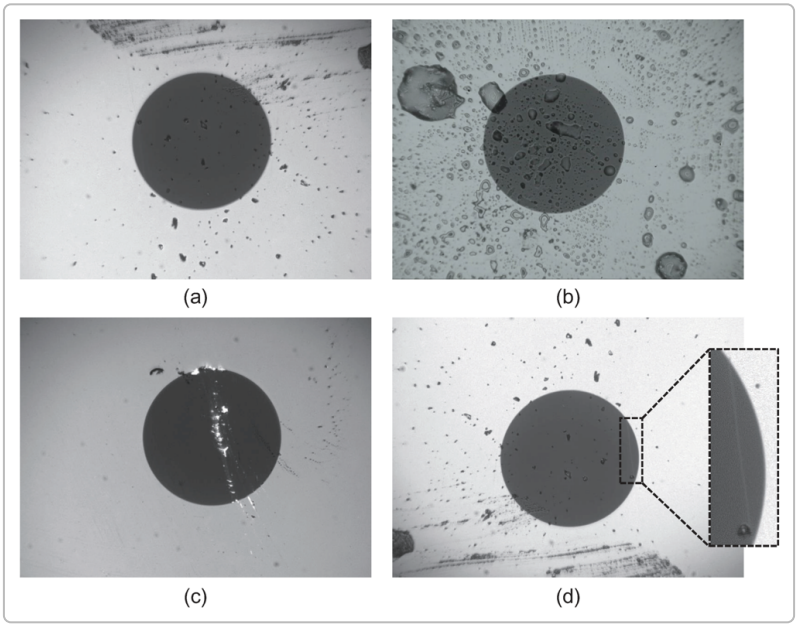
ChatGPT

Enhancing Fiber Optic Defect Detection and Analysis System

Introduction

Fiber optic connector end-face inspection is crucial for maintaining high-bandwidth, low-loss optical networks [1] [2]. Even microscopic defects or contamination on a fiber core, cladding, or ferrule can lead to signal attenuation or back-reflection, degrading performance [3] [2]. Traditionally, human inspectors used microscopes to judge end-face quality, but this is labor-intensive and inconsistent [4]. The goal is to develop an automated **Python-based pipeline** for real-time segmentation of fiber regions (core, cladding, ferrule) and accurate anomaly (defect) detection, all under CPU-only constraints. This report surveys advanced **academic research and methods** that can bolster such a system, focusing on segmentation accuracy, anomaly classification, execution speed, robustness to noise/lighting, and self-learning capabilities. We highlight peer-reviewed papers, PhD-level works, ready-to-use algorithms, datasets, and key innovations with integration potential. The findings are organized into traditional image processing approaches, modern deep learning techniques, anomaly detection strategies, optimization for CPU real-time performance, and continual learning mechanisms.

Challenges in Fiber End-Face Inspection



Examples of fiber connector end-face defects: (a, b) contamination (dust/oil particles across core and ferrule), (c) a prominent scratch across the core, and (d) a faint hairline scratch (magnified in inset) [5].

Automated inspection faces several challenges. **Variety of Defects:** End-face defects range from dust, oil smudges, pits/chips in the glass, to faint scratches [2]. These vary widely in size, shape, and contrast (e.g. tiny spots vs. long thin scratches). **Low Contrast Features:** Some critical defects like fine scratches have extremely low contrast (differing by only a few gray levels from background [6] [7]), making them easy to miss. **Illumination and Noise:** Uneven lighting or glare can obscure defects, and noise from the imaging system can create false textures [8]. **Region Segmentation:** The fiber end-face image is typically circular (the ferrule and fiber core appear as concentric circles). Accurately segmenting the core and cladding regions is essential because inspection standards (e.g. IEC 61300-3-35) define different acceptance criteria for defects depending on which zone (core, cladding, adhesive ring, or outer ferrule contact zone) they lie in [9] [10]. **Real-Time & Resource Limits:** For practical use (e.g. in field test devices or in a VS Code/Pylance environment), the solution must analyze high-resolution microscope images quickly on CPU only. This requires highly efficient algorithms. **Continuous Improvement:** Finally, defect types or environmental conditions may evolve; a desirable system should learn from new data (e.g. new defect examples) without extensive re-training – enabling **continual learning** or self-optimization.

The following sections explore solutions to these challenges, from classical machine vision techniques to cutting-edge deep learning models, and how to integrate them into a CPU-bound Python pipeline.

