ChatGPT

AI-Enhanced Fiber Optic Defect Detection Pipeline Proposal

Introduction

Fiber optic end-face inspection is currently handled by a multi-stage pipeline involving image preprocessing, region segmentation, defect detection, and anomaly analysis. The existing system uses traditional computer vision (e.g. thresholding, Hough circle detection, and handcrafted feature algorithms) to segment the fiber core/cladding and identify defects [1]. While this approach works, it relies on manual parameter tuning and consensus of 11 algorithms, leading to brittle performance under varying lighting or noise conditions [2]. Moreover, anomaly detection is limited by rule-based logic and a knowledge base, which may miss subtle defects or raise false alarms [3] [4]. To improve accuracy, robustness, and automation, we propose an **end-to-end AI upgrade** that integrates state-of-the-art deep learning into all pipeline stages. The new design will leverage advanced **segmentation networks** and **unsupervised anomaly detection models** to automatically learn defect features, even without a labeled dataset. We outline below the proposed architectures, training strategies, module replacements, and methods for performance monitoring and explainability, all while keeping the system modular for batch or interactive use.

1. Deep Learning Architecture Enhancements

Segmentation Network (Fiber & Defect Segmentation): We recommend replacing the "UnifiedSegmentationSystem" with a deep neural network for image segmentation. A **U-Net** architecture (convolutional encoder-decoder with skip connections) is a strong choice for segmenting both fiber regions (core, cladding, ferrule) and defect areas. U-Net variants have proven effective in pixel-wise defect localization [5]. To handle extremely small scratches or specks, we propose an *attention-augmented U-Net*: incorporating **spatial attention** layers or Transformer blocks in the decoder to focus on fine details [6] [7]. Recent research shows that removing excessive downsampling and adding attention feed-forward modules preserves small anomaly features [7]. For example, a modified U-Net with a hybrid residual block (combining spatial attention and Transformer-style feed-forward networks) achieved superior segmentation of tiny defects [6]. The segmentation network can be multi-task: one head segments the **fiber zones** (core/cladding) and another head outputs an **anomaly mask**, or we deploy two separate models (one for region segmentation and one for defect segmentation) depending on training data availability. By using deep segmentation, we eliminate the need for 11 separate algorithms and consensus logic, as the network will learn to delineate regions and defects in one coherent model [1]. This yields a more robust and unified segmentation stage.

Defect Detection & Anomaly Localization Model: For the defect detection stage, we propose an AI module that can localize anomalies without requiring labeled defect samples. A **deep autoencoder** architecture is well-suited here: it learns to **reconstruct normal fiber images** and highlights defects by reconstruction error [8]. Specifically, a **convolutional autoencoder** or **variational autoencoder (VAE)** can be trained on defect-free end-face images to model the "normal" appearance. At inference, any deviation (scratch, dirt) will not be reconstructed correctly, resulting in a difference image that pinpoints the defect

[9]. This approach falls under *unsupervised anomaly segmentation*, as it only needs normal samples for training [10]. We can enhance the autoencoder with skip connections (U-Net style) so that it better preserves spatial detail of anomalies. Alternatively, a **Vision Transformer**-based encoder-decoder could be used for anomaly detection: e.g. a **Masked Autoencoder (MAE)** pre-trained on generic images can be fine-tuned on fiber images to reconstruct missing patches [11]. Transformer models capture global context and have shown success in anomaly localization tasks [11]. Another option is a **GAN** (Generative Adversarial Network) where the generator learns to produce normal-looking fiber images and anomalies are detected via generator loss or discriminator response – though GANs can be trickier to train with limited data.

Architecture Comparison: To choose the optimal design, we compare candidate architectures in the table below, considering their suitability given no labeled dataset:

Architecture	Role in Pipeline	Training Data Needs	Advantages	Drawbacks
U-Net CNN (2D)	Segmentation of fiber regions and defects (supervised or semi-supervised)	Requires segmentation masks (can be synthetic or few manual labels)	Excellent pixel-level localization; efficient on moderate data; well-proven for defects [5].	Needs labeled masks or proxy labels; may miss global context for varied backgrounds.
Transformer-Based U-Net (e.g. Swin-Unet)	Segmentation (advanced)	Pre-training or more data beneficial (can use self-supervision)	Global receptive field captures subtle, context-dependent defects; attention improves small defect detection [7].	Larger model, higher compute; requires careful tuning, risk of overfitting with small data.
Convolutional Autoencoder	Anomaly detection (unsupervised)	Only normal images (no labels) [10]	Learns normal pattern and flags any deviation; simple training, produces anomaly heatmap via reconstruction error [9].	Might reconstruct small anomalies as normal (false negatives) if too powerful; needs balance to not overfit normal.
Variational Autoencoder	Anomaly detection (unsupervised)	Only normal images	Models distribution of normal images; can detect outliers by low probability reconstructions.	Reconstructions can be blurry, reducing defect localization accuracy; training complexity slightly higher.

