Baseline Strategy for Vesuvius Challenge Surface Detection

Introduction

The Vesuvius Challenge – Surface Detection task involves segmenting the **recto surfaces** of ancient Herculaneum papyrus scrolls from 3D X-ray CT volumes. These recto surfaces correspond to the outward-facing sides of each papyrus sheet within a rolled scrollkaggle.com. The goal is to produce a binary volumetric segmentation of these sheet surfaces (foreground) against the background interior/air, with **minimal topological errors**. In practice, this means the predicted papyrus sheets should form continuous, unbroken surfaces that neither merge distinct layers nor contain spurious holes. The evaluation metric is a weighted combination of three measures – **SurfaceDice@2.0**, **VOI_score (Variation of Information)**, and **TopoScore** – each ranging from 0 to 1kaggle.com. SurfaceDice@2.0 checks how well the predicted surface lies within a 2-voxel tolerance of the true surface, allowing slight localization error without heavy penaltykaggle.com. VOI_score measures the similarity between the segmentation and ground truth in an information-theoretic sense (lower VOI corresponds to higher score), and TopoScore assesses topological fidelity (penalizing extra/missing connected components or holes). The final competition score is `0.30*TopoScore + 0.35*SurfaceDice + 0.35*VOI_score`kaggle.com, emphasizing a balance of geometric accuracy and correct topology.

To meet these criteria, our baseline strategy leverages **academically validated and competition-proven techniques** from biomedical 3D segmentation, topology-aware learning, and volumetric modeling. We design a robust 3D segmentation pipeline that preserves the continuous sheet-like structure of the papyrus layers. Key considerations include careful **preprocessing** of the CT volumes and labels, choice of a suitable **network architecture** (comparing 2.5D vs. full 3D models, including modern transformer-based networks), specialized **loss functions** to penalize topological errors (e.g. ensuring connectivity), effective **data augmentation** tailored to carbonized scroll data, and thorough **training/inference procedures** (patch-based training, sliding-window inference with test-time augmentation) to handle large volumes within the 9-hour execution limit. We also outline a **validation protocol** aligned with the competition metrics to guide model development. Throughout, we cite relevant papers and benchmarks to justify each component. The overall aim is a **production-ready baseline** that is **robust, reproducible**, and a strong foundation for this leaderboard challenge.

Data Representation and Preprocessing

Data Format: The dataset consists of 3D CT volume **chunks** of scrolls, each with binary labels given as 0 (background), 1 (papyrus surface), or 2 (unlabeled) for each voxel. The chunk sizes can vary, and voxel spacing may differ per volume. These variations require preprocessing to ensure consistency and to optimize model performance.

Voxel Spacing Alignment: We first **resample** all volumes to a common isotropic voxel spacing or a consistent resolution (e.g. using the median spacing across the dataset) so that geometric features like the 2-voxel SurfaceDice tolerance have uniform meaningemergentmind.com. We apply trilinear interpolation for the CT intensity data and nearest-neighbor interpolation for the label masks to avoid smoothing label boundariesemergentmind.com. Standardizing the spacing ensures that a "2 voxel" surface offset corresponds to the same physical distance in all cases, and that the model does not have to learn scale differences between scans. This alignment is an established best-practice in medical imaging pipelines like nnU-Netemergentmind.com.

Intensity Normalization: The X-ray CT scans of carbonized papyrus have a limited contrast – the carbonized papyrus and ink have subtle density differences from background. We perform intensity normalization to reduce variance between scans. Following nnU-Net conventions, we **clip extreme intensity outliers** (e.g. below the 0.5th and above the 99.5th percentile of voxel values) to remove spuriously bright/dark artifacts, then apply **z-score normalization** (subtract the mean and divide by the standard deviation) per volumeemergentmind.com. This standardization brings all inputs to a comparable scale, helping the model focus on meaningful intensity gradients (e.g. the slight edge contrast at the papyrus surface) rather than global brightness differences. In case the CT data is in Hounsfield Units or 16-bit depth, we also convert to 32-bit float and scale into a convenient range (0–1 or standardized) for network input.

Denoising: The scans may contain significant photon noise or ring artifacts due to the high-resolution, low-energy CT imagingnature.comnature.com. While the deep network can learn to denoise to some extent, a light preprocessing denoising step can improve signal-to-noise ratio for thin surface detection. We consider applying a **3D Gaussian filter or non-local means** to smooth high-frequency noise without blurring the papyrus sheets excessively. In related work on CT scroll imaging, researchers balanced X-ray exposure and noise, and post-processed volumes with noise reduction techniques (e.g. averaging multiple projections or TV-regularized reconstruction) to enhance page visibilitynature.com. For our baseline, a mild Gaussian smoothing (e.g. σ=0.5–1 voxel) can be used if the raw slices appear very grainy. This should be done cautiously to not erase the faint contrast of ink or papyrus edges.

