8.3±9.8
hazelnut	96.7±6.1	66.4±14.2	421±1	25.2±20.7	65.9±38.3	4.9±6.8	0.3±0.1	10.3±6.5	9.6±9.6
leather	98.8±1.5	86.3±7.7	421±1	26.0±24.2	66.5±40.6	5.6±8.9	0.4±0.1	10.7±7.4	9.4±10.8
metal_nut	98.1±1.2	62.1±7.9	421±1	24.0±21.6	63.0±34.6	5.1±7.9	0.4±0.1	10.2±6.6	8.3±10.4
pill	65.9±9.5	29.1±10.9	511±1	24.7±21.5	64.8±38.6	4.8±7.0	0.3±0.1	10.3±6.6	9.3±10.3
screw	67.6±9.0	28.2±10.0	489±1	24.6±20.6	63.4±37.1	4.4±6.4	0.3±0.1	10.1±6.2	9.8±10.2
tile	86.9±7.2	32.2±10.9	421±1	23.0±20.4	61.4±34.9	4.5±6.0	0.3±0.1	10.1±6.3	8.0±9.5
toothbrush	77.4±4.0	52.9±6.7	400±1	23.8±20.4	63.5±35.4	4.8±6.8	0.3±0.2	10.1±6.5	8.5±9.3
transistor	84.5±5.0	30.2±6.1	420±3	25.2±20.2	65.7±37.3	5.1±6.7	0.3±0.1	10.4±6.5	9.3±9.6
wood	98.5±1.1	75.6±5.8	410±0	25.6±20.6	67.1±36.8	5.7±8.3	0.4±0.1	10.8±6.7	8.7±8.8
zipper	86.7±8.2	45.3±13.2	465±1	26.2±22.2	65.3±35.8	5.6±8.6	0.4±0.1	10.7±7.0	9.6±9.9
Mean	**86.6±12.7**	**51.5±21.7**	**434±31**	**24.7±21.2**	**64.3±37.3**	**5.0±7.1**	**0.4±0.1**	**10.3±6.5**	**9.0±10.1**

Table 14: **STFPM** – VisA metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
candle	61.4±6.4	22.2±7.0	601±0	25.7±22.2	63.0±41.5	5.4±9.0	0.4±0.1	10.7±7.3	9.3±9.7
capsules	66.9±4.8	27.0±6.4	489±1	27.2±22.5	67.1±40.9	6.3±9.4	0.4±0.1	11.2±7.4	9.3±9.7
cashew	71.4±12.7	39.8±14.3	462±1	26.7±23.3	67.2±41.3	5.2±8.5	0.4±0.1	10.8±7.6	10.3±10.7
chewinggum	89.1±9.3	37.0±8.7	462±1	25.4±21.8	66.5±38.2	5.2±7.8	0.4±0.1	10.6±6.8	9.2±9.9
fryum	84.3±7.7	31.1±9.4	462±1	25.9±21.9	65.5±40.4	4.7±6.7	0.3±0.1	10.6±6.9	10.2±10.7
macaroni1	63.4±5.6	27.5±4.5	601±0	25.1±22.0	63.1±39.7	4.6±6.8	0.4±0.2	10.3±7.0	9.9±10.3
macaroni2	67.1±5.8	26.7±6.0	600±1	25.2±22.6	61.8±39.2	4.7±6.8	0.3±0.2	10.4±7.3	9.7±10.6
pcb1	57.4±2.8	17.8±1.7	600±3	25.8±22.4	62.4±41.7	5.0±7.1	0.4±0.1	10.6±6.8	9.9±10.8
pcb2	56.2±4.6	17.3±1.6	600±0	24.9±20.6	62.8±35.4	5.1±6.9	0.4±0.1	10.5±6.7	9.0±9.5
pcb3	71.0±4.1	24.7±3.0	603±3	25.0±21.5	61.8±37.9	4.9±6.8	0.3±0.1	10.4±6.7	9.4±10.2
pcb4	66.3±10.9	17.8±2.8	603±3	25.6±22.5	64.3±36.7	5.5±7.5	0.4±0.1	10.7±6.9	9.1±11.2