Traditional Image Processing Techniques

Early and computationally efficient methods for fiber end-face inspection rely on classical image processing and morphology. A representative work by Mei *et al.* (2018) proposed a complete automated inspection pipeline using **morphological filtering and thresholding**, without deep learning [11] [12]. Their system first performs **autofocus** and then robust **circle detection** to locate the fiber center and cladding boundary. Notably, they use Hough transform-based circle finding to determine the fiber's concentric zones [13] [14]. According to the IEC 61300-3-35 standard, the image is divided into the core zone, cladding zone, adhesive (epoxy) zone, and outer contact zone by concentric circles (as shown by the blue circles in their Figure 5) [9] [10]. This zone segmentation is crucial for applying different defect criteria per region. In their workflow, once zones are established, two specialized detection algorithms handle different defect types [15] [16]:

- **DO2MR (Difference of Min-Max Ranking filter)** for **regional defects** like dirt, oil, pits and chips [15] [12]. DO2MR is a local statistical filter that highlights anomalous pixels by computing the difference between local max and min intensities in a neighborhood [17] [18]. This effectively amplifies isolated bright/dark spots against smoother background [19] [20], making contamination stand out despite uneven illumination. After filtering, a sigma-based adaptive threshold is applied to segment candidate defect regions [21] [22], followed by morphological opening to remove noise [22].

- **LEI (Linear Enhancement Inspector)** for **scratch detection** [15] [23]. Scratches are low-contrast, elongated defects that standard filters struggle with [24]. LEI uses oriented linear structuring elements to enhance line features: essentially a bank of line detectors at multiple angles that accumulate subtle linear contrasts [25]. By scanning the image with these oriented filters, faint scratches become amplified (their small pixel-wise intensity differences add up along the line) [26]. The result is a set of "response maps," one per orientation, which are then binarized and aggregated to form a complete scratch mask [27] [28].

These classical algorithms are computationally lightweight (just filtering and basic arithmetic), making them feasible for real-time CPU use. Despite their simplicity, the results were quite strong: Mei *et al.* reported

96.0% image-level defect detection accuracy for regional defects and **89.3%** for scratch detection [29]. This demonstrates that well-designed filtering can achieve high precision. Moreover, the **entire pipeline is transparent and explainable** – each defect must pass certain morphological criteria, and the IEC standard's thresholds are explicitly applied for grading [30] [31]. Such transparency is valuable for user trust and regulatory compliance.

However, classical methods also have limitations: they may require careful parameter tuning (filter sizes, thresholds) for different image conditions, and extremely challenging cases (e.g. a scratch crossing a speckled background, or contamination that mimics fiber texture) might confound them [32]. Nonetheless, these techniques form a strong foundation. They can be implemented with Python libraries like **OpenCV** or **scikit-image** – which provide efficient functions for filtering, Hough circle detection, thresholding, and morphology (all in optimized C/C++ under the hood for CPU performance).

Example – Zone Segmentation and Defect Rules: Using OpenCV's Hough Circle Transform to find the fiber outline, one can delineate core vs cladding regions and then apply zone-specific defect criteria. The IEC standard provides a table of allowable scratch lengths or particle sizes per zone (for instance, any scratch in the core zone may fail, while a small scratch in the outer zone might be acceptable) [9] [30]. By computing features of each segmented defect (size, area, length) [33] [30], the system can classify defect severity. Mei *et al.* use handcrafted features (geometry, area, entropy, etc.) for each defect region and then automatically grade the connector against IEC limits [34] [35]. This rule-based analysis achieved quantitative consistency where human judgment would vary.

Example – Hybrid Classical+ML Approach: Fernandez *et al.* (2021) took a slightly different approach by integrating classical processing with a simple neural network classifier [36]. They built a **portable inspection device** comprising an optical microscope, a Raspberry-Pi-class computer, and a Python algorithm pipeline [37]. Their pipeline: (1) histogram equalization to normalize brightness, (2) Gaussian smoothing to reduce noise, (3) Canny edge detection to highlight scratches, (4) Hough transform to locate the fiber's circular outline (ROI), and (5) extract radiometric features from the image [36]. These features (essentially statistical descriptors of contamination) are fed into an **artificial neural network (ANN)** that outputs a cleanliness score or pass/fail decision. Despite using a modest neural network (suitable for an embedded CPU), their device's assessments agreed with expert human inspectors with a **Cohen's kappa of 0.926**, indicating highly reliable performance [38]. This demonstrates that even on CPU-constrained hardware, combining classic vision (for feature extraction) with a lightweight machine learning model can yield robust results.

Off-the-shelf libraries: Most of the above traditional techniques are available in open-source form. For example, OpenCV's `HoughCircles` can find the fiber core/cladding circle, and its morphological functions (e.g. `cv2.morphologyEx` with custom kernels) can implement something akin to DO2MR and LEI. Thresholding methods like Otsu's or adaptive threshold are in **scikit-image** and OpenCV. These libraries are highly optimized in C++, enabling real-time processing of high-res images on a CPU. Additionally, the **PyPI ecosystem** has tools like `mahotas` or `scipy.ndimage` which offer advanced morphology and filtering if needed. The key advantage of classical methods is deterministic speed and low resource usage, though maintaining accuracy on complex defects may require clever tuning or multi-step processing.

