Appendix

A. Implementation Details

A.1. Two-Phase Optimization Strategy

Note that directly optimizing the $\mathcal{L}_g$ (E.q.8) will not make normal patches build strong correlations with most patches in the whole image. Instead, as both $\mathcal{T}^g$ and $\mathcal{S}^g$ have learnable parameters, it's easier to generate trivial solutions [27], where different patches have not learned position-adaptive $\mathcal{T}^g$, and $\mathcal{S}^g$ of normal and abnormal patches also collapse to a similar discrete distribution. Similarly, directly optimizing the $\mathcal{L}_e$ (E.q.10) also can not make each normal patch establish a stronger correlation with a specific external normal pattern, the collapse issue may also exist. To better optimize the intra- and inter-correlation branches, we follow the minimax strategy in [20] to propose a two-phase optimization strategy. Specifically, in the first phase, we minimize the $\mathrm{Div}(\mathcal{T}^g, \mathrm{SG}[\mathcal{S}^g])$ item to make the target correlations adapt to various image patterns of different patches. In the second phase, we maximize the $\mathrm{Div}(\mathrm{SG}[\mathcal{T}^g], \mathcal{S}^g)$ item to force the intra-correlations to pay more attention to the non-adjacent patches with the maximum entropy constraint. Through the two-phase optimization strategy, we can gradually distribute the weights in the intra-correlations to the non-adjacent patches, instead of all image patches forming similar intra-image patch-to-patch correlations, which can effectively reduce the risk of over-fitting. For the inter-correlation branch, the optimization goal is to make each normal patch establish a stronger correlation with a specific external normal pattern, so we first maximize the $\mathrm{Div}(\mathcal{T}^e, \mathrm{SG}[\mathcal{S}^e])$ item and then minimize the $\mathrm{Div}(\mathrm{SG}[\mathcal{T}^e], \mathcal{S}^e)$, which is opposite to the optimization process in the intra-correlation branch. When optimizing intra- and inter-correlations, we also need to add the entropy constraint items: maximizing $\mathrm{Ent}(\mathcal{S}^g)$ for the intra-correlation branch and minimizing $\mathrm{Ent}(\mathcal{S}^e)$ for the inter-correlation branch. Thus, combining the reconstruction loss $\mathcal{L}_l$, the loss functions of two phases are:

$$\begin{aligned}\mathcal{L}_1 =& \mathcal{L}_l + \lambda_1 \mathrm{Div}(\mathcal{T}^g, \mathrm{SG}[\mathcal{S}^g]) - \lambda_1 \mathrm{Div}(\mathcal{T}^e, \mathrm{SG}[\mathcal{S}^e]) \\ \mathcal{L}_2 =& \mathcal{L}_l - \lambda_1 \mathrm{Div}(\mathrm{SG}[\mathcal{T}^g], \mathcal{S}^g) + \lambda_1 \mathrm{Div}(\mathrm{SG}[\mathcal{T}^e], \mathcal{S}^e) \\ & - \lambda_2 \mathrm{Ent}(\mathcal{S}^g) + \lambda_2 \mathrm{Ent}(\mathcal{S}^e)\end{aligned} \tag{13}$$

where $\mathcal{L}_l$ is the reconstruction loss defined in E.q.3, $SG[\cdot]$ means to stop gradient backpropagation, λ_1 and λ_2 are used to trade off the loss items. With the two-phase optimization strategy, each normal patch can establish stronger correlations with most normal patches and a stronger correlation with a specific external normal pattern, this is much harder for anomalies to achieve these correlations, thereby beneficial to amplify the normal-abnormal distinguishability. We further conduct ablation experiments on the direct optimization of $\mathcal{L}_g$ and $\mathcal{L}_e$ and the two-phase optimization strategy, the results are shown in Table 10 of App. B.2.

When implementing the two-phase optimization strategy, we can first calculate the backpropagated gradients of $\mathcal{L}_1$ loss and retain the gradient graph, and then calculate the backpropagated gradients of $\mathcal{L}_2$ loss. The backpropagated gradients calculated in the second phase will be accumulated to the gradients in the first phase, and then we can call the optimizer to update the model parameters.

Table 5. Computation Cost Analysis of our model and other compared models.