pipe_fryum	79.2±8.3	36.0±9.1	462±1	23.8±21.4	65.1±36.5	4.6±6.3	0.4±0.1	10.2±6.4	8.7±10.6
Mean	**69.5±12.3**	**27.1±10.2**	**545±66**	**25.5±21.9**	**64.2±38.9**	**5.1±7.5**	**0.4±0.1**	**10.6±6.9**	**9.5±10.2**

Table 15: **Dinomaly** – MVTec AD metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
bottle	100.0	97.0±0.1	750±1	41.7±26.5	85.5±46.5	10.9±9.7	0.7±0.3	21.2±9.6	8.9±10.5
cable	100.0±0.0	95.0±0.2	1139±0	44.5±28.4	87.7±46.4	12.9±11.6	0.8±0.3	20.0±8.5	10.8±9.5
capsule	99.2±0.2	96.7±0.1	1021±0	39.1±19.7	83.9±43.5	11.4±8.8	0.7±0.2	20.0±7.7	6.9±6.4
carpet	100.0±0.0	98.0±0.0	924±1	43.4±28.4	80.2±48.7	12.2±12.4	0.7±0.3	20.2±9.0	10.3±10.5
grid	99.9±0.0	97.7±0.0	738±1	41.5±25.4	84.7±47.9	11.4±9.2	0.7±0.2	20.6±9.1	8.7±9.8
hazelnut	100.0	96.3±0.1	877±2	40.0±27.1	80.4±47.1	12.5±11.8	0.7±0.3	19.0±7.7	7.7±9.0
leather	100.0	98.3±0.1	969±0	37.8±22.1	76.8±42.4	11.5±9.9	0.7±0.2	19.2±7.5	6.5±7.0
metal_nut	100.0	94.6±0.1	942±3	40.1±28.4	79.4±53.1	11.4±11.9	0.7±0.2	19.4±8.9	8.6±8.9
pill	98.7±0.1	97.8±0.0	1248±0	39.4±28.5	74.1±44.9	10.7±11.6	0.7±0.2	19.5±8.7	8.5±10.6
screw	97.7±0.3	98.2±0.1	1202±1	40.7±26.2	80.9±42.7	11.1±10.5	0.7±0.3	20.0±8.3	9.0±9.2
tile	100.0±0.0	91.3±0.2	929±2	37.4±22.2	81.0±43.1	10.7±9.5	0.7±0.2	19.4±7.5	6.6±7.4
toothbrush	100.0	95.1±0.2	711±1	39.4±30.9	75.0±54.5	10.8±13.1	0.7±0.3	18.8±9.6	9.1±9.7
transistor	99.2±0.1	85.1±0.8	820±2	40.6±24.5	87.8±47.2	11.9±10.6	0.7±0.3	20.8±8.9	7.2±7.5
wood	99.9±0.1	94.4±0.1	737±0	45.0±28.8	93.8±52.4	14.3±13.0	0.8±0.3	20.4±8.9	9.4±8.4
zipper	99.8±0.0	97.0±0.1	1146±0	38.1±23.5	79.0±44.0	9.3±9.6	0.7±0.2	19.8±7.8	8.4±9.5
Mean	**99.6±0.7**	**95.5±3.4**	**943±178**	**40.6±2.3**	**82.0±5.3**	**11.5±1.2**	**0.7±0.0**	**19.9±0.7**	**8.4±1.3**

Table 16: **Dinomaly** – VisA metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
candle	98.3±0.2	97.4±0.1	1463±0	43.3±24.5	96.0±51.5	14.4±12.9	0.8±0.3	21.3±8.4	6.8±5.1
capsules	98.5±0.2	97.7±0.2	1204±0	48.8±35.8	99.0±65.2	16.3±21.7	0.8±0.4	21.7±10.1	10.0±8.9
cashew	98.5±0.1	94.0±0.3	1139±0	37.0±22.5	80.6±42.2	9.6±9.1	0.7±0.2	19.8±8.2	6.9±7.2