Mask and Label Handling: The provided labels include *unlabeled* regions (value 2) which typically indicate areas where ground truth is uncertain or not provided (e.g. boundaries of the scan or damaged regions). We handle these by creating a **mask** that identifies all voxels with label 2, and ensuring the loss function ignores these voxels. In practice, we set the loss weight to 0 for unlabeled voxels or use masked loss formulations. This prevents penalizing the model for arbitrary predictions in unknown regions. During training, these voxels contribute nothing to the loss; during inference, we can choose to automatically set any predictions in those areas to background (0) to be safe, or simply leave them as predicted since they won't count toward metrics. We *do not merge* unlabeled (2) into background (0) for training, as that could incorrectly supervise the model to treat uncertain papyrus as definite background. Instead, we **exclude unlabeled voxels from both loss and metric computation** so that evaluation focuses only on genuinely labeled areas.

Cropping and Chunking: Each volume chunk likely already focuses on a region of interest (a segment of scroll) and may contain a lot of empty background around the scroll. To reduce memory waste and speed up training, we crop each volume to the tightest axis-aligned bounding box that contains all labeled foreground voxelsemergentmind.com. This is feasible since the label mask will highlight where papyrus surfaces are present. Cropping removes uniform air regions and significantly downsizes input dimensions, which is beneficial for training large 3D models. If any volume is extremely large (beyond what fits into GPU memory even as patches), we can further subdivide it into overlapping sub-chunks offline. However, because the competition data is already given as manageable "chunks", we assume basic cropping suffices. We maintain the spatial coordinate mapping so that predictions can be reassembled to original positions if needed.

Summary of Preprocessing: In summary, we **resample** volumes to uniform spacingemergentmind.com, **normalize** intensities (clip and z-score)emergentmind.com, optionally **denoise** to enhance SNR, **mask out unlabeled regions** from training, and **crop** out irrelevant background. These steps standardize the inputs and make the learning task easier, which is validated by robust segmentation frameworks in the literature adopting similar preprocessing for multi-site medical dataemergentmind.comemergentmind.com.

Model Architecture Selection and Trade-offs

Selecting an appropriate network architecture is critical for capturing the thin, sheet-like surfaces in 3D while balancing memory and time constraints. We consider three main approaches: **2.5D CNN**, **full 3D CNN (U-Net/V-Net)**, and **Transformer-based 3D networks (e.g. Swin UNETR)**. Table 1 summarizes their characteristics:

Table 1. Comparison of segmentation architecture approaches.

Approach	Description	Pros	Cons
2.5D CNN	Treat the volume as stacks of 2D slices; train a 2D U-Net on slices with multiple adjacent slices as channels (context).	– Lower memory footprint than 3D. – More training samples (each slice is an example), helping generalization when data is limited. – Simpler implementation (2D convs).	– Lacks full 3D spatial context; continuity between slices is not explicitly learned, risking discontinuities between predicted slices. – Cannot directly model out-of-plane surface curvature.
3D CNN (U-Net / V-Net)	Volumetric encoder-decoder (3D U-Net or V-Net) operating on 3D patches of the volume.	– Captures complete 3D context of surfaces, which aids in preserving continuous layers across slices. – Proven high accuracy in biomedical segmentation	– Much higher GPU memory and compute demand; requires patch-wise training and inference (sliding window). – Fewer training

Approach	Description	Pros	Cons
		tasks[arxiv.org](arxiv.org). – Better at reducing false positives by using context from adjacent slices[frontiersin.org](frontiersin.org).	samples (each patch is 3D), potentially needing heavy data augmentation to avoid overfitting. – Training from scratch can be slower.
Transformer-based 3D (Swin UNETR)	Hybrid model with a 3D Swin Transformer encoder and a CNN decoder (skip connections)[arxiv.org](arxiv.org).	– Models long-range dependencies via self-attention, which can better capture global scroll geometry (e.g. long continuous surfaces wrapping around). – Achieved state-of-the-art segmentation accuracy in medical benchmarks (e.g. BraTS brain tumor challenge)[arxiv.org](arxiv.org).	– Computationally heavy: self-attention in 3D is memory-intensive. – Requires large training data or pretraining to shine; may overfit small datasets. – Slower inference, which could challenge the 9h limit if not carefully optimized.