Advanced Deep Learning Approaches for Segmentation and Detection

In recent years, researchers have applied deep learning to this problem, aiming to boost defect detection accuracy and robustness. Deep neural networks, especially convolutional neural networks (CNNs), excel at learning complex visual patterns and can often detect subtle defects that rule-based methods miss [39] [40]. The challenge, however, is to design models that are **lightweight enough for CPU inference** yet powerful in performance. Several notable approaches stand out:

- **Scratch-Oriented U-Net (Liang *et al.*, 2023):** Recognizing the difficulty of detecting **weak fiber scratches**, Liang and colleagues developed a specialized U-Net-like segmentation network [41]. They also enhanced the imaging process by using **multimodal illumination (multi-wavelength + polarized light)** to better reveal fine scratches in the input images [42]. Their CNN architecture is a *U-shaped encoder-decoder* but optimized for efficiency and precision: it includes *residual encoding modules* to ease gradient flow, *serial-parallel multi-scale fusion* to capture defects of different widths, and *triple-convolution decoder blocks* for refined mask output [41]. This design dramatically improved scratch segmentation accuracy while keeping the model size low. In detection tests, their model **accurately segmented faint scratches** on optical surfaces and even generalized well to an unrelated dataset (building crack images) [43]. Impressively, it achieved better performance than comparable deep models *with significantly fewer parameters* [41] – indicating a lean design suitable for CPU use. This research suggests that a tailored CNN can outperform classical methods for low-contrast defect detection, without being prohibitively heavy. Though the paper doesn't explicitly mention the parameter count, their emphasis on "fewer parameters" implies it's in the low millions at most, which is feasible to run on a modern CPU (especially with optimizations like batch normalization folding, etc.). Integrating a U-Net model in Python can be done via frameworks like **PyTorch or TensorFlow**; one can train offline (possibly on GPU for speed) and then export the model to run on CPU using frameworks like **ONNX Runtime** or **OpenCV DNN**. The authors did not release code publicly (to our knowledge), but their detailed architecture description in the paper could guide implementation [41].

- **Multi-Scale Mixed Kernel CNN with Structural Re-parameterization (Liang *et al.*, 2023)**: Beyond scratches, the same group addressed general **optical surface defect identification** (scratches, pits, bubbles, etc.) with an emphasis on speed and accuracy [44] [45]. They designed a CNN that uses **multi-size convolutional kernels** in the early layers to capture features of varying scales and orientations – including an asymmetric mix of square and line kernels for rotational robustness [46]. During training, the network has a multi-branch topology (to learn diverse features), but thanks to a **structural re-parameterization** technique, the branches are mathematically folded into a single streamlined model for inference [47] [45]. The result is a model that, at runtime, is as efficient as a plain CNN but carries the wisdom of a more complex architecture. The performance is striking: on their test set, this model reached **97.39% classification accuracy** and could process images at **201.76 frames per second** with only **5.23 million parameters** [48] [49]. Such a high throughput (200+ FPS) implies that even on a single CPU core it could potentially achieve real-time (though their figure might be on a high-end CPU or batch mode). The small parameter count is extremely encouraging for deployment on CPU and even edge devices. The concept of *re-parametrization* (as used here and in works like RepVGG) is very practical: one can train a complex model (requiring GPU perhaps), then convert it to an equivalent simpler model for inference – which can be run in Python

with minimal overhead. For integration, one might reimplement their architecture or adopt similar designs available in open-source. Notably, this approach proves that **deep learning can meet real-time requirements**, shattering the notion that CNNs are too slow without a GPU [48].

- **Deep CNN with Feature Pyramid & Attention (Mei *et al.*):** The authors of the earlier morphology paper have also explored deep learning in subsequent works. In one reference, Shuang Mei and colleagues developed a DCNN approach for **weak scratch detection** that uses a *feature pyramid network* with a **criss-cross max pooling (CCMP)** module to capture fine boundary details [50]. The CCMP block is designed to extract precise edge features in all directions, ensuring that even scratches at arbitrary angles are detected clearly [50]. Their model, as described, consists of a typical convolutional feature extractor, a *Boundary Feature Enhancement (BFE)* module incorporating the CCMP for refining edges, and a *Segmentation and Aggregation Unit (SAU)* to produce the final mask [51]. This aligns with modern segmentation designs that use attention or specialized pooling to not miss thin structures. While performance metrics weren't quoted in the snippet, it's cited as a **novel method for fiber end-face scratch detection using deep learning** [50]. This indicates that multiple research groups have independently validated deep CNNs as a superior approach to the toughest defect (scratches) – likely achieving detection of scratches that manual or morphological methods might miss.