chewinggum	99.1±0.2	87.4±0.7	1138±1	40.2±24.9	80.1±45.8	10.2±9.7	0.7±0.2	18.9±7.1	10.4±9.5
fryum	99.3±0.1	93.2±0.1	1138±1	40.5±27.0	86.9±48.3	11.8±11.1	0.7±0.3	20.5±9.6	7.5±9.4
macaroni1	98.2±0.2	96.1±0.2	1463±0	40.8±30.5	78.9±45.6	11.5±12.2	0.7±0.3	19.3±9.0	9.2±11.5
macaroni2	89.6±13.0	96.8±1.7	1463±0	43.1±30.3	88.9±57.4	12.8±12.4	0.7±0.3	20.7±9.9	8.9±11.0
pcb1	98.1±0.4	96.4±0.7	1463±0	42.6±31.2	84.6±53.1	12.5±15.0	0.7±0.3	19.5±8.5	9.9±11.9
pcb2	97.5±0.4	92.0±0.8	1463±0	43.3±27.2	90.0±54.6	11.9±12.5	0.8±0.3	21.7±10.1	8.9±7.6
pcb3	99.1±0.1	92.9±0.2	1471	44.0±26.2	88.4±44.6	12.6±10.8	0.8±0.2	20.7±8.2	9.9±9.2
pcb4	99.4±0.1	92.9±0.6	1471±0	40.5±26.2	89.2±45.4	10.2±9.6	0.8±0.2	20.9±9.3	8.6±10.6
pipe_fryum	99.7±0.1	97.0±0.1	1138±0	44.3±26.8	85.4±47.8	12.3±10.7	0.7±0.2	20.3±8.1	11.0±9.7
Mean	**97.9±2.7**	**94.5±3.0**	**1334±162**	**42.4±2.9**	**87.3±6.1**	**12.2±1.8**	**0.7±0.0**	**20.4±0.9**	**9.0±1.4**

Table 17: **AnomalyDINO** – MVTec AD metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
bottle	100.0	94.4	1284	80.2±37.4	196.8±176.8	50.4±29.4	0.7±0.1	21.4±7.6	7.8±6.9
cable	95.8	90.4	1389	83.4±43.7	209.6±185.4	51.8±32.9	0.7±0.2	22.3±9.1	8.5±8.8
capsule	94.4	94.0	504	119.7±66.4	485.6±619.3	57.0±21.9	0.7±0.2	53.7±45.3	8.3±7.1
carpet	99.8	94.5	1689	79.6±30.4	200.0±159.6	48.9±25.3	0.7±0.1	22.0±5.4	8.0±7.9
grid	100.0	95.0	1588	81.2±29.5	192.2±131.6	49.9±24.8	0.7±0.1	22.6±5.5	8.1±7.1
hazelnut	99.6	96.5	1075	120.4±59.9	453.7±383.5	56.9±20.5	0.7±0.1	54.9±40.1	7.9±7.8
leather	100.0	94.3	1498	82.1±29.7	190.2±108.5	50.0±24.7	0.7±0.2	23.0±5.7	8.5±8.0
metal_nut	100.0	93.8	1356	77.4±28.8	184.6±109.8	47.7±23.9	0.7±0.2	20.9±5.4	8.1±7.4
pill	97.4	96.3	826	121.2±55.3	445.5±384.3	57.8±20.8	0.7±0.2	54.6±35.4	8.1±8.0
screw	88.4	83.8	673	120.6±43.7	441.7±348.5	59.1±17.8	0.7±0.2	52.8±27.7	7.9±7.9
tile	100.0	84.7	1412	76.3±28.2	176.8±130.9	47.8±21.2	0.7±0.2	20.9±5.5	6.9±7.0
toothbrush	99.4	92.4	240	115.0±48.6	438.7±436.3	57.3±18.8	0.7±0.1	50.0±29.3	7.0±7.4