Method	FLOPs	Learnable Parameters	Training Time (one epoch)	Inference Speed
DRAEM [64]	198.7G	97.4M	15s	22fps
PatchSVDD [60]	23.6G	15.3M	16s	1.2fps
MKD [41]	24.1G	24.9M	10s	23fps
PatchCore [35]	12.1G	/	14s	19fps
CFLOW [15]	30.7G	24.7M	74s	9.5fps
FOD (Ours)	10.9G	16.2M	15s	21.4fps

A.2. Computation Cost

We provide computation cost analysis of our model and other compared models. All the values are measured with one NVIDIA RTX 3090 GPU and AMD EPYC 7453 28-Core CPU on the MVTecAD dataset, the results are shown in Table 5. For all models, we input a 256 $\times$ 256 image to calculate the FLOPs and set batch size to 4 to estimate the training time. Compared with other models, our model has the same order of magnitude of learnable parameters (PatchCore [35] has no learnable parameters) and fewer FLOPs, but our model can achieve better detection results.

B. Additional Results

B.1. Detailed Results

The detailed pixel-level AUROC results of each category on the MVTecAD dataset are shown in Table 6. The detailed results of each category for anomaly detection and localization performance on the BTAD and MVTec3D-RGB datasets are shown in Table 7 and 8.

Table 7 shows the AUROCs of our method and the SOTA methods for detecting anomalies on the three classes of BTAD. Our FOD can achieve 96.0% mean detection AUROC, which can outperform the best competitor CFLOW [15] by a margin of 1.2%.

The results for individual classes of MVTec3D-RGB are given in Table 8. We are able to outperform all previous SOTA methods regarding the average of all classes by a margin of 3.3%. Note that this dataset is more challenging than the MVTecAD dataset when comparing the best results

Table 6. Detailed pixel-level AUROCs on the MVTecAD dataset.

Category	Pixel-level Anomaly Localization					
	DRAEM [64]	PatchSVDD [60]	MKD [41]	PatchCore [35]	CFLOW [15]	FOD (Ours)
Carpet	0.955	0.953	0.990	0.991	0.994	0.990
Grid	0.997	0.961	0.986	0.988	0.993	0.989
Leather	0.986	0.978	0.978	0.994	0.997	0.995
Tile	0.992	0.911	0.952	0.948	0.969	0.948
Wood	0.964	0.916	0.953	0.954	0.969	0.954
Bottle	0.991	0.978	0.985	0.989	0.988	0.987
Cable	0.947	0.964	0.972	0.985	0.975	0.986
Capsule	0.943	0.958	0.979	0.992	0.989	0.990
Hazelnut	0.997	0.978	0.982	0.986	0.984	0.989
Metal nut	0.995	0.980	0.972	0.980	0.971	0.985
Pill	0.976	0.963	0.971	0.963	0.976	0.986
Screw	0.976	0.957	0.983	0.994	0.988	0.992
Toothbrush	0.981	0.983	0.986	0.988	0.983	0.987
Transistor	0.909	0.970	0.886	0.968	0.923	0.989
Zipper	0.988	0.961	0.981	0.981	0.986	0.977
Mean	0.973	0.961	0.970	0.980	0.979	**0.983**

Table 7. Detailed comparison of our method with the SOTA methods for the image-level anomaly detection and pixel-level anomaly localization performance on the BTAD dataset.

Category	Image-level Anomaly Detection					
	DRAEM [64]	PatchSVDD [60]	MKD [41]	PatchCore [35]	CFLOW [15]	FOD (ours)
Product01	0.995	0.984	0.938	0.984	1.000	0.995
Product02	0.774	0.836	0.882	0.818	0.857	0.864
Product03	0.998	0.951	0.985	1.000	0.987	1.000
Mean	0.922	0.924	0.935	0.934	0.948	**0.960**
	Pixel-level Anomaly Localization					
Product01	0.927	0.948	0.949	0.973	0.971	0.971
Product02	0.936	0.954	0.963	0.961	0.967	0.957
Product03	0.964	0.990	0.983	0.993	0.996	0.996
Mean	0.942	0.964	0.965	0.976	**0.978**	0.975

(99.2% for MVTecAD vs. 88.4% AUROC for MVTec3D-RGB). Nevertheless, we detect defects in 6 out of 10 categories at an AUROC of more than 90%, while other methods only achieve moth than 90% AUROC in most four categories. This demonstrates the robustness of our method.