- **Object Detection Models (e.g. YOLO, Mask R-CNN):** Apart from custom architectures, researchers have tried applying general-purpose detection networks to industrial defect tasks. For example, Yang *et al.* used a single-stage detector (YOLOv5) for **steel weld defect** detection and found it competitive with two-stage detectors like Faster R-CNN [52]. Similarly, for **fiber connector end-faces**, one could train a YOLO or SSD model to detect and localize defects (treating each contaminant or scratch as an object with a bounding box). YOLO models (especially smaller variants like YOLOv5n or YOLOv8-tiny) can run on CPU at decent speeds, and their accuracy in detecting objects could translate to finding large defects. However, one limitation is that bounding boxes might not precisely outline irregular defect shapes. An alternative is **Mask R-CNN**, which provides segmentation masks for detected regions. A Mask R-CNN with a light backbone (like ResNet-18 or MobileNet) could be an option; indeed, in a related domain, researchers successfully improved Mask R-CNN for small, low-contrast defects in CT images [53] [54]. In our context, a fine-tuned Mask R-CNN could segment defects on the fiber. The downside is that these models are heavier; Mask R-CNN is typically tens of millions of parameters. Yet, some recent works propose **pruned or quantized versions** and even transformer-based detectors optimized for small defects [55] [56]. If real-time performance is slightly relaxed (say, a few FPS is acceptable), using these proven architectures might simplify development because frameworks like **Detectron2** or **MMDetection** can provide ready implementations. The user's pipeline could invoke a pre-trained model (converted to ONNX) via OpenCV's DNN module to get detections on CPU.

Public Code and Models: Many deep learning approaches in academic papers do not come with open-source code for the specific application (fiber optics). However, the building blocks are often available. For instance, the **feature pyramid network, residual modules, and attention mechanisms** used by the above researchers are standard components one can assemble using libraries like Segmentation Models Pytorch or TensorFlow Keras segmentation APIs. The challenge is obtaining enough training data – more on that in the datasets section. If obtaining a labeled dataset is feasible, one could also leverage **transfer learning**: e.g. start with a model pre-trained on a similar task (perhaps a model trained on detecting

scratches or thin objects in another domain) and fine-tune it on fiber images. This can drastically cut training time and data needs.

Performance on CPU: To ensure deep models run fast on CPU, consider techniques like **model quantization** (8-bit inference using frameworks like OpenVINO or PyTorch quantized). Quantization can often 2-4× speed up inference with minimal accuracy loss, and is well-suited if deploying via OpenCV or ONNX. Another trick is **model pruning** – removing redundant channels/filters from the CNN. There is abundant literature on pruning for speed; tools like TensorFlow Model Optimization Toolkit or PyTorch's pruning APIs could be applied to a trained fiber-inspection model. **Knowledge distillation** is yet another approach: train a large accurate model, then train a smaller "student" model to mimic it, achieving nearly the same accuracy with far fewer parameters (this could be useful if one trains a complex Mask R-CNN but then distills its knowledge into a simpler network more apt for CPU deployment).

In summary, deep learning offers state-of-the-art accuracy for segmentation and defect detection, especially for challenging anomalies like scratches or subtle contamination. With careful architecture choices and optimization, it's entirely feasible to run these models in real-time on a CPU. The table below compares some key methods and tools discussed so far:

Method / Tool	Technique & Key Idea	Advantages	Considerations
Hough Circle + IEC Zoning (*Classical CV*)	Use Hough transform to find fiber core/cladding circle; divide image into standard zones (core, cladding, etc.) per IEC 61300-3-35 [14] [9].	Robustly isolates regions for focused analysis; ensures compliance with standard criteria; fast (millisecond-level on CPU).	Heavy contamination or poor contrast can foil circle detection (need robust pre-processing); one-time calibration per connector type (e.g. size) may be needed.
DO2MR & LEI Morphology (Mei et al. 2018)	Difference of min-max filtering for spot defects; directional morphology for scratches [15] [25]. Entirely rule-based.	High precision for various defect types; no training data needed; easily integrates into equipment [29] [16].	Requires tuning of filter window and thresholds; may struggle with extremely low-contrast scratches without fine tuning [24].
Classic CV + ANN (Fernandez et al. 2021)	Extract handcrafted features (histogram, edges, etc.) and feed to a small neural network classifier [36].	Low computational cost; leverages simple ML for improved consistency; proven in portable device (Cohen's $\kappa \approx 0.93$) [38].	Only yields a cleanliness score (no pixel-level mask); limited to training on predefined defect criteria (not easily extensible to new defect types without re-training).