transistor	95.7	74.2	1313	76.7±26.0	179.8±107.5	49.0±19.5	0.7±0.2	20.5±5.3	6.4±6.0
wood	98.9	91.4	1494	75.9±22.8	193.9±137.7	47.4±18.4	0.7±0.2	21.1±4.3	6.7±7.3
zipper	99.3	81.8	1478	78.2±22.1	197.3±117.7	48.9±19.1	0.7±0.2	21.6±4.2	7.1±7.0
Mean	**97.9±3.1**	**90.5±6.2**	**1188±417**	**92.5±44.5**	**279.1±304.6**	**52.0±23.1**	**0.7±0.2**	**32.2±26.0**	**7.7±7.4**

Table 18: **AnomalyDINO** – VisA metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
candle	96.5	94.7	4957	141.4±64.7	499.8±438.7	60.6±21.7	0.7±0.1	72.2±43.9	7.9±8.2
capsules	90.3	81.9	2644	122.4±47.1	394.2±291.4	56.7±19.5	0.7±0.1	57.3±29.7	7.7±8.0
cashew	96.5	92.9	763	124.5±58.7	477.8±462.3	59.5±21.0	0.7±0.1	55.8±38.2	8.5±8.9
chewinggum	97.6	89.4	863	118.3±58.2	442.2±429.1	57.4±20.4	0.7±0.2	53.2±37.7	7.0±6.5
fryum	95.6	77.3	630	125.6±64.9	426.9±331.8	58.8±22.1	0.7±0.2	57.4±43.7	8.7±9.4
macaroni1	83.3	67.5	2982	130.5±52.5	458.8±341.7	59.7±20.0	0.7±0.2	62.8±34.0	7.3±8.0
macaroni2	73.2	79.7	2935	120.1±48.1	415.4±307.7	56.6±17.9	0.7±0.1	56.0±31.6	6.7±6.6
pcb1	92.9	86.2	2200	121.2±65.6	447.0±632.8	57.7±21.8	0.7±0.2	54.5±45.5	8.4±8.5
pcb2	88.4	73.8	2149	142.0±89.2	523.2±505.1	62.9±26.9	0.7±0.2	70.5±62.6	7.9±8.5
pcb3	88.5	73.9	1529	128.6±73.4	456.4±447.1	58.9±21.9	0.7±0.2	61.4±54.5	7.7±7.7
pcb4	95.8	84.3	2013	116.1±43.7	452.8±428.4	55.0±17.6	0.7±0.1	53.0±27.5	7.5±6.8
pipe_fryum	99.7	95.4	890	134.2±78.1	501.4±401.9	61.6±24.0	0.7±0.2	63.0±54.0	8.9±7.6
Mean	**91.5±7.2**	**83.1±8.6**	**2046±1198**	**127.1±63.3**	**458.0±425.5**	**58.8±21.3**	**0.7±0.2**	**59.8±43.2**	**7.8±7.9**

Table 19: **MobileViT-PC** – MVTec AD metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
bottle	98.4±0.2	81.7±0.2	307±11	92.9±42.5	185.5±99.1	54.6±32.1	0.7±0.1	34.7±10.4	3.0±1.4
cable	82.8±0.6	60.3±0.4	378±11	92.5±47.6	179.9±108.5	53.6±36.4	0.6±0.1	34.7±11.4	3.5±1.9
capsule	64.6±0.7	59.8±0.7	342±11	87.9±35.8	173.8±80.5	49.7±26.7	0.6±0.1	33.5±8.7	4.1±3.5
carpet	71.0±0.8	66.6±0.6	332±21	88.5±42.1	168.8±84.8	49.6±30.6	0.6±0.1	34.5±11.1	3.8±1.9
grid	73.6±0.9	52.0±0.5	310±15	90.2±41.9	181.4±96.3	52.1±33.2	0.6±0.1	34.0±8.9	3.5±2.0