2.5D Approach: A 2.5D network uses 2D convolutional architectures (like a U-Net) but inputs a stack of adjacent slices (for example, 3 or 5 consecutive slices) and produces a segmentation for the central slice. This gives some 3D context (the network sees a few slices worth of neighbors) without the full cost of 3D conv layers. Such approaches have been popular on Kaggle and in some biomedical studies when GPU memory was a limiting factor. A 2.5D baseline is easier to train and often yields decent segmentation of sheet surfaces on a slice-by-slice basis. In fact, the official Kaggle baseline for this challenge started with a 2D slice model for simplicity. However, the major drawback is that the model might not enforce **cross-slice continuity** well – each slice is segmented independently (with limited neighbor info), so thin surfaces could become fragmented or misaligned between slices. Topologically, this is dangerous: a continuous papyrus layer could appear broken when viewed in 3D if some slices are missed. Additionally, 2.5D models can struggle to aggregate evidence that's distributed across many slices (e.g. a faint surface that only becomes obvious when considering a thicker context).

Empirical studies comparing 2.5D and 3D segmentation show mixed results, often task-dependent. For example, Ottesen et al. (2023) found a 2.5D model had higher sensitivity (fewer misses) in detecting small brain metastases, while a 3D model produced fewer false positives, thanks to better contextual awareness[frontiersin.org](frontiersin.org). This suggests 2.5D might detect more papyrus surface voxels in ambiguous areas (due to focusing on 2D texture), but a 3D model might be more precise, avoiding spurious "surfaces" that are not truly connected to the scroll layer. Given our goal to avoid false mergers and holes, the precision and topological consistency offered by 3D models are compelling advantages.

3D CNN (U-Net / V-Net): A full 3D U-Net architecture is well-suited to segmenting papyrus surfaces as volumetric structures. **U-shaped encoder-decoder networks with skip connections** have become the de facto standard in biomedical segmentationarxiv.org. A 3D U-Net can directly learn the **sheet continuity** in all directions, likely reducing slice-to-slice fragmentation. For instance, Milletari et al.'s V-Net (a 3D CNN) achieved accurate prostate MRI segmentations by processing entire volumes and introduced the **Dice loss** to mitigate class imbalancearxiv.org – demonstrating the efficacy of end-to-end 3D segmentation. In our context, a 3D network can learn the papyrus layer as a connected 2D manifold embedded in 3D space, which should naturally discourage isolated patches or holes in the prediction. The trade-off is computational: 3D convolutions on high-resolution scans are memory-intensive. We will need to train on smaller sub-volumes (patches) and use *sliding-window inference* (discussed later). Nonetheless, many top solutions in similar 3D segmentation competitions (and likely in the scroll Grand Prize challenge) used 3D CNNs or even ensembles of 3D models for best performancescrollprize.org. The consensus in literature is that, given sufficient data and augmentation, 3D models are more accurate and maintain performance better on limited data than 2D, because they leverage the full spatial contextmedrxiv.org. For our baseline, we choose a **3D U-Net** variant as the primary model due to its strong balance of accuracy and manageable complexity.

Our 3D U-Net baseline will be a **moderate-capacity** model to fit within run-time limits. It will have an encoder path with several 3D downsampling steps (e.g. 4 levels deep) and a symmetric decoder. Each level will use multiple 3×3×3 convolutions with normalization (BatchNorm or InstanceNorm) and ReLU/LeakyReLU activations, which is standard in medical modelsemergentmind.com. We can consider using **residual blocks** (like V-Net and nnU-Net do) to ease training deeper networksemergentmind.com. The output will be a sigmoid or softmax 3D volume of the same size as the input patch, representing the probability of the papyrus surface at each voxel. We also employ **deep supervision** (auxiliary outputs from intermediate decoder layers) to help gradients flow and to encourage multi-scale consistencyemergentmind.com – this is proven to stabilize training in 3D segmentation tasksemergentmind.comemergentmind.com. Overall, the 3D CNN provides a strong baseline backbone.

Transformer-based Model (Swin UNETR): For completeness and future scalability, we note that transformer-based 3D networks have shown excellent results in medical imaging. The **Swin UNETR** architecturearxiv.org uses a Swin Transformer encoder that partitions the volume into local self-attention windows but also enables global interactions, addressing the limited receptive field of pure CNNsarxiv.org. Hatamizadeh *et al.* (2022) report that Swin UNETR achieved state-of-the-art performance on the Brain Tumor Segmentation (BraTS) challengearxiv.org. The long-range attention could, in theory, help capture the global shape of a scroll (e.g. recognizing a layer that spans across the field of view) and might better distinguish closely adjacent surfaces (by attending to subtle differences or gaps). However, transformers typically require more data to train effectively and are slower. For a baseline, a full transformer might be overkill and could risk not converging well with limited training volumes (the papyrus data is unique and not abundant). Additionally, the compute overhead of Swin UNETR might strain the 9h inference budget unless carefully optimized (perhaps by model pruning or using smaller window sizes). Therefore, we opt to stick with the **convolutional U-Net family** for the baseline, while acknowledging that a refined solution could integrate transformer blocks or use a pretrained Swin UNETR to boost performance. We will ensure our design (e.g. using large patch sizes and heavy augmentation) captures enough context so that a CNN can approximate the benefits of global attention.