Architecture	Role in Pipeline	Training Data Needs	Advantages	Drawbacks
Pretrained CNN + One-Class Model (e.g. ResNet + feature modeling)	Anomaly detection (unsupervised)	Only normal images (pretrained on ImageNet)	Leverages robust pretrained features; can use distance or clustering in feature space to spot anomalies [12].	May not capture fiber-specific textures if domain gap is large; needs fine-tuning to reduce false positives.
GAN (e.g. f-AnoGAN)	Anomaly detection (unsupervised)	Only normal images	Can produce sharp reconstructions; discriminator may directly flag anomalies.	GANs are hard to train on small data; mode collapse or unstable training can occur, and it requires longer training time.
Vision Transformer (ViT) + Decoder	Anomaly segmentation (unsupervised or few-shot)	Large pretraining (self-supervised) + normal images	Highly expressive features; patch-level anomaly attention; state-of-art results in some anomaly benchmarks [11].	Very data-hungry unless used with heavy pretraining; inference might be slower (patch-based processing).

Recommendation: Based on the above, a **hybrid solution** is ideal: use a **CNN-based segmentation network (U-Net)** with attention mechanisms for **region and defect segmentation**, and complement it with an **autoencoder-based anomaly detector** to catch subtle defects. This combination ensures both **deterministic segmentation** of known structures (fiber core/cladding) and **flexible detection** of novel anomalies via reconstruction difference [8]. The architectures are modular: the U-Net can output a coarse defect mask (or at least focus on segmenting the fiber regions), while the autoencoder produces a detailed anomaly heatmap. The two can be fused – for example, limit the autoencoder's search to the fiber region mask from U-Net to reduce false positives in the background.

2. Leveraging Pre-Trained Models & Self-Supervision

Because we lack a labeled training set, we will capitalize on **pre-trained models and self-supervised learning** to initialize and train our networks:

- **Pre-trained Backbones:** We will initialize the segmentation U-Net encoder with a CNN pre-trained on ImageNet (e.g. ResNet50 or EfficientNet). These models have learned rich generic features (edges, textures) that can transfer to detecting fiber features [11]. Fine-tuning such a backbone on even a small synthetic or unlabeled dataset can significantly speed up convergence and improve accuracy. Similarly, a Vision Transformer pre-trained via self-supervised learning (like DINO or MAE on ImageNet) can be used as an anomaly feature extractor – its embeddings can help distinguish normal vs abnormal patterns without task-specific labels [11].

- **Self-Supervised Feature Learning:** We propose to pre-train on the available unlabeled fiber images using self-supervised tasks. Possible strategies include:

- *Contrastive Learning:* Apply augmentations (rotation, blur, etc.) to fiber images and train an encoder (CNN or ViT) to maximize agreement between augmented views of the same image (SimCLR/MoCo approach). This yields representations that cluster similar textures and could separate defect regions as outliers.
- *Masked Image Modeling:* Use a Masked Autoencoder (MAE) – mask random patches of the image and train a transformer encoder-decoder to reconstruct them. This task forces the model to understand the context of fiber structures to fill in missing parts. A model like this, once trained on our domain images, will inherently highlight parts that don't fit the learned context (i.e. defects).

- *Context Prediction:* Train an autoencoder or CNN to predict a missing quadrant of the image or the correct ordering of shuffled patches (jigsaw puzzle task). This again teaches the network general knowledge of what a normal fiber end-face looks like.

- **Foundation Segmentation Models:** We will also explore using the **Segment Anything Model (SAM)** from Meta AI as a starting point for segmentation. SAM is trained on a massive dataset for general object segmentation and can generate masks given prompts. We could use SAM in a zero-shot manner to get coarse masks of the fiber core or obvious defects by prompting with points on those regions (which can be automated by brightness/contrast heuristics). Those masks can either serve as initial labels to fine-tune our segmentation model or even be integrated into an interactive workflow for verification. However, SAM might require careful prompt design for microscopic fiber images, so it will be a supplementary aid rather than the core solution.