hazelnut	78.3±0.9	58.2±1.2	402±66	88.4±35.5	183.1±89.8	49.5±27.0	0.6±0.1	35.0±8.8	3.4±1.6
leather	93.7±0.5	88.3±0.3	339±13	87.7±36.7	171.6±81.2	50.5±28.7	0.6±0.1	33.0±8.7	3.5±2.5
metal_nut	85.2±1.0	68.4±0.3	357±15	85.7±34.3	153.7±54.7	48.5±26.0	0.6±0.1	32.8±8.4	3.7±2.2
pill	79.4±2.0	54.7±2.6	428±11	91.3±40.1	174.2±71.7	52.6±30.0	0.6±0.1	34.6±10.3	3.5±2.2
screw	29.0±0.5	61.5±0.8	453±17	87.5±28.6	174.3±72.2	49.3±22.6	0.6±0.1	33.5±6.5	4.0±3.1
tile	82.6±0.5	41.8±0.5	318±11	85.4±31.4	167.0±67.2	48.1±24.2	0.6±0.1	32.8±7.7	3.9±2.9
toothbrush	100.0±0.1	62.6±0.6	274±3	84.5±33.7	159.1±64.6	50.8±26.9	0.6±0.1	29.8±7.3	3.2±1.7
transistor	94.0±0.6	83.9±0.4	311±11	92.3±41.5	176.5±90.0	54.1±32.1	0.6±0.2	33.8±9.7	3.7±2.8
wood	88.7±0.7	43.3±0.2	308±13	88.7±36.2	174.3±81.7	50.8±29.0	0.6±0.1	33.2±7.9	4.1±3.5
zipper	93.2±0.5	80.9±0.3	404±14	88.8±30.1	169.6±61.2	50.2±23.4	0.6±0.1	33.2±6.4	4.8±3.5
Mean	**81.0±17.1**	**64.3±13.8**	**351±54**	**88.8±37.0**	**172.8±80.7**	**50.9±28.5**	**0.6±0.1**	**33.5±8.9**	**3.7±2.5**

Table 20: **MobileViT-PC** – VisA metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
candle	83.3±0.9	60.6±0.9	806±223	96.1±40.8	191.2±109.1	47.6±30.8	0.6±0.1	42.7±9.6	5.1±4.2
capsules	70.6±0.8	29.2±0.4	556±80	99.7±65.3	199.8±178.6	54.4±47.3	0.6±0.1	39.9±18.1	4.8±3.9
cashew	93.1±0.6	20.1±0.5	473±65	86.5±28.0	164.9±72.5	46.1±23.2	0.6±0.1	35.1±5.8	4.7±3.7
chewinggum	78.2±1.8	24.2±0.8	475±69	99.6±71.2	185.3±163.0	55.8±53.0	0.6±0.1	38.7±18.8	4.4±5.1
fryum	92.4±0.4	53.2±0.5	473±67	90.5±29.8	171.3±76.7	49.5±23.7	0.7±0.1	36.2±6.5	4.1±3.0
macaroni1	72.8±0.8	27.4±0.9	812±214	100.8±32.4	182.4±59.5	52.0±25.7	0.6±0.1	43.7±7.4	4.5±2.3
macaroni2	59.1±0.7	22.8±1.1	816±209	105.8±43.5	194.7±78.4	54.5±34.5	0.6±0.1	45.1±9.7	5.6±5.5
pcb1	84.3±0.6	71.8±0.4	811±220	91.1±26.1	174.9±67.6	44.5±20.5	0.6±0.1	41.5±6.1	4.5±2.7
pcb2	98.5±0.2	61.9±0.6	824±199	98.2±34.8	177.4±65.3	51.2±27.0	0.6±0.2	42.4±8.3	4.1±2.3
pcb3	80.0±0.3	62.2±0.7	816±215	93.8±37.3	168.2±62.4	46.6±28.9	0.6±0.1	42.0±8.5	4.5±2.6