In summary, the baseline architecture will be a **3D U-Net** (or V-Net-style) model for strong volumetric segmentation. This choice is justified by extensive evidence of U-Net's success in 3D biomedical segmentationarxiv.org and by the need for topological continuity that 3D context provides. We will mitigate the higher memory cost through patch-based training/inference. A 2.5D model is simpler and could be a fallback if resources are extremely constrained, but its tendency to produce topological artifacts (broken or merged surfaces between slices) makes it less ideal for this particular TopoScore-sensitive challenge. Transformer models offer an exciting avenue but will be noted as potential improvements beyond the baseline.

Loss Functions and Topology-Aware Training

Segmenting thin surfaces in a large volume is a highly class-imbalanced problem – the papyrus surface voxels are vastly outnumbered by background voxels. Moreover, our targets require not just voxel-level accuracy but also **topologically correct surfaces**. To address these, we design a **composite loss function** that combines conventional segmentation losses with specialized topology-aware components:

- **Weighted Binary Cross-Entropy (WBCE):** This voxel-wise loss ensures the model predicts the correct class for each voxel. We weight the positive class (surface) higher because of imbalance (e.g. each surface voxel might get weight `w>1` relative to background) so that missing a surface is penalized more heavily than a background misclassification. Cross-entropy focuses on overall classification accuracy and tends to shape the probabilities correctly, but by itself can struggle with extreme imbalancegithub.com. We include it to provide per-voxel supervision, especially important in regions where the surface is ambiguous or partially labeled.
- **Dice Loss:** We include a soft **Dice coefficient loss** to directly optimize for overlap between prediction and ground truth surface regionarxiv.org. Dice loss is defined as `1 - (2 * |P ∩ G| / (|P| + |G|))` for the papyrus class, which improves segmentation of small structures by considering relative overlap rather than absolute voxel count. Milletari et al. (V-Net) introduced Dice loss specifically to handle cases with very few foreground voxelsarxiv.org. In our case, the surface is a thin sheet – perhaps only 1–3 voxels thick – in a huge volume, so the Dice term is crucial to drive the network to find that needle in the haystack. We will use a *soft* Dice (differentiable approximation operating on probabilities). The **combination** of Dice and cross-entropy is a proven strategy in medical segmentation, balancing region overlap and fine detailsemergentmind.com. Specifically, nnU-Net and others often use an **equal weighted sum**: *Loss = Dice_loss + CE_loss* (or a weighted sum)emergentmind.com. This has been shown to yield better results than either alone, as cross-entropy gives precision on boundaries and Dice gives recall on the whole regionemergentmind.com.
- **Topology-Preserving Loss (TopoLoss):** To minimize topological errors (which directly impact TopoScore), we incorporate a **topology-aware loss** term. One option is the "TopoLoss" from Hu *et al.* (2019)papers.neurips.cc, which uses concepts from persistent homology to enforce that the predicted segmentation has the same **Betti numbers** (count

of connected components and holes) as the ground truth. In essence, this loss makes topological equivalence differentiable by tracking the birth/death of homology features in the probability mappapers.neurips.cc. By adding this term, we penalize the network when it produces an extra disconnected segment or misses a component, or if it creates a hole in a surface where there should be none. Hu *et al.* demonstrated that using such a loss dramatically reduced topological errors like broken connections in fine structures, without sacrificing per-pixel accuracypapers.neurips.ccpapers.neurips.cc. We will compute this by analyzing the prediction probabilities and ground truth via a small persistent homology computation (likely using a library or custom function for 3D 0D/1D homology). The loss encourages the predicted papyrus surfaces to have the same count of continuous sheets (and any enclosed voids) as the truth. In practice, ground truth surfaces likely form one continuous sheet per scroll layer; TopoLoss will push the prediction to also form one piece per layer, avoiding fragmentation or mergers.