Method / Tool	Technique & Key Idea	Advantages	Considerations
Scratch U-Net (X. Liang et al. 2023)	U-Net with residual and multi-scale modules to segment **faint scratches** [41]. Enhanced imaging (multi-spectral light) used to improve input.	Excellent at detecting hairline scratches; fewer parameters than typical CNN, saving CPU memory [41]. Generalizes to other fine line defects (e.g. cracks) [57].	Needs representative training data (scratches under varied conditions); multi-spectral illumination setup may not be available in all systems (their method benefits from hardware).
Mixed-Kernel CNN (Re-Param) (X. Liang et al. 2023)	Multi-branch CNN with mixed convolution shapes for multi-scale features; structurally re-parametrized into a single branch for inference [58] [45].	**Ultra-fast**: achieved ~201 FPS on test images; high accuracy ~97% on multiple defect classes [48]. Compact model (5.2 M params) suits CPU execution [59].	Implementation is complex – custom training procedure to merge branches; no public code, so significant effort to reproduce. Requires a decent dataset of various defects to reach reported accuracy.
Deep CNN + CCMP (S. Mei et al.)	Deep CNN with feature pyramid and Criss-Cross Max-Pooling for boundary refinement [50]. Tailored for weak scratches on end-face.	Captures subtle, oriented features in all directions; likely improves true positive rate on very fine defects.	Possibly high memory use (feature pyramids); must integrate into existing pipeline (likely as an add-on module for scratch localization). Training data for scratches still needed.
Unsupervised GAN (NOLSAGAN) (Mei et al. 2024)	**No-label anomaly segmentation** using a Generative Adversarial Network with self-attention to reconstruct normal appearance, flagging defects by deviation [60].	Doesn't require defect labels – model learns the normal fiber texture and finds anomalies automatically; can continuously learn from new "good" samples (self-updating) to improve sensitivity.	GANs are challenging to train and tune; requires enough clean images to capture normal variability. Inference might be slower due to generator + attention layers (must be optimized for CPU). Limited info on real-time performance.

Method / Tool	Technique & Key Idea	Advantages	Considerations
YOLO/tiny Detectors (General, e.g. YOLOv5)	Train object detection model to detect defects with bounding boxes (optionally segment via Mask R-CNN).	Leverages well-maintained architectures; many pre-trained weights exist to fine-tune; can detect and classify multiple defect types at once. Real-time possible with smaller models on CPU.	Bounding boxes might not precisely delineate defect extents; needs hundreds of annotated examples for training each defect class. Mask R-CNN gives masks but is heavier. Quantization or smaller backbones needed for speed.

Anomaly Detection and Self-Learning Strategies

Beyond supervised learning of known defect types, an important capability is to detect **novel or rare anomalies** without explicit labels – and to adapt over time as new data comes in. In fiber inspection, this could mean recognizing an unusual defect (e.g. a burn mark or a broken fiber tip) that wasn't in the original training set. Two research directions are relevant here: **unsupervised anomaly detection** and **continual (incremental) learning**.

Unsupervised Anomaly Detection: These methods learn a model of normal (defect-free) fiber end-face images and then flag deviations. A state-of-the-art example is the NOLSAGAN framework by Mei *et al.* (2024) [60]. NOLSAGAN stands for *NO-Label Self-Attention GAN* – it employs a Generative Adversarial Network to perform **feature reconstruction** of a normal image, using self-attention layers to better model the image's important regions (like the fiber core pattern) [60]. During inference, the GAN tries to reconstruct a given image; any defects present will not be perfectly recreated (since the model only knows how to make "good" end-faces), and thus the difference (input minus reconstruction) highlights the anomalies. Essentially, this yields an unsupervised segmentation of defects. Such an approach is very attractive for a Python pipeline: one could train the GAN on a collection of clean end-face images (which are easier to obtain than defect-labeled images) and then detect anomalies on the fly by thresholding the reconstruction error. The **self-attention** component helps the model focus on critical regions (e.g., ensuring the core region is modeled in high fidelity, since scratches there are crucial). While the full details of NOLSAGAN are in an IEEE Transactions paper, the concept aligns with other unsupervised anomaly methods in literature, such as Autoencoders or Variational Autoencoders trained on normal data. The *training* of such GAN/AEs can be time-consuming, but once trained, inference can be optimized for CPU (especially if using a smaller network). There are open-source libraries like **Anomalib** (by IBM) which implement various anomaly detection models (GAN-based, Knowledge Distillation-based, etc.) that could potentially be adapted to fiber images.