B.2. Ablation Results

Ablation results in pixel-level AUROC are shown in Table 9. The pixel-level AUROC results demonstrate the same conclusion as in Table 4: the three key designs in our model: recognition views, entropy constraint, and reference features are all effective. These results also verify that our proposed explicit correlation learning approach is effective and the intra- and inter-image correlations are complementary factors to the patch-wise representation discrepancies.

Optimization Strategy. Ablation results in optimization strategy are shown in Table 10. Directly optimizing the $\mathcal{L}_g$ and $\mathcal{L}_e$ cannot make the intra-correlations pay more attention to the non-adjacent areas and will force the inter-correlations to pay more attention to diverse normal patterns. Moreover, direct optimization will cause the optimization problem of RBF kernel [27], thus cannot strongly amplify the difference between normal and abnormal patches as expected. The two-phase optimization strategy first optimizes the target-correlations to provide better guidance to the intra- and inter-correlations. Thus, the two-phase optimization strategy obtains more distinguishable correlation distributions than direct optimization and thereby performs better.

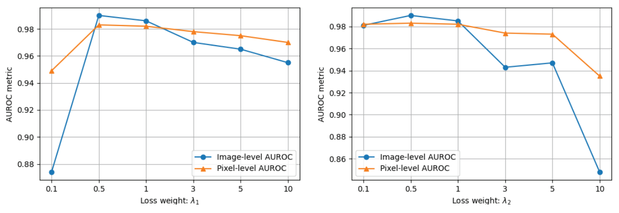

Figure 5. Hyper-parameter sensitivity for loss weights λ_1 and λ_2.

Hyper-parameter Sensitivity. We adopt the loss weights λ_1 and λ_2 to trade off the reconstruction loss, the correlation part and the entropy constraint part. The loss weight hyper-parameters λ_1 and λ_2 are set to 0.5 and 0.5 by default in the main text through comprehensive ablation experiments. To illustrate the sensitivity of our model, we further provide the model performance under different choices of the loss weights. Note that to avoid too many experiments, we only conduct experiments on the MVTecAD dataset, and fix λ_2 to 1 to change λ_1 and then fix λ_1 to the best value to change λ_2. The ablation results are shown in Figure 5 and Table 11. It can be found that λ_1 and λ_2 are stable and easy to tune in the range of 0.5 to 1. The results verify that our model is not very sensitive to the loss weight hyper-parameters, which is essential for applications.

We also show hyper-parameter sensitivity for the number of heads and layers in Table 12 and 13, respectively. It can be found that when setting the number of heads to 8 and the number of layers to 3 can achieve the best result. Thus, we use 8 and 3 as the default values in the main text.

Feature Levels. Besides, we also explore the impact of different network layers on model performance and show the results in Table 14. For single-layer features, $8\times$ one-layer yields the best result as it trades off both semantic representation capability and fine-granularity of the features. Multi-scale feature fusion helps to improve the detection performance as it's conducive to cover more types and scales of anomalies. Note that using the $\{4\times, 8\times, 16\times\}$ three-layer features doesn't gain significant performance improvement compared with $\{8\times, 16\times\}$ two-layer features, but it instead increases the computational cost. Therefore, we use $\{8\times, 16\times\}$ two-layer features by default throughout the main text.

B.3. External Reference Features

External reference features are used for providing accumulated knowledge of normality for the inter-correlation learning branch. Thus, these features should represent all possible normal patterns of all normal samples. To this end, we can employ many methods to generate the external refer-

Table 8. Detailed comparison of our method with the SOTA methods for image-level anomaly detection and pixel-level anomaly localization performance on the MVTec3D-RGB dataset.