- **clDice (Connectivity-aware Dice):** Another complementary loss is **clDice (centerline Dice)**, introduced by Shit *et al.* (CVPR 2021) for tubular structure segmentationarxiv.org. While papyrus sheets are not tubular, the key idea of clDice is to ensure connectivity by comparing the *skeletons* of the prediction and ground truth. **soft-clDice** computes Dice on the voxel-wise skeleton (or centerline) of structuresarxiv.org. For vessels or roads, this ensures that all segments are connected. For our surfaces, we could adapt this by computing a surface skeleton (a medial surface) or treating each papyrus sheet as a 2D structure and computing its 1-voxel thin representation. If implementing clDice directly is complex for surfaces, we might instead apply clDice on a transformed representation (e.g., treat the papyrus surface as a set of 1-voxel curves by slicing it). The rationale is to **penalize any gap** in the surface – if a section of surface is missing, the skeleton of the predicted surface will be shorter or broken compared to ground truth, and clDice loss will increase. The clDice loss is differentiable and has been shown to **preserve connectivity up to homotopy equivalence** in 3D segmentationsarxiv.org. Training with soft-clDice led to significantly more connected predictions in a variety of datasets (including 3D neuron and vessel segmentation)arxiv.org. We anticipate a similar benefit here: by combining clDice with standard Dice, the model is nudged to **fill in any small breaks** in the papyrus layer. This directly helps avoid topological holes (a break in a sheet would appear as a hole from the perspective of topology). If implementing a true 3D surface skeleton is too involved, an alternative is to use **morphological connectivity losses**: for example, one can convolve the predicted segmentation with a 3D 6-connected kernel and encourage agreement with ground truth dilation – effectively rewarding predictions that connect neighboring surface voxels. However, for academic grounding, we stick to known formulations like clDice.
- **Weighted Loss Combination:** Our final loss is a weighted sum of these components. We set the primary balance between **Dice and Cross-Entropy** (e.g. each 0.5) as in nnU-Netemergentmind.com, then add a smaller weight for the topology terms. For instance, $Loss = \lambda_1 \cdot WBCE + \lambda_2 \cdot Dice + \lambda_3 \cdot TopoLoss + \lambda_4 \cdot clDice$, with $\lambda_1=\lambda_2=1$ and λ_3,λ_4 maybe 0.1–0.2 to start. The topology losses should not dominate learning early on (when the network is still figuring out where surfaces are at all), but by training's end, they help fine-tune connectivity. We will monitor the effect: if we find that the network is having trouble achieving basic accuracy due to the added complexity, we can reduce λ_3,λ_4 initially (or apply them after a certain number of epochs once the base segmentation is reasonable).

Both TopoLoss and clDice are **academically validated**: TopoLoss was shown to drastically reduce Betti number errors in segmentationspapers.neurips.ccpapers.neurips.cc, and soft-clDice consistently improved connectivity metrics and graph similarity in multiple 3D benchmarksarxiv.org. In competitions, variants of these losses (or their ideas) have been used for tasks requiring connected segmentations (for example, vessel segmentation challenges where missing a segment is catastrophic – analogous to missing a piece of papyrus). By including them, our baseline directly targets the TopoScore metric.

In addition to these, we also consider standard tricks: for instance, **distance map regression** or boundary loss – sometimes segmenting very thin structures can be aided by predicting a distance transform of the surface and then thresholding. However, since our labels are binary surfaces and we have a tolerance in scoring, sticking to direct segmentation with above losses is straightforward and effective. We may, however, incorporate a **surface distance loss**: a term that penalizes distance between predicted and true surface voxels (using e.g. the average symmetric surface distance). This can be done by multiplying the segmentation by a pre-computed distance field of the ground truth surface. Given that SurfaceDice is the metric, a loss focusing on surface distance could correlate well. In practice, a simplified approach is to weight the loss more on boundary voxels – e.g. apply a higher weight near the surface edges in ground truth (since if you miss by more than 2 voxels, it hurts the SurfaceDice). This can be achieved by a **boundary-weighted BCE or boundary Dice**, which is another tweak we can incorporate if needed.

Finally, we ensure that **unlabeled regions (label=2)** do not contribute to any loss term. In implementation, this is done by masking out those voxels in the WBCE and Dice calculations (so they do not count in union or intersection), and by not considering them in topology computations (e.g. we remove any unlabeled voxels from the ground truth when computing Betti numbers or skeletons). Effectively, we treat unlabeled areas as “don’t care”. This is important to not misguide the topology metrics – e.g. if a papyrus surface passes through an unlabeled zone, the ground truth might look like it’s disconnected (gap in labeling) when in reality it’s just unknown. By ignoring those zones, we don’t penalize the model for either connecting through or not; hopefully, the model will learn to continue a surface through an unlabeled gap if the visual evidence is there.

In summary, our loss function is **multi-component**: it tackles **class imbalance and region accuracy** (Dice + WBCE) as well as **topological fidelity** (TopoLoss, clDice). This combination is informed by both academic research and competition experience. For example, the winning solutions of the scroll challenge reportedly placed heavy emphasis on segmentation continuity and even employed custom losses or post-processing to ensure whole surfacesscrollprize.org. By baking topology awareness into the loss, we aim to get a clean prediction directly, with minimal need for ad-hoc fixes later.