Another unsupervised approach could be classical: for instance, building a **"reference model"** of the clean fiber appearance using feature descriptors. The code snippet from the user's `OmniFiberAnalyzer` suggests a strategy: it keeps a `reference_model` with features and statistical parameters (mean, covariance) of known-good images, then flags new images whose features deviate beyond learned thresholds. This is essentially an **outlier detection** approach using classical ML (one could use PCA or one-class SVM on handcrafted features like intensity histograms, texture metrics, etc.). While simpler than deep

learning, this approach can be effective if the feature set is well-chosen. It's easier to update incrementally as well – each new "good" image can update the running mean/covariance of features, allowing the model to slowly adapt to changes in imaging or fiber types.

Continual and Self-Learning: Continual learning refers to updating the model with new data without retraining from scratch, ideally without forgetting old knowledge. In the context of fiber inspection, this could mean periodically retraining or fine-tuning the defect detection model as new defect examples are confirmed (especially if new defect modes appear in the field), or as the imaging conditions change. Modern research has started addressing this for anomaly detection; for example, an *Unsupervised Continual Anomaly Detection (UCAD)* framework was proposed to incrementally update an anomaly detector with new normal data streams [61]. The method by Fahim *et al.* (2023) uses contrastive learning to avoid catastrophic forgetting while adding new data to the model's knowledge [61]. While not specific to fiber, the principle can be borrowed: one can design the pipeline to periodically incorporate new "trusted clean" images into the reference model or re-train the anomaly segmentation network on an expanded dataset.

Another scenario is **active learning** – the system could flag borderline cases and ask for a human label ("is this defect significant or not?"), then use that feedback to improve. For instance, if the pipeline isn't sure whether a particular pattern is a scratch or just a polishing mark, it could log it for review. With a human-in-the-loop providing the ground truth, the system's classifier could be updated (since many frameworks allow continued training on new examples). Libraries like scikit-learn or PyTorch make it straightforward to update models or at least do batch retraining with combined old+new data.

Practical implementation: The user's pipeline already hints at a *knowledge base* (`fiber_anomaly_kb.json`) and a reference model that can learn thresholds. This is a solid design for continual learning in a unsupervised manner. To bolster it with academic insights, one could incorporate some of the following: (a) use **feature embedding from a pre-trained CNN** as opposed to raw handcrafted features – e.g., pass images through a pre-trained ResNet (or a smaller network) and use the intermediate feature vector as the representation of the fiber end-face. Then perform outlier detection on those feature vectors. Pre-trained CNN features often capture more nuance than manual features. (b) Use a **one-class classifier** like an Isolation Forest or One-Class SVM on those features. These algorithms, available in `scikit-learn` and `PyOD` library, excel at unsupervised anomaly detection by learning the boundary of normal data in feature space. They are relatively fast on CPU for moderate feature dimensions and dataset sizes. (c) Periodically retrain the one-class model as new normal samples accumulate (but use techniques like storing a fixed-size buffer to avoid unbounded growth, or using partial fit if supported).

In summary, enabling the system to **self-learn from new data** will increase its longevity and robustness. Many of the cited works support this: the Data-Pixel white paper notes that with deep learning, the model can be retrained on new defect types beyond the original standard (e.g. to detect "unpolished fiber" or "broken fiber" if those weren't in the IEC spec) [39]. This only requires providing examples of the new condition and updating the model, which is far simpler than trying to hard-code a new rule [62] [40]. The implication is that **flexible learning systems outperform static rule-based systems** in the long run, as they can adapt to evolving quality criteria or unforeseen defect modes.

Datasets and Benchmarks

One bottleneck in academic progress for this domain has been the lack of public datasets. As noted by Mei *et al.* (2018), *"as far as we know, there is no public dataset about optical fiber end face [images] available for*

study so far.” [63] . Each research group has had to collect their own images, often under proprietary or specific conditions. For example, the 2018 Sensors paper used 320 sample images (80 of each type of common defect) captured with their lab setup [64] . Likewise, Liang *et al.* (2023) likely built an experimental dataset of scratch images under different lighting. This makes direct benchmarking difficult – algorithms are often evaluated on private data or according to IEC criteria rather than a common metric.