- **Domain-Specific Pretraining:** If any related dataset exists (e.g. *MVTec AD* for industrial anomaly detection, which includes surfaces with scratches), we will leverage models or weights from that domain. For instance, models like PaDiM or PatchCore (which use pre-trained WideResNet features for anomalies) have been released for MVTec AD – those could be adapted to our fiber data by feeding our images through and seeing if the feature distributions differentiate defects [12]. Another example is a published model for optical surface scratch segmentation [13], which might be fine-tuned to our case.

By using these pre-trained and self-supervised approaches, the new system can **avoid training from scratch**, which is crucial given no ground-truth labels. Instead, the models start with strong generic vision capabilities and then **adapt to fiber optics via unsupervised learning** on the provided images.

3. Training Strategies Without Labeled Data

Designing an effective training strategy is key given the lack of labeled defect examples. We propose a combination of **unsupervised, semi-supervised, and synthetic data techniques**:

- **Unsupervised Anomaly Training:** For the autoencoder-based anomaly detector, we will collect a set of images that are **assumed to be defect-free** (perhaps by manual screening or using the current system to filter out obvious bad ones). This will serve as the training set for the autoencoder. The model will be optimized to minimize reconstruction error on these normal images. As demonstrated by multiple studies, such reconstruction models can then detect defects at test time by producing

high errors on anomalous regions [10] [8]. We will train the autoencoder at multiple scales (e.g. using image pyramids or multi-scale patches) to ensure anomalies of different sizes are caught [12]. No manual labels are needed; the absence of defects in training data is the only requirement.

- **Synthetic Data Generation:** To compensate for zero real labels, we will generate **synthetic training data** for supervised tasks. One approach is to simulate fiber end-face images with defects: start with a clean fiber image (or a procedural rendering of a fiber with known geometry), then overlay synthetic anomalies (scratches, dust specks, pit marks) using graphics or augmentation. These synthetic defects can be simple shapes or noise patterns inserted at random locations. We will produce corresponding ground-truth masks for these synthetic anomalies (since we know where we added them). This technique allows training a segmentation network to explicitly recognize defect regions. A *cut-and-paste* self-supervision method will be used: e.g., take a real clean image, cut a small patch from another region or another image, and paste it onto the fiber surface to mimic a foreign particle or scratch – label that patch as a defect [14]. This **CutPaste approach** (as introduced by Li *et al.* in 2021) creates realistic training pairs without any manual labeling. Another sophisticated method is **DRÆM** (Discriminative Reconstruction Anomaly Embedding): it trains a network on images with *automatically generated anomalies*, tasking the model to both reconstruct the image back to normal and segment out the anomaly [15]. We will use a simplified version of this: inject random artifacts on normal images and train the model to remove the artifact (reconstruct the clean image) and concurrently predict the artifact mask. This dual loss ensures the model learns a robust embedding for anomalies [16] [15]. Synthetic training will be continually refined to mimic real defect characteristics (using domain knowledge of typical scratch shapes, contamination blobs, etc.).

- **Semi-Supervised Refinement:** If even a small number of real defect images are available (or can be identified through the unsupervised model's initial run), we will leverage them in a semi-supervised loop. For example, we could apply the trained autoencoder to the unlabeled dataset to flag potential defect regions, then have an operator confirm a handful of true defects. These can serve as **pseudo-labels** to fine-tune the segmentation network (i.e. treat the autoencoder's high-error regions as defect masks for training, with human verification). We can also employ a **mean teacher** paradigm for segmentation: train a model on the synthetic data, use it to predict masks on real unlabeled images, and gradually incorporate those predictions as training data if the model is confident. Consistency regularization can enforce that augmentations of an image result in the same defect mask output, improving reliability without labels. Additionally, **active learning** could be used in deployment: the system flags uncertain cases (e.g., anomaly score around the decision threshold) and, if the user provides feedback on those, the model can be periodically retrained or fine-tuned, effectively creating a growing labeled set over time.