Data Augmentation Strategies

Data augmentation is essential for this task to improve generalization and robustness, especially given the limited number of unique scrolls available for training. Augmentation not only enlarges the effective training set but also helps the model become invariant to irrelevant transformations and noise, which is crucial since the scan conditions or scroll orientations might vary. We adopt a range of **3D augmentations** inspired by both biomedical imaging practice and the specifics of carbonized scroll data:

- **Geometric Transformations:** We apply random spatial transforms to simulate different orientations and deformations of the scroll in the volume. This includes:
    - **Rotations:** Random rotations up to, say, 15–30 degrees around each axis (roll, pitch, yaw)emergentmind.com. Scrolls in the CT may not be perfectly aligned to the axes, so rotation augmentation teaches the model rotational invariance. We limit the angle to keep surfaces still reasonably in frame.
    - **Flips:** Random mirroring along axes (x, y, z) with 50% chance eachemergentmind.com. Since there's no inherent left-right or up-down bias in the X-ray volumes, flipping is a safe augmentation. It effectively generates data from the "mirror image" scroll, which should be equally valid.
    - **Elastic Deformation:** We use small random elastic deformations in 3D to mimic subtle warping of the papyrusemergentmind.com. Ronneberger et al. famously used elastic deformations in 2D U-Net training to simulate tissue deformationsarxiv.org, and it's applicable here because papyrus sheets might not be perfectly flat or could have slight crinkles. We implement this by perturbing a coarse 3D grid of control points (e.g. with random displacements of a few voxels) and interpolating, using an approach similar to elastic augmentation in 3D. This helps the model handle cases where the scroll surface isn't a perfect plane but has minor bends or wrinkles.
    - **Scaling and Translation:** We randomly scale the volumes by a small factor (e.g. ±10% size) and translate them by a few voxelsemergentmind.com. Scaling accounts for any slight calibration differences or if some scrolls appear larger/smaller due to scanning setups. Translation is implicitly done if we allow cropping at random positions (which we do when extracting patches).
    - **Spatial Stretching:** Voxel anisotropy or different aspect ratios could cause surfaces to appear stretched; we simulate slight anisotropic scaling (e.g. stretch or squeeze along one axis by ~5-10%). This, combined with rotation, forces the network to not overfit to one specific orientation or resolution.
- **Intensity Transformations:** The X-ray absorption values can vary between scans (different scanner settings or different material composition). We therefore apply intensity augmentations:
    - **Brightness and Contrast Shifts:** We jitter the intensity by adding a small random bias and scaling by a random factoremergentmind.com. For example, randomly shift the mean by ±10 HU and scale contrast by 10%. This simulates different scan exposures or reconstructions. We ensure after augmentation the intensities are clipped to a normal range to avoid out-of-bound values.
    - **Gamma Correction:** Apply a random gamma curve to intensities (raising to a power between 0.7–1.5)emergentmind.com. This changes the distribution of bright vs dark, effectively simulating different levels of image contrast. It can help if some scroll data is slightly more or less contrasty.
    - **Noise Injection:** Additive Gaussian noise can be added to the voxel intensitiesemergentmind.com to mimic increased photon noise. Given some scans

might be noisier (depending on CT settings), this helps the model learn to be resilient to noise. We calibrate the noise level to be similar to what we observe in real data (e.g. if the image has ~1% fluctuation).
    - **Blur/Sharpen:** With a small probability, apply a slight blur (Gaussian blur σ ~0.5) or sharpening filter. This covers the scenario where some images might be slightly out of focus or over-smoothed by the scanner's reconstruction algorithm. Similarly, a sharpening can simulate higher acutance images. These are minor but can improve robustness.
- **Elastic Contrast/Artifact Augmentations:** Carbonized papyrus scrolls have some unique visual patterns – for instance, ash or char might create patchy textures, and there could be striping artifacts from CT. We incorporate:
    - **Local Elastic Contrast Changes:** Similar to elastic deformation, but instead of moving voxels, we could randomly stretch intensity in local regions – for example, apply a random intensity scale factor to a random 3D subregion. This mimics uneven illumination or parts of the scan being slightly different in calibration.
    - **Cutout / Missing Parts:** We randomly zero-out or mask small cuboidal regions of the input (a form of cutout augmentation). This can simulate missing data or occlusions, and forces the model to rely on surrounding context. Given unlabeled regions exist, this is realistic – parts of some scrolls may not have labels or may be eroded; the model should still perform around those.
    - **Misalignment simulation:** In case the dataset might have slight misalignments between slices (though probably not, since CT is consistent), we could simulate by shifting alternate slices subtly. However, this may be overkill.