Image-level Anomaly Detection											
Method	Bagel	Cable	Carrot	Cookie	Dowel	Foam	Peach	Potato	Rope	Tire	**Mean**
DRAEM [64]	0.988	0.445	0.819	0.635	0.759	0.862	0.849	0.506	0.986	0.724	0.757
PatchSVDD [60]	0.892	0.831	0.570	0.695	0.722	0.626	0.618	0.653	0.999	0.827	0.743
MKD [41]	0.940	0.616	0.782	0.275	0.656	0.736	0.684	0.703	0.910	0.575	0.688
PatchCore [35]	0.887	0.939	0.903	0.703	0.972	0.809	0.750	0.581	0.959	0.884	0.839
CFLOW [15]	0.973	0.887	0.871	0.789	0.989	0.735	0.810	0.692	0.983	0.786	0.851
FOD (Ours)	0.940	0.952	0.911	0.844	0.987	0.844	0.843	0.662	0.992	0.864	**0.884**
Pixel-level Anomaly Localization											
DRAEM [64]	0.977	0.972	0.985	0.930	0.982	0.959	0.981	0.984	0.984	0.983	0.976
PatchSVDD [60]	0.953	0.923	0.817	0.857	0.870	0.897	0.907	0.792	0.709	0.790	0.852
MKD [41]	0.991	0.974	0.989	0.957	0.977	0.896	0.975	0.977	0.986	0.977	0.970
PatchCore [35]	0.959	0.979	0.982	0.967	0.968	0.988	0.977	0.979	0.987	0.985	**0.977**
CFLOW [15]	0.984	0.982	0.984	0.974	0.987	0.900	0.982	0.983	0.980	0.981	0.974
FOD (Ours)	0.988	0.992	0.992	0.979	0.995	0.862	0.989	0.987	0.992	0.982	0.976

Table 9. Ablation results in recognition views, entropy constraint, external reference features and anomaly scoring. *Patch-wise*, *Intra* and *Inter* mean patch-wise discrepancy, intra- and inter-correlation, respectively. *w/o* and *w/* mean without and with entropy constraint. *Mean* and *Coreset* refer to mean and coreset features [35] as the external reference features. *Rec*, *Div* and *Rec&Div* mean the pure reconstruction criterion, pure KL divergence (E.q.6) and the combined criterion (E.q.12).

Recognition Views	Entropy Constraint	Reference Features	Anomaly Scoring	MVTecAD	BTAD	MVTec3D-RGB
Patch-wise	/	/	Rec	0.974	0.975	0.964
Intra	w/o	/	Div	0.717	0.602	0.778
	w/	/	Div	0.804	0.620	0.863
	w/	/	Rec&Div	0.972	0.970	0.961
Inter	w/	Mean	Rec&Div	0.978	**0.976**	0.964
	w/	Coreset	Rec&Div	0.948	0.897	0.928
Intra+Inter	w/	Mean	Div	0.846	0.831	0.965
Patch-wise+Intra +Inter (Ours)	w/	Mean	Rec&Div	**0.983**	0.975	**0.976**

Table 10. Ablation results in optimization strategy. *Direct* and *Two-phase* mean direct optimization of $\mathcal{L}_g$ and $\mathcal{L}_e$ and the two-phase optimization strategy, respectively.

Dataset	MVTecAD		BTAD		MVTec3D-RGB	
Strategy	Direct	Two-phase	Direct	Two-phase	Direct	Two-phase
Image-level AUROC	0.986	0.991	0.959	0.960	0.828	0.884
Pixel-level AUROC	0.957	0.982	0.956	0.975	0.952	0.976

Table 11. Hyper-parameter sensitivity for loss weights λ_1 and λ_2.

λ_1	0.1	0.5	1	3	5	10
Image-level AUROC	0.874	0.990	0.986	0.970	0.965	0.955
Pixel-level AUROC	0.949	0.983	0.982	0.978	0.975	0.970
λ_2	0.1	0.5	1	3	5	10
Image-level AUROC	0.981	0.990	0.985	0.943	0.947	0.848
Pixel-level AUROC	0.982	0.983	0.982	0.974	0.973	0.935

Table 12. Hyper-parameter sensitivity for the number of heads.

Number of heads	2	4	8
Image-level AUROC	0.987	0.990	0.992
Pixel-level AUROC	0.982	0.983	0.983

ence features, such as mean features, nearest features, sampling key features by coreset subsampling algorithm [35], generating prototype features by memory module [28], and learning codebook features through vector quantization [65] or sparse coding techniques [55]. However, because the RBF-kernel in $\mathcal{T}^e$ is position-sensitive, the reference features are better to preserve the positional information. In the following, we will introduce how to generate reference features in detail.

Table 13. Hyper-parameter sensitivity for the number of layers.

Number of layers	2	3	4	5	6
Image-level AUROC	0.986	0.990	0.983	0.979	0.957
Pixel-level AUROC	0.982	0.983	0.978	0.978	0.969

Table 14. Ablation results in feature levels. The experiments are conducted on the MVTecAD dataset. $4\times$, $8\times$, and $16\times$ mean feature maps with $\{4\times, 8\times, 16\times\}$ downsampling ratios, respectively.