pcb4	96.9±0.2	49.4±0.4	826±201	94.4±33.2	176.7±70.6	46.1±24.1	0.6±0.1	42.7±8.9	5.0±3.5
pipe_fryum	69.7±1.4	63.1±2.9	471±65	96.1±46.5	183.9±116.8	53.7±35.6	0.6±0.1	37.8±11.0	4.0±2.5
Mean	**81.6±11.7**	**45.5±18.4**	**680±232**	**96.1±42.6**	**180.9±100.1**	**50.2±32.4**	**0.6±0.1**	**40.6±11.0**	**4.6±3.6**

Table 21: **FastViT-PC** – MVTec AD metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
bottle	100.0	77.5±0.1	523±0	102.5±31.0	207.8±92.8	48.1±24.4	0.6±0.2	39.2±8.2	14.6±8.0
cable	86.2±0.6	57.3±0.3	542±0	105.9±33.0	224.9±155.4	49.8±26.5	0.6±0.1	40.4±8.9	15.1±9.1
capsule	86.5±0.7	71.5±0.2	542	103.4±32.6	202.9±86.3	47.8±26.6	0.6±0.2	39.5±8.5	15.5±8.6
carpet	99.4±0.1	82.2±0.1	542	106.8±32.1	216.5±83.1	51.1±25.5	0.6±0.2	40.7±8.3	14.4±6.7
grid	63.2±1.2	57.0±0.4	532	106.2±33.1	212.4±103.6	48.2±27.4	0.6±0.2	40.8±9.4	16.5±7.8
hazelnut	92.2±0.4	81.8±0.3	559	106.8±40.8	232.7±135.5	51.3±30.8	0.6±0.2	41.0±12.1	13.8±7.9
leather	100.0±0.0	86.8±0.0	537	110.4±41.5	218.8±124.8	52.5±33.5	0.6±0.1	41.4±11.5	15.9±8.3
metal_nut	89.0±0.8	74.5±0.3	534	106.6±36.4	216.2±108.9	48.7±29.3	0.6±0.2	41.1±10.0	16.2±7.5
pill	69.2±0.7	57.0±0.4	556±0	105.7±31.1	223.2±124.4	48.8±26.8	0.6±0.1	40.3±7.7	16.1±7.1
screw	69.7±0.7	67.6±0.7	565	106.2±39.1	215.0±123.9	47.5±29.1	0.6±0.1	41.1±11.8	17.0±7.7
tile	96.8±0.1	68.3±0.0	534	113.1±52.1	222.0±124.2	55.1±41.3	0.7±0.1	42.4±13.6	15.0±8.9
toothbrush	82.7±0.9	52.0±0.4	483	108.6±52.5	213.6±149.1	51.6±39.7	0.6±0.1	40.9±15.8	15.5±7.1
transistor	89.3±0.5	66.6±0.4	532±0	106.7±29.2	214.5±80.3	49.8±25.0	0.6±0.1	40.6±8.0	15.7±8.1
wood	98.0±0.2	57.6±0.1	529±0	105.1±29.6	214.1±138.6	48.7±23.2	0.6±0.1	40.4±7.6	15.4±7.7
zipper	94.6±0.4	71.9±0.2	544±0	101.4±25.5	210.5±101.7	46.7±18.9	0.6±0.1	39.5±7.1	14.6±8.4
Mean	**87.8±11.6**	**68.6±10.4**	**537±18**	**106.4±36.6**	**216.3±116.9**	**49.7±28.9**	**0.6±0.1**	**40.6±10.1**	**15.4±7.9**

Table 22: **FastViT-PC** – VisA metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
candle	90.7±0.4	88.1±0.2	661	106.4±24.5	220.3±83.7	48.0±19.3	0.6±0.1	42.3±7.4	15.5±6.8