We apply these augmentations with reasonably high probability each epoch, and often in combination (many libraries like MONAI or Albumentations allow composing 3D transforms). The augmentations are applied on-the-fly during training to continuously provide new variants. We ensure that when we transform the image, we apply the same transform to the label mask (for geometric ones) to keep them aligned.

All these augmentations are **grounded in common practice**. For example, nnU-Net relies heavily on spatial augmentations (rotation, scaling, elastic) and intensity augmentations (gamma, noise) to generalize across medical scansemergentmind.com. These have been shown to improve robustness to scanner variability and anatomical diversityemergentmind.com. In our case, variations might come from different scrolls (some more burnt, some less, some scanned with different parameters). Augmentation also helps mitigate overfitting given the limited training data – the model can't just memorize a particular texture of papyrus; it must recognize it under distortions.

Specifically, **elastic deformation and small intensity changes** are particularly relevant: papyrus scrolls may have slight physical deformations (not perfectly cylindrical or flat) and the ink or papyrus reflectivity might vary, so training on deformed and contrast-varied samples prevents the model from latching onto one rigid representation. Also, **test-time augmentation** (which we discuss in inference) is essentially using these same transformations (mostly flips/rotations) at test time to ensemble predictions. It's worth noting that Kaggle solutions often report heavy use of TTA. For instance, a high-ranking solution applied *mirroring along all axes and 90° rotations, yielding 8 augmented predictions per volume*emergentmind.com and averaged them for a more stable output. We incorporate similar ideas.

One must be careful not to introduce augmentations that violate physical realism: for example, extremely large rotations could put the scroll half out of frame, and very strong elastic warps could create unrealistic gaps or overlaps in the papyrus that wouldn't occur physically. We therefore constrain augmentation parameters to mild ranges (as above). The aim is to expose the model to plausible variations without confusing it with impossible structures.

To summarize augmentation: our baseline uses **heavy data augmentation** (both spatial and intensity) to improve generalizationemergentmind.com. Techniques like random rotations, flips, elastic deformations, brightness/contrast shifts, gamma adjustment, and noise injection are all includedemergentmind.com. This is in line with both academic best practices and competition experience, where heavy augmentation consistently boosts performance for segmentation tasksemergentmind.com. It will help ensure our model detects papyrus surfaces reliably even under different noise levels, orientations, or minor damage patterns in the scrolls.

Training Strategy

Patch-Based 3D Training: Due to memory limitations, we cannot feed entire 3D volumes (even cropped) into the model at once if they are large. Instead, we adopt a **sliding patch** training approach. We sample random 3D patches of a fixed size (e.g. 128×128×64 voxels, depending on GPU memory) from the volumes during training. Each patch comes with its corresponding label sub-volume. To ensure the model learns to identify surfaces in context, we choose patch sizes that are large enough to contain meaningful stretches of papyrus layer (preferably an entire local "tile" of the surface and some surrounding background). For example, a patch might cover a $5mm^3$ region of the scroll, which could include a portion of a papyrus sheet and some blank space. We avoid patches that are too small (e.g. 32^3) which might only contain either background or a fragment of surface without context.

Foreground Sampling: A naive random patch sampling might often pick mostly empty background (since background occupies much of the volume). Training on too many empty patches is inefficient and could bias the model to just predict background everywhere (which trivially gives low loss but poor result). To counter this, we implement a **sample balancing strategy**: we ensure that a certain fraction of training patches contain at least some foreground (surface) voxelsemergentmind.com. For instance, we can require that 50% of patches have >1% of voxels labeled as surface. This can be done by sampling uniformly from the volume but rejecting/reshuffling if the patch has no foreground, or by maintaining two queues of patches (foreground-heavy and random). The nnU-Net framework uses a similar strategy of *mandating foreground content in a fixed fraction of patches* to handle class imbalanceemergentmind.com. This way, the model frequently "sees" papyrus layers during training and learns their features, instead of degenerate training on empty images. Over the course of training, this approach significantly boosts convergence on the target classemergentmind.com. We still allow some patches of pure background to teach the model that not every region contains papyrus (reducing false positives), but we cap their proportion (e.g. at most 50%).

Batch Size and Optimization: We use the largest batch size that fits in GPU memory with our patch size (likely batch 2 or 4 for 3D patches on a 16GB GPU). A small batch is common in segmentation due to large patch sizes, so we will use **gradient accumulation** if needed to simulate a larger batch for stable optimization. The optimizer can be **AdamW** (Adam with decoupled weight decay) or stochastic gradient descent with momentum (SGD). nnU-Net found SGD with Nesterov momentum (0.99) effective for segmentationemergentmind.com, but AdamW often converges faster. We might start with AdamW (initial learning rate ~1e-3) and then switch or try SGD if needed. We will employ a **learning rate schedule**, such as a poly (polynomial decay) schedule where the LR starts at lr0 and decays to 0 by end of training. This is standard in segmentation and was used in nnU-Net and other challengesemergentmind.com. We could also consider a **cosine annealing** schedule with restarts if fine-tuning.