Feature Level	4×	8×	16×	8×&16×	4×&8×&16×
Image-level AUROC	0.885	0.981	0.975	0.990	0.988
Pixel-level AUROC	0.932	0.981	0.970	0.983	0.983

Mean Features. Using patch-wise averaged features as the external reference features is really simple but effective. Formally, for position (i, j), we first extract the set of patch features at (i, j), $X_{ij} = \{x_{ij}^k\}, k \in [1, N]$ from the N normal training images. Then, the reference features at position (i, j) is computed as $x_{ij}^f = \frac{1}{N}\sum_{k=1}^{N} x_{ij}^k$. The final external reference features are composed of averaged features at all locations and then flattened into 1D: $X_f = \text{Flatten}(\{x_{ij}^f\})$.

Nearest Features. To represent all possible normal patterns and also preserve the positional information, another simple way is to retain all normal features and then select the nearest features as the reference features. Specifically, we first extract the features of all images from the normal training set, which are denoted as $\mathcal{X} = \{X^k\}_{k=1}^N \in \mathbb{R}^{N\times d\times H\times W}$. Then, for each position (i, j), we select its nearest normal feature in the $p \times p$ neighborhood as the reference feature x_{ij}^f. The $p \times p$ neighborhood is defined as follows:

$$\mathcal{N}^p_{(i,j)} = \{(i',j')|i' \in [i - \lfloor p/2 \rfloor, i + \lfloor p/2 \rfloor], \\ j' \in [j - \lfloor p/2 \rfloor, j + \lfloor p/2 \rfloor]\} \quad (14)$$

The reference feature x^f_{ij} is calculated as follows:

$$x^f_{ij} = \underset{x \in \mathcal{X}^p_{(i.j)}}{\operatorname{argmin}} ||x_{(i,j)} - x||_2 \quad (15)$$

where $\mathcal{X}^p_{(i.j)} = \{x^k_{(i',j')}|(i',j') \in \mathcal{N}^p_{(i,j)}, k = 1, 2, \ldots, N\}$ is the neighborhood features for position (i,j), $x_{(i,j)}$ is the input feature at position (i,j).

Coreset Features. Following [35], we can employ a coreset subsampling algorithm to sample key features as the reference features. The normal features $\mathcal{X} = \{X^k\}^N_{k=1}$ are also first extracted by a pre-trained network. Then, we can establish a coreset feature pool $\mathcal{X}^C$ by the coreset subsampling mechanism. Conceptually, coreset feature pool $\mathcal{X}^C$ aims to most closely and especially more quickly approximate the original features $\mathcal{X}$ in the feature space. Therefore, it can effectively preserve the key normal patterns in normal features. The *minimax facility location coreset selection* algorithm is utilized, the procedure to generate $\mathcal{X}^C$ can be defined as follows:

$$\mathcal{X}^{C*} = \underset{\mathcal{X}^C \subset \mathcal{X}}{\operatorname{argmin}} \max_{x_1 \in \mathcal{X}} \min_{x_2 \in \mathcal{X}^C} ||x_1 - x_2||_2 \quad (16)$$

The exact computation of $\mathcal{X}^{C*}$ is NP-Hard. We follow [35] to use the iterative greedy approximation strategy to sample each coreset feature. The ith coreset feature x^c_i in the coreset feature pool is sampled as follows:

$$x^c_i \leftarrow \underset{x \in \mathcal{X} - \mathcal{X}^C}{\operatorname{argmax}} \min_{x^c \in \mathcal{X}^C} ||x - x^c||_2 \quad (17)$$

Then the coreset feature pool $\mathcal{X}^C$ is updated by $\mathcal{X}^C \leftarrow \mathcal{X}^C \cup \{x^c_i\}$. We can repeat the sampling process (E.q.17) until the pre-defined coreset size.

Prototype Features. In MemoryAE [28], the authors propose to use a memory module to generate prototype features of normal data for lessening the powerful reconstruction capability of CNNs to abnormal video frames. The memory module contains P prototypes recording various prototypical patterns of normal data. However, the prototype features used in our method are slightly different, we need to learn M prototype features at each location to preserve the position information. We denote prototype features at position (i,j) by $\mathcal{P}_{(i,j)} = \{p^m_{(i,j)}\}^M_{m=1}$. We then perform the memory writing operation to update the prototype features.