- **Parameter Tuning with Proxy Metrics:** Since we cannot fully supervise with ground truth, we will use proxy measurements to guide training and model selection. For example, when training on synthetic anomalies, we will monitor the segmentation accuracy on a hold-out set of synthetic images (where ground truth is known) to choose the best model. We will also monitor the reconstruction error distribution on known-good vs known-bad images: a well-trained anomaly model should yield a clear gap (higher errors on known defective images). If available, we might use a small subset where defects are labeled (even if not pixel-perfect, perhaps image-level labels) to compute an approximate ROC curve for anomaly detection and tune the threshold for triggering a defect alert.

In summary, the training strategy is **heavily unsupervised**, bolstered by **synthetic data** to inject knowledge of what defects look like, and possibly **light human-in-the-loop** adjustments. This ensures the AI modules learn effectively despite no explicit labeled dataset.

4. Replacing and Augmenting Pipeline Modules with AI Components

The table below outlines how each region of the current codebase will be replaced or augmented by the new AI-driven components, while remaining modular and interoperable with the rest of the system:

Current Module (Stage)	Existing Functionality	AI-Enhanced Replacement	Implementation Details
Stage 1: Image Processing (`process.py`)	Generates multiple filtered versions of the input (e.g. contrast enhancement, noise reduction) for downstream analysis [1].	*AI Augmented Preprocessing* (optional) – Incorporate a deep learning based enhancer (if needed).	*Implementation:* In most cases, the current augmentation (rotations, illumination variations) can remain. Optionally, a **preprocessing CNN** (like a denoising autoencoder) could be trained to normalize lighting or remove noise, making segmentation easier. This network would be applied to each input image (or variation) before segmentation. It will be modular (one function call), so it can be toggled on/off.

Current Module (Stage)	Existing Functionality	AI-Enhanced Replacement	Implementation Details
Stage 2: Segmentation (`separation.py` with `UnifiedSegmentationSystem`)	Segments fiber into core, cladding, ferrule using 11 algorithm consensus (intensity thresholding, edge detection, Hough circle, etc.) [17]. Produces binary masks for each region, requiring min_agreement_ratio among methods [18] [19].	*Deep Segmentation Module* – Replace multi-method logic with a **unified CNN model** that outputs region masks (core, cladding, background) and possibly defect mask in one pass.	*Implementation:* A **U-Net model** (with attention enhancements) will be loaded in this stage. It takes a single image and outputs a segmentation map with multiple classes (e.g., background, core, cladding, ferrule, defect). The code will map these to region masks similar to existing output format. If needed, two models will be used: one trained for region segmentation (core/cladding) and one for defect segmentation. These models will be encapsulated in a class (e.g., `AI_Segmenter`) with a method `segment_image(image)` that returns masks. The pipeline will call this instead of running all traditional algorithms. The **min_agreement_ratio** logic and redundant algorithms will be removed – instead, we might keep one fallback (e.g., if AI fails to find a core due to some extreme case, the old geometric method could be a backup). However, the goal is full replacement. The deep model's output can be post-processed slightly (morphological cleanup) to ensure it matches expected format (e.g., fill small holes in masks).

Stage 3: Defect Detection (`detection.py` and knowledge_base rules)	Analyzes the segmented regions to detect defects like scratches or contamination. Uses size thresholds (min/max area), intensity features, and a knowledge base of patterns to decide if a region is a defect [20] [19]. Outputs defect coordinates or masks with a confidence score.	*AI Anomaly Detector* – Replace rule-based detection with a **deep anomaly detection model** that produces an **anomaly heatmap and defect mask** for each image.	*Implementation:* We will integrate the trained **autoencoder/anomaly model** here. For each input image (or focused on the fiber region provided by Stage 2), the model will output either: (a) a heatmap of anomaly scores per pixel, or (b) a list of detected defect regions with scores. In practice, the autoencoder will reconstruct the image; we then compute the difference (e.g., mean squared error per pixel or structural dissimilarity) between input and reconstruction to obtain an anomaly map [21]. This map can be thresholded to get binary defect areas. We will replace the `detection_knowledge_base` logic with a learned threshold: e.g., determine a reconstruction error threshold corresponding to an acceptable defect size. The config parameters like **min_defect_area_px** will still be used to ignore tiny noisy blobs [20], but now those blobs are identified by the model rather than simple intensity. We may still use **blob detection** on the anomaly map to aggregate connected defect pixels into a bounding box or contour, similar to the current clustering step [21]. Each defect will be assigned an anomaly score (e.g., the peak error within that region). The output format will match the existing "defect report" structure (location, size, confidence). This AI detector will handle all defect types uniformly – whereas the old system had to specify patterns, the new model detects *any*

Current Module (Stage)	Existing Functionality	AI-Enhanced Replacement	Implementation Details
			deviation from normal, offering broader coverage [22] .