capsules	51.4±0.9	24.3±0.6	598	102.6±24.9	210.3±112.6	46.0±17.7	0.7±0.1	40.5±7.1	15.3±6.1
cashew	88.2±0.3	76.9±0.1	576	102.8±28.5	216.9±130.1	46.4±21.1	0.7±0.2	40.3±8.0	15.4±6.6
chewinggum	95.1±0.4	73.3±0.2	577±1	104.2±26.0	205.9±75.3	48.3±20.4	0.6±0.1	40.1±6.9	15.1±6.9
fryum	85.4±0.6	61.8±0.5	576	105.1±32.1	216.9±125.7	49.5±25.1	0.6±0.1	40.9±9.3	14.0±7.5
macaroni1	73.4±0.7	71.6±0.6	661	114.3±33.8	244.1±124.4	53.7±23.8	0.6±0.1	44.0±9.8	15.9±7.4
macaroni2	52.0±0.7	34.2±1.1	664±2	108.6±30.4	221.7±111.5	49.5±24.7	0.6±0.1	42.0±7.6	16.4±7.4
pcb1	85.5±0.5	70.9±0.5	661	112.0±42.2	234.9±118.0	53.4±31.0	0.6±0.1	43.3±11.1	14.7±7.4
pcb2	78.5±0.6	70.6±0.4	661±0	110.2±39.0	225.0±115.1	51.2±29.2	0.6±0.2	42.6±10.0	15.6±9.1
pcb3	74.8±0.5	65.4±0.3	661	109.5±29.6	211.6±81.5	51.1±24.2	0.6±0.1	42.3±7.7	15.5±7.3
pcb4	95.9±0.2	64.6±0.3	683±1	108.3±29.4	216.6±89.0	49.2±22.4	0.6±0.1	42.2±8.2	16.3±6.0
pipe_fryum	83.4±0.3	88.8±0.1	576	101.5±25.4	199.3±67.2	45.9±20.4	0.6±0.1	39.9±7.1	15.1±7.7
Mean	**79.5±14.2**	**65.9±18.3**	**629±42**	**107.1±30.9**	**218.6±104.7**	**49.4±23.5**	**0.6±0.1**	**41.7±8.5**	**15.4±7.2**

Table 23: **FastViT-PC-CSA** – MVTec AD metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
bottle	100.0	87.4±0.0	3231±524	72.0±20.3	81.3±37.9	5.7±15.0	0.2±0.1	65.4±5.4	0.7±0.5
cable	86.9±0.9	70.2±0.2	3274±524	73.3±13.2	76.8±18.0	4.2±10.6	0.2±0.1	68.3±3.4	0.6±0.4
capsule	95.0±0.2	87.8±0.1	3265±525	71.6±12.6	76.6±22.1	3.8±9.6	0.2±0.1	67.0±3.6	0.6±0.5
carpet	95.9±0.4	68.7±0.1	3341±560	87.9±12.6	93.9±21.6	4.1±9.6	0.2±0.1	82.9±4.5	0.7±0.5
grid	97.6±0.2	88.8±0.1	3303±561	83.6±13.5	92.5±28.8	3.6±8.5	0.2±0.1	79.1±5.4	0.7±0.5
hazelnut	99.6±0.1	93.6±0.0	3541±562	116.5±13.9	126.4±30.9	3.2±7.4	0.2±0.1	112.3±7.3	0.8±0.6
leather	100.0±0.0	82.5±0.0	3277±560	80.1±16.2	88.4±32.7	4.4±10.5	0.2±0.1	74.7±5.9	0.8±0.6
metal_nut	95.7±0.3	88.3±0.1	3232±560	73.3±14.8	78.4±22.4	4.6±11.0	0.2±0.1	67.9±4.3	0.7±0.5
pill	87.2±0.2	92.5±0.0	3333±561	84.8±14.3	94.0±28.7	4.7±11.3	0.2±0.1	79.2±4.4	0.6±0.5