To update each prototype feature $p^m_{(i,j)}$ at position (i,j), we first need to select all input features declaring that the $p^m_{(i,j)}$ is the nearest one. Thus, we compute the cosine similarity between each input feature $x^k_{(i,j)}$ and all prototypes $\mathcal{P}_{(i,j)}$. The matching weights $w^{k,m}_{(i,j)}$ are as follows:

$$w^{k,m}_{(i,j)} = \frac{\exp(x^k_{(i,j)}(p^m_{(i,j)})^T)}{\sum^M_{m'=1} \exp(x^k_{(i,j)}(p^{m'}_{(i,j)})^T)} \quad (18)$$

Note that multiple input features can be assigned to a single prototype in the memory. We denote by U^m the set of indices for the corresponding input features for the mth item in the memory. We update the mth prototype using the input features indexed by the set U^m as follows:

$$p^m_{(i,j)} \leftarrow L_2(p^m_{(i,j)} + \sum_{k \in U^m} \nu'^{k,m}_{(i,j)} x^k_{(i,j)}) \quad (19)$$

where L_2 means the $L2$ normalization. By using a weighted average of the input features, we can concentrate more on the input features close to the prototype. To this end, we can compute matching weights $\nu^{k,m}_{(i,j)}$ similar to E.q.18:

$$\nu^{k,m}_{(i,j)} = \frac{\exp(x^k_{(i,j)}(p^m_{(i,j)})^T)}{\sum^K_{k'=1} \exp(x^{k'}_{(i,j)}(p^m_{(i,j)})^T)} \quad (20)$$

and renormalize it as follows:

$$\nu'^{k,m}_{(i,j)} = \frac{\nu^{k,m}_{(i,j)}}{\max_{k' \in U^m} \nu^{k',m}_{(i,j)}} \quad (21)$$

Codebook Features. Besides, we can also employ vector quantization (VQ) [49] to learn codebook features as the reference features. Codebook features are highly semantic as VQ is based on quantizing the input features with features from a codebook $D \in \mathbb{R}^{N_e \times d}$ which has been trained for optimal decoding of spatial configurations of quantized features into high-fidelity images. For each input feature $x_{(i,j)}$ at position (i,j), we can obtain a quantized feature representation $z_{(i,j)}$ by replacing the feature vector $x_{(i,j)}$ with its nearest neighbor e_k in D:

$$z_{(i,j)} = e_k, \quad where\ \ k = \underset{j}{\operatorname{argmin}} ||x_{(i,j)} - e_j||_2 \quad (22)$$

After quantizing the input features to the codebook features, we feed the quantized features to a decoder. The decoder output feature $o_{(i,j)}$ at position (i,j) aims at reconstructing the input feature $x_{(i,j)}$. During learning the codebook features, we maximize the cosine similarity between the decoder output $o_{(i,j)}$ and the input $x_{(i,j)}$. Note that the quantization process (E.q.22) is non-differentiable, but we could approximate the gradient similar to the straight-through estimator and directly copy gradients from decoder input $z_{(i,j)}$

Table 15. Ablation results in external reference features. *Mean*, *Nearest*, *Coreset*, *Prototype*, and *Codebook Features* refer to mean, nearest, coreset [35], prototype [28], and codebook [65] features as the external reference features, respectively. ·/· means image-level and pixel-level AUROCs, respectively.

Reference Features	Dataset		
	MVTecAD	BTAD	MVTec3D-RGB
Mean Features	0.990/0.983	0.960/0.975	0.884/0.976
Nearest Features	0.973/0.969	0.947/0.973	0.842/0.970
Coreset Features [35]	0.925/0.948	0.884/0.897	0.700/0.928
Prototype Features [28]	0.987/0.978	0.958/0.975	0.898/0.982
Codebook Features [65]	0.970/0.970	0.955/0.970	0.797/0.956

to input feature $x_{(i,j)}$ [49]. The learning objective is defined as:

$$\max \sum_{i=1}^{H} \sum_{j=1}^{W} \cos(o_{(i,j)}, x_{(i,j)}) \\ - ||sg[x_{(i,j)}] - e_k||_2 - ||x_{(i,j)} - sg[e_k]||_2 \quad (23)$$

With the codebook features, we can use the quantized feature $z_{(i,j)}$ as the reference feature x^f_{ij} as position (i, j).