However, there are a few resources that could be leveraged: - **IEC 61300-3-35 standard** documentation itself often includes example images of various grades of defects (though not a large dataset, it provides guidance on what is considered a defect in each zone). - Some companies have released small sample sets. For instance, we found an open dataset on Roboflow called “Fiber-Defect-Detection” (by Derik Muñoz) with 56 images of defects [65] . This is a tiny set but could serve as a starting point for testing or augmentation. - **Synthetic data**: Interestingly, a patent publication suggests generating training data by taking clean fiber images and algorithmically overlaying synthetic defects (spots, lines) to create labeled data for training ML models [66] . This approach can produce unlimited variations of defects in a controlled way. While synthetic data might not capture all real-world nuance (e.g. the precise texture of a scratch), it can be useful for pre-training a model which is then fine-tuned on a smaller real dataset. - **Cross-domain transfer**: There are publicly available datasets in related domains, such as the **MVTec AD dataset** (for general surface defect anomaly detection) or datasets of **micro scratches on materials** that could possibly be repurposed. If one trains a model on MVTec “metal screw defect” or “capsule defect” images, the model learns to segment anomalies in a generic sense. Transfer learning from such a model to fiber images could accelerate training.

For benchmarking, each research tends to report metrics like precision, recall, accuracy, etc., on their own test sets. For instance, the morphology method was 96% accurate on their set [29] ; the deep scratch U-Net presumably outperformed others on scratch IoU; the mixed-kernel CNN was 97.4% accurate, etc. If one wanted to evaluate their pipeline, they could follow the **IEC acceptance criteria** as a benchmark: e.g., run the pipeline on a set of fiber images and check how often the pipeline’s pass/fail grading agrees with an expert or with the standard’s judgment. This is effectively what the IJECE 2021 study did (using Cohen’s kappa to measure agreement with human inspectors) [38] .

In absence of a large curated dataset, a practical approach is to curate your own incremental dataset: start with a batch of images (possibly from different connector types, lighting conditions), label them (even if just labeling defects vs no defect, or marking zones of defects), and use that to train or validate the system. Over time, keep adding difficult or misclassified cases to the training set (active learning). After a few iterations, the model will become much more robust.

Tools and Libraries for Integration

Implementing these advanced methods in a Python environment like Pylance/VSCode is achievable with the right libraries:

- **OpenCV**: Already likely in use, OpenCV is indispensable for classical steps (reading images, preprocessing, filtering, edge detection, Hough transform, etc.). It also offers the `cv2.dnn` module which can load and run deep learning models in formats like ONNX, Caffe, TensorFlow. This could allow you to deploy a trained CNN model for defect segmentation within the same pipeline without heavy dependencies.
- **scikit-image & SciPy**: For additional image processing routines (e.g. more advanced threshold algorithms, segmentation algorithms like active contours or random walker, if needed).

- **PyTorch / TensorFlow**: For training deep models. PyTorch in particular is very user-friendly in Python and has a rich ecosystem (e.g. `torchvision` for pre-trained models, `Albumentations` for data augmentation which is important for robustness). Once a model is trained, you can export it. With PyTorch, one can script or trace the model to TorchScript for CPU or convert to ONNX. With TensorFlow, one can export a frozen graph or TFLite model for CPU.
- **ONNX Runtime or OpenVINO**: Both are optimized inference engines for neural networks on CPU. ONNX Runtime can be installed via pip and used to run the model forward pass very efficiently (often faster than pure PyTorch on CPU, due to graph optimizations). OpenVINO is Intel's toolkit that can take an IR or ONNX model and run optimized inference leveraging SSE/AVX vector instructions; it's particularly effective if using Intel CPUs. These frameworks often achieve near-C++ speeds from Python.
- **PyOD**: This is a Python library for **outlier detection** (anomaly detection) that implements a variety of algorithms (Isolation Forest, One-Class SVM, Autoencoders, etc.) in a user-friendly way. It could be handy for quickly trying an anomaly detection approach on feature vectors or even image patches.
- **Anomalib**: Mentioned earlier, for unsupervised anomaly segmentation. It's a bit more specialized and heavier (built on PyTorch), but if unsupervised methods are a priority, it provides implementations of popular algorithms (GANomaly, f-AnoGAN, PatchCore, etc.) along with pre-trained weights on some datasets that could be fine-tuned.
- **segmentation_models.pytorch**: For quickly prototyping segmentation networks (U-Net, FPN, DeepLabV3, etc.) with various backbones. This can accelerate experimentation to find a good architecture that balances accuracy and speed.
- **Logging and Profiling**: Since performance is key, tools like **profilers (cProfile, PyTorch profiler)** and OpenCV's tick meter can help identify bottlenecks in the pipeline. Often, combining steps or using vectorized numpy operations can yield big speed-ups in Python for the classical parts.

Finally, integrating everything in a cohesive pipeline will require careful engineering: using multi-threading or async processing might be beneficial (e.g. capture image in one thread, process in another, so the microscope capture doesn't stall). However, since the question scope is research-focused, the main point is that all the pieces (from advanced algorithms to efficient libraries) are available to assemble.