screw	89.7±0.2	94.1±0.0	3429±562	97.9±13.7	104.6±20.6	4.7±11.4	0.2±0.1	92.3±4.5	0.6±0.5
tile	95.1±0.2	52.4±0.0	3250±560	76.1±15.5	85.6±30.7	5.1±12.2	0.2±0.1	70.2±3.9	0.6±0.4
toothbrush	99.4±0.1	79.5±0.5	2927±562	32.6±10.8	37.2±16.9	3.6±8.4	0.2±0.0	28.2±2.6	0.6±0.5
transistor	92.7±0.3	76.1±0.2	3219±561	71.4±14.2	78.5±24.9	4.7±11.4	0.2±0.1	65.8±3.7	0.7±0.6
wood	99.3±0.1	66.2±0.1	3272±562	81.2±18.2	87.3±25.7	5.8±14.1	0.2±0.1	74.6±5.0	0.6±0.5
zipper	97.2±0.1	73.2±0.0	3276±560	78.4±13.7	85.4±28.2	3.6±8.3	0.2±0.1	73.8±5.5	0.8±0.6
Mean	**95.4±4.5**	**80.1±12.1**	**3278±129**	**78.7±17.5**	**85.8±18.6**	**4.4±0.8**	**0.2±0.0**	**73.4±17.5**	**0.7±0.1**

Table 24: **FastViT-PC-CSA** – VisA metrics per category (mean ± s.d.).

Category	I-AUROC	AUPRO	GPU (MB)	Total (ms)	p99 (ms)	Pre (ms)	H2D (ms)	Fwd (ms)	Post (ms)
candle	94.1±0.2	92.7±0.2	4893±0	257.2±22.3	268.3±34.1	4.3±10.0	0.2±0.1	251.8±15.2	0.9±0.6
capsules	81.8±0.9	86.0±0.5	4042±0	160.4±18.3	177.0±37.3	5.3±12.4	0.2±0.1	153.9±9.9	0.9±0.7
cashew	97.5±0.4	88.7±0.2	3868±0	147.6±49.6	175.6±100.1	13.5±34.3	0.3±0.1	132.7±15.8	1.1±0.9
chewinggum	97.3±0.4	60.2±0.3	3873	130.4±7.7	142.7±19.8	0.9±1.1	0.2±0.0	128.4±6.7	0.9±0.6
fryum	97.7±0.1	82.5±0.1	3868±0	134.8±11.6	147.9±26.8	1.2±1.7	0.2±0.0	132.2±9.7	1.2±0.8
macaroni1	97.1±0.2	94.2±0.2	4873±52	266.1±26.9	295.4±69.8	0.5±0.1	0.2±0.0	264.3±26.3	1.2±0.7
macaroni2	81.5±0.4	86.1±0.3	4873±52	255.8±18.1	266.3±30.7	0.5±0.2	0.2±0.0	254.1±17.4	1.0±0.7
pcb1	98.2±0.2	88.6±0.1	4893±52	273.6±70.8	286.8±92.0	4.3±10.0	0.2±0.1	267.1±57.3	2.0±3.5
pcb2	95.4±0.5	87.6±0.1	4878±52	254.5±24.1	267.6±45.4	4.6±10.9	0.2±0.1	248.9±15.3	0.8±0.6
pcb3	98.3±0.1	90.5±0.1	4899±52	254.6±21.4	263.6±36.5	4.2±10.0	0.2±0.1	249.5±13.9	0.8±0.6
pcb4	97.8±0.1	78.7±0.2	4894±52	256.4±29.1	269.0±44.7	5.3±12.9	0.2±0.1	250.1±17.4	0.8±0.7
pipe_fryum	98.7±0.1	92.4±0.0	3670±523	132.2±17.5	139.4±28.7	4.9±11.8	0.2±0.1	126.3±7.2	0.7±0.6
Mean	**94.6±6.0**	**85.7±8.8**	**4460±533**	**210.3±66.3**	**224.9±77.7**	**4.1±12.7**	**0.2±0.1**	**205.0±63.7**	**1.0±1.2**