Stage 4: Anomaly Analysis (`data_acquisition.py`, a.k.a. OmniFiberAnalyzer)	Aggregates defects, performs clustering/ deduplication of close defects, classifies defect types, and decides pass/fail based on counts/sizes. Also produces visualization and reports [23] [19].	*AI-Driven Analysis & Classification* – Enhance the final analysis with learned insights: use the model outputs to classify defect types (if needed) and confidence, and produce explainable metrics.	*Implementation:* The clustering of nearby defects can remain (or even be simplified if the autoencoder already distinguishes separate blobs). We will augment this stage by introducing a simple **defect classification model** if defect type labeling becomes available (for example, a small CNN that takes an extracted defect patch and classifies it as scratch vs contamination – this could be trained on synthetic labeled defect patches). If no labels for type, we can use an unsupervised clustering on the features of the defect region (features extracted from the segmentation or anomaly model) to group similar defect patterns – effectively an AI-driven clustering instead of manual thresholds. The **pass/ fail criteria** can be updated to use the anomaly model's confidence: e.g., require no detected defect above a certain anomaly score for a "pass". This threshold can be calibrated by analyzing a batch of known good vs bad samples (if available) or set conservatively initially. The output reports will include the defect masks/ overlays generated by the AI modules for transparency. We will ensure the `data_acquisition` stage can log additional info like anomaly scores, and still produce the final JSON or database entry in the same format as before (for compatibility with any external systems). Visualization code will overlay the segmentation mask and defect heatmap on the

Current Module (Stage)	Existing Functionality	AI-Enhanced Replacement	Implementation Details
			original image for each defect, replacing the previous highlights drawn by classical methods. This provides an intuitive view of what the AI found, increasing trust in the system.

Each new component will be encapsulated to **minimize disruption** to the pipeline's structure. For example, the `AI_Segmenter` and `AI_AnomalyDetector` classes can be configured via the same `config.json` (we will add relevant parameters such as model file paths, thresholds, etc., under `separation_settings` and `detection_settings`). This way, the main orchestrator (`app.py`) still runs Stage 1 → 2 → 3 → 4 in sequence, but internals of Stage 2 and 3 are swapped out for our AI modules. We will maintain the ability to run these stages independently for testing (e.g., one could call the segmentation module on an image from a Jupyter notebook to visualize the mask), preserving modularity.

5. Performance Monitoring, Visualization & Explainability

Introducing AI warrants strong monitoring and explainability to ensure the system is reliable and its decisions are understandable:

- **Performance Monitoring:** We will implement continuous performance logging for the AI components. Key metrics include: reconstruction error statistics (mean, max per image), number of defects detected per image, size of each defect mask, and the model's confidence scores. By logging these across batches, we can detect if the model's behavior drifts (e.g., suddenly flagging too many defects, which could indicate a needed recalibration or model update). We will also periodically evaluate the system on a small validation set. If no labeled set exists, we can use a *pseudo-validation* approach: use the synthetic defect images as a benchmark to ensure the segmentation model is performing as expected (reporting high IoU for synthetic defect segmentation and near-zero false positives on synthetic-clean images). The pipeline's existing output already includes a **quality scoring and pass/fail determination** [23] – we will compare the AI's decisions to historical decisions to ensure consistency or justified improvements. Any big discrepancies can be flagged for review.

- **Visualization Tools:** The new pipeline will enhance visualization by overlaying AI results on images. For each processed image, we will save: the raw image with the **segmentation boundaries** drawn (core/cladding outline identified by the network), and another copy with **defect highlights**. Defect highlights could be the contour of the defect mask or a heatmap coloration showing anomaly intensity. For example, a small scratch might be shown in red on the fiber surface. These visual outputs will be integrated into the reporting module (e.g., saved as PNGs in the results folder, as currently done). In interactive mode, the GUI can display these overlays in real time, allowing the user to see exactly what the AI identified. Since the question specifically asks for explainability, we will also produce **explanation heatmaps** for model decisions: for a given defect detection, we can provide the reconstruction difference image that shows the defective region clearly (the difference image *is* a form of explanation – it shows what part of the image the autoencoder couldn't rebuild,

hence highlighting the anomaly). We might also use **Grad-CAM** or similar techniques on the segmentation network if we implement a classifier (e.g., to explain a "scratch" classification by showing which pixels influenced that class).