Results. Ablation results in external reference features are shown in Table 15. As we mentioned, the reference features are better to preserve the positional information, the results also show that the methods (*e.g.*, Coreset Features and Codebook Features) that can't preserve the position information performs worse. Prototype features can achieve comparable performance with mean features, but it's more intricate to generate prototype features by memory module [28]. The ablation results demonstrate that although the mean features are simple, they are quite effective and can achieve better results than these more intricate reference feature generation methods.

C. Qualitative Results

We present in Figure 6 additional anomaly localization results of categories with different anomalies in the MVTecAD dataset.

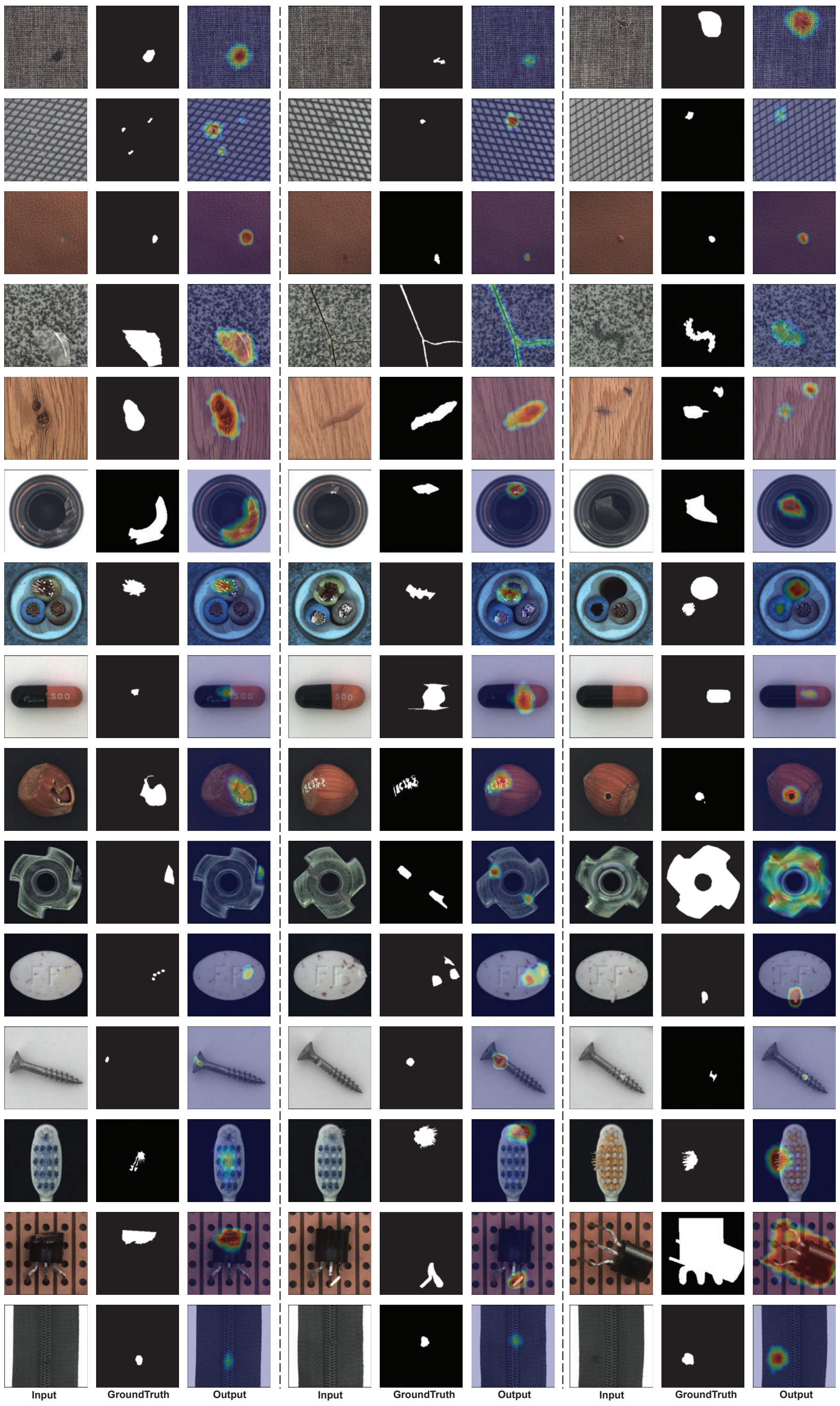

Figure 6. Visualization of anomaly localization maps generated by our method on industrial inspection data. All examples are from the MVTecAD dataset.