Conclusion

Real-time fiber optic connector inspection is a challenging but solvable problem with the aid of modern computational methods. By combining **classical image processing** (for reliable geometry detection and initial filtering) with **cutting-edge deep learning** (for high-accuracy segmentation of subtle defects), one can significantly boost both the precision and automation of the inspection. Notably, deep learning approaches have matured to the point that **CPU-only inference is feasible**, thanks to network optimizations and efficient libraries – as evidenced by research prototypes achieving ~200 FPS on CPU [48]. Moreover, incorporating **unsupervised anomaly detection** allows the system to handle unforeseen defects and continuously improve by learning from new data, a feature that rule-based systems lack.

In implementing these advances, it's wise to prioritize methods with **public implementations or easy integration**. For example, one might start by implementing the proven morphological filters (leveraging OpenCV for speed) and then progressively integrate a pre-trained deep model (perhaps a small U-Net or a Mask R-CNN) for the tasks where it excels (like scratch detection). Public datasets are limited, so a parallel effort to build a custom image dataset – even if modest in size – will pay dividends, especially for training and validating machine learning components.

The convergence of academic research and practical tools means your Python-based pipeline can achieve: **improved segmentation accuracy** (ensuring core, cladding, and ferrule regions are correctly identified and analyzed), **enhanced anomaly detection** (fewer missed defects and false alarms, even for low-contrast scratches), **optimized speed** (leveraging CPU-efficient algorithms to meet real-time demands), **robustness to noise and lighting** (via techniques like adaptive filtering, data augmentation, and illumination normalization), and **self-learning capability** (through unsupervised modeling and incremental updates). By following the guidance from the literature – such as leveraging morphology for coarse detection and CNNs for fine segmentation, using attention mechanisms for tiny features, and structurally optimizing models for inference – the pipeline can be markedly enhanced. Each research work cited offers a piece of the puzzle, and together they provide a roadmap to a state-of-the-art fiber end-face inspection system that is both accurate and deployable in the field on CPU platforms.

Sources: The information and innovations discussed are drawn from a range of peer-reviewed articles and studies, including *Sensors* (Mei *et al.* 2018) [15] [29], *IJECE* (Fernandez *et al.* 2021) [36], *J. Opt. Soc. Am. A* (Liang *et al.* 2023) [41] [45], *IEEE T-IM* (Mei *et al.* 2024) [60], as well as industry white papers [67] [40]. These provide validated insights into techniques that can be translated into the Python environment to achieve the desired improvements. Each of the referenced works contributes to an aspect of the problem – from core segmentation to anomaly learning – and their convergence is the key to a comprehensive solution. By implementing and iteratively refining these methods, one can transform the existing fiber optic defect detection pipeline into a faster, smarter, and more reliable system poised for real-world deployment.

---

[1] [2] [3] [4] [5] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] Automated Inspection of Defects in Optical Fiber Connector End Face Using Novel Morphology Approaches
https://www.mdpi.com/1424-8220/18/5/1408

[6] [7] [39] [40] [62] [67] data-pixel.com
https://www.data-pixel.com/wp-content/uploads/2021/06/White-Paper-AI-APPLIED-TO-FIBER-OPTIC-METROLOGY.pdf

[36] [37] [38] Device to evaluate cleanliness of fiber optic connectors using image processing and neural networks | Fernandez | International Journal of Electrical and Computer Engineering (IJECE)
https://ijece.iaescore.com/index.php/IJECE/article/view/24616

[41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [57] [58] [59] Deep Learning Based Automated Inspection of Weak Microscratches in Optical Fiber Connector End Face | Request PDF
https://www.researchgate.net/publication/349467748_Deep_Learning_Based_Automated_Inspection_of_Weak_Microscratches_in_Optical_Fiber_Connector_End_Face

[53] [54] [55] [56] Defect R-CNN: A Novel High-Precision Method for CT Image Defect Detection
https://www.mdpi.com/2076-3417/15/9/4825

[60] dblp: Guojun Wen
https://dblp.org/pid/94/4145.html

[61] Unsupervised Continual Anomaly Detection with Contrastively ...
https://arxiv.org/abs/2401.01010

[63] Automated Inspection of Defects in Optical Fiber Connector End ...
https://pmc.ncbi.nlm.nih.gov/articles/PMC5982614/

64 Automated Inspection of Defects in Optical Fiber Connector End ...
https://www.researchgate.net/publication/324921556_Automated_Inspection_of_Defects_in_Optical_Fiber_Connector_End_Face_Using_Novel_Morphology_Approaches

65 Fiber-Defect-Detection Instance Segmentation Dataset by Derik Muoz
https://universe.roboflow.com/derik-muoz/fiber-defect-detection

66 enhanced optical fiber inspection using image ... - Justia Patents
https://patents.justia.com/patent/20250078238