- **Explainability Methods:**

- *Saliency Maps:* For the segmentation model, saliency or Grad-CAM can be applied to visualize which parts of the image contributed to classifying a pixel as defect. This could help validate that the network is looking at, say, the actual scratch and not a spurious reflection.
- *Attention Maps:* If using transformers or attention modules, we can extract attention weights to show what regions the model focused on. For instance, an attention map from the last layer might show a spotlight on the defect region [24]. This can be overlaid on the image for interpretability.
- *Reconstruction Error Heatmaps:* As mentioned, the autoencoder's difference image is a direct explainability tool – it intuitively shows "here's where the image looks abnormal". We will ensure this heatmap is scaled and maybe binarized for clarity in reports.
- *Anomaly Score Distribution:* We will visualize the distribution of anomaly scores for each image (perhaps as a histogram or a simple text summary: e.g., "maximum anomaly score = X at location (x,y)"). This gives users a sense of defect severity. In batch processing, we can chart these scores to identify outliers (images with unusually high anomaly scores could be worst cases that need attention).

- *Human-Readable Descriptions:* The final module can be extended to output a brief description of findings, e.g., "1 small scratch detected near core at 2 o'clock position, estimated length 50µm." This could be derived from the mask and known pixel-to-micron scale. While not explicitly asked, such descriptions enhance explainability for end-users not looking directly at images.

- **Performance Optimization:** In terms of speed and resource usage (part of performance monitoring), we will benchmark the new system vs the old. The classical pipeline likely runs on CPU with multiple image variations; the deep models will leverage GPU if available for acceleration. We will measure average processing time per image in batch mode and ensure it meets requirements (the model can be optimized via techniques like model quantization or using OpenVINO/TensorRT if needed). If the AI model is too slow in generating multiple variations, we can reduce the number of image variations needed – since the model itself should handle variations robustly, the `num_variations` could potentially be lowered from 5 to 1 or 2, compensating for any increase in per-image processing time.

In essence, the upgraded pipeline will **monitor itself** and provide rich insight into its operation. By visualizing what the AI sees as a defect and logging detailed metrics, we build trust in the system. We will also implement alerts in monitoring: e.g., if the anomaly detector starts flagging a very high percentage of images as defective, it may indicate either a genuine issue (like a batch of bad connectors) or a model drift – either way, an engineer can be notified to review the situation.

6. Modular Design and Modes of Operation

Maintaining modularity is a core design principle of this upgrade. Each AI component will be designed as a **self-contained module** with a clear interface, so the system remains flexible and debuggable:

- **Module Encapsulation:** The segmentation CNN and anomaly detection model will be encapsulated in classes or functions that mirror the behavior of the original modules. For example, `separation.py` can offer a function `segment_regions(image)` that internally loads the model (if not already loaded) and returns region masks (core, cladding, etc.). Similarly, `detection.py` can provide `detect_defects(image, region_masks)` which runs the autoencoder and returns a list of defects. This way, higher-level orchestration doesn't need to change drastically – it just calls these functions. We will ensure that all parameters (thresholds, model file paths, etc.) are configurable via the same `config.json` so that the pipeline can be tuned without code changes.

- **Interactive vs Batch Modes:** In **interactive mode**, the models will be loaded once at the start of the application and kept in memory (perhaps as global singleton instances or within the `app.py` context). Then each image the user selects is processed by the already-loaded models, providing near real-time results. We will optimize for quick inference: e.g., use GPU if available, use batch size of 1 (for interactive) to minimize latency, and perhaps use half-precision FP16 to speed up computations. In **batch mode**, where the user processes a folder of images, we can take advantage of batch processing by feeding multiple images through the GPU at once if memory allows. For instance, we could process images in batches of N through the U-Net to fully utilize the GPU. The code will detect the mode and adapt accordingly. Also, in batch mode, resource management is important – we will add options in config to adjust batch size or turn off interactive visualizations to save memory.

- **Modularity for Maintenance:** Because the AI components are modular, one can independently upgrade or replace them. For example, if a new better segmentation model is developed later, it can be swapped in by updating the `AI_Segmenter` class without affecting the detection module. The decoupling between segmentation and detection is maintained (the detection mostly uses the original image and perhaps the region mask, but it does not heavily depend on how the mask was produced internally). This is similar to the original design where stages were separated – we preserve that architecture, just with learned modules inside [25].

- **Integration with Existing Code:** We plan to *augment rather than entirely remove* certain functionalities to ensure robustness. For instance, we might keep the old fiber-finding algorithm as a **fallback**: if the AI segmentation fails to find a fiber (e.g., model confidence for core is below a threshold), the system can revert to using the classical Hough circle approach from `hough_separation.py`. This kind of hybrid approach guarantees that in edge cases the pipeline still produces an output. These fallbacks will be modular checks that can be toggled. Over time, as the AI's reliability is proven, the fallback may rarely trigger or can be phased out.

- **Logging and Debug Modes:** Each module will have a debug mode to output intermediate results (e.g., the raw anomaly heatmap, or the intermediate feature maps if needed) to help developers diagnose issues. This will assist in maintaining the system and fine-tuning it.

- **Scalability:** The new design remains scalable: new defect types or new environmental conditions can be handled by retraining or fine-tuning the models without rewriting algorithm code. The modular design allows training offline and just updating model weight files in the deployment. In an R&D setting, one could even run multiple segmentation models in parallel (similar to the original multi-method idea) if desired and fuse their outputs, but with deep models – however, we anticipate one well-trained model is sufficient.

In conclusion, the proposed AI upgrade will **seamlessly integrate** into the existing fiber optic inspection pipeline, replacing manual algorithms with learning-based components that improve with data. We have designed the solution to honor the constraints (no labeled data, need for robustness) by using unsupervised and self-supervised techniques, and to deliver **higher accuracy** and **reliability**. The modular architecture, coupled with thorough monitoring and explainability, ensures that the system can be trusted in both batch processing of large datasets and interactive use by technicians. This upgrade will transform the pipeline into a smarter, more automated system that keeps pace with state-of-the-art AI in computer vision [26] [27], ultimately leading to better defect detection outcomes and easier maintenance of the fiber optic quality standards.

Sources: The design draws upon recent research in industrial defect detection, including segmentation networks with attention for small flaw detection [6] [7], and unsupervised anomaly localization using autoencoders [28] [12]. Studies have shown that even without labeled data, models can be trained on normal samples to accurately spot anomalies [10] [8]. Techniques like synthetic anomaly generation (e.g., DRAEM) enable training robust detectors with no real defect samples [15]. These insights have informed the proposed architecture and training strategy. All modifications will be empirically tested to ensure they meet or exceed the performance of the original system, with an emphasis on reducing false negatives (missed defects) while controlling false positives [3] [4]. The end result will be a cutting-edge, AI-driven fiber optic inspection pipeline that is accurate, fast, and maintainable.

---

[1] [2] [17] [18] [19] [20] [23] [25] readme.md
file://file-JFjGA3eYfJkXcTssW5evKB

[3] [4] data-pixel.com
https://www.data-pixel.com/wp-content/uploads/2021/06/White-Paper-AI-APPLIED-TO-FIBER-OPTIC-METROLOGY.pdf

[5] [6] [7] [10] [12] [24] [26] [27] AnomalySeg: Deep Learning-Based Fast Anomaly Segmentation Approach for Surface Defect Detection
https://www.mdpi.com/2079-9292/13/2/284

[8] [21] [22] [28] [2307.07893] Anomaly Detection in Automated Fibre Placement: Learning with Data Limitations
https://arxiv.org/abs/2307.07893

[9] [14] [15] [16] DRAEM - A Discriminatively Trained Reconstruction Embedding for Surface Anomaly Detection
https://openaccess.thecvf.com/content/ICCV2021/papers/Zavrtanik_DRAEM_-_A_Discriminatively_Trained_Reconstruction_Embedding_for_Surface_Anomaly_ICCV_2021_paper.pdf

[11]
https://arxiv.org/pdf/2207.10298

13 Surface weak scratch detection for optical elements based on a ...
https://opg.optica.org/josaa/abstract.cfm?uri=josaa-40-6-1237