Class Imbalance Handling: Apart from loss weighting and patch sampling, we monitor the **class balance per patch**. If we see that even in "foreground" patches the surface occupies a very small fraction, we might increase the Dice loss weight or use **Tversky loss** (which is a variant giving more weight to false negatives) to further focus on the thin structures. The Tversky loss (with parameter α,β to control FNs vs FPs) can be adjusted to emphasize recall (important not to miss surface pieces, since a missed piece breaks topology). For baseline, we stick to weighted Dice/BCE as described, but this is a lever to tune.

Training Schedule: We train for a sufficient number of epochs to ensure convergence (maybe 100–200 epochs, but this depends on seeing many patches). Instead of epochs, sometimes it's easier to define a total number of iterations since our data sampling is stochastic. We could target e.g. 100k iterations, with evaluation every few thousand. Early stopping might be applied if validation metrics plateau.

Mixed Precision: We enable **mixed precision training** (FP16 autocasting) to accelerate training and reduce memory usage, which is a widely used technique including in nnU-Net's official implementationcatalog.ngc.nvidia.com. Mixed precision allows us to use larger patches or batch sizes by halving memory for activations, and speeds up tensor core operations on modern GPUs. We keep a careful eye on loss scaling to avoid numeric instability, but generally frameworks handle this. Mixed precision has become standard in production deep learning due to NVIDIA's optimizations, and it does not degrade accuracy significantly.

Regularization: In addition to augmentation, we use **weight decay** (especially if using Adam, weight decay ~1e-5 or 1e-4) to regularize. Also, as mentioned, we incorporate **deep supervision**: intermediate outputs from decoder layers (at coarser scales) that each compute a loss (weighted smaller than main loss)emergentmind.com. This forces the network to learn consistent predictions at multiple resolutions, helping convergence and possibly improving the VOI metric (since coarse segmentation gets refined). We sum the deep supervision losses (with decreasing weight for lower resolutions).

Validation During Training: We set aside a validation strategy (detailed in the next section) and periodically evaluate the model on a validation set or fold. Importantly, we will **monitor the competition metrics** (SurfaceDice, VOI, TopoScore) on validation rather than just plain Dice. This is because a model could improve voxel-wise accuracy but introduce a topological error –

e.g. split one surface into two almost-perfect halves, which hardly changes Dice but would hurt TopoScore severely. Monitoring TopoScore during training is tricky (it's not a smooth metric), but we can compute it on validation volumes for checkpoints. We may favor a checkpoint that has slightly lower Dice but higher TopoScore if the overall weighted score is better. That said, often improving overall segmentation quality will improve all metrics, but the balance is key.

Training Reproducibility: We set fixed random seeds for numpy and torch for initial runs to ensure reproducibility of the baseline. We will likely do multiple training runs if we use cross-validation (which we might due to few volumes), but each run can be deterministic for comparison. We log training curves for the various loss components to see that they're all decreasing. If the topology loss is not decreasing much, we might increase its weight gradually. Conversely, if we notice the topology loss dominating and causing regular Dice to stagnate, we dial it down a bit.

In summary, the training strategy revolves around **patch-based stochastic training with heavy augmentation, balanced sampling, and a composite loss**. This is fully in line with best practices such as nnU-Net, which emphasizes large patch sizes, extensive augmentation, and mixed precision to effectively learn from limited dataemergentmind.comemergentmind.com. Our focus on including foreground in patches and using a robust optimizer schedule aims to ensure the model learns the papyrus surfaces as completely as possible. By the end of training, we expect the model to output probability maps where papyrus surfaces are clearly highlighted as continuous sheets.

Inference and Post-processing

At inference time, we need to deploy our trained model on the full 3D volumes and produce the binary segmentation within the 9-hour execution budget. The key challenge is that the model was trained on patches, so we must tile the volume and stitch outputs seamlessly. We also apply test-time augmentations to maximize accuracy and use post-processing to fix any minor topological defects.

Sliding Window Inference: We perform **sliding-window prediction** over each volumeemergentmind.com. That is, we divide the volume into overlapping sub-volumes (of the same size as training patches or maybe slightly larger if possible) and run the model on each. We choose an overlap (e.g. 50% overlap in each dimension) such that every voxel is covered by multiple patchesemergentmind.com. Overlap is important to avoid edge effects: a patch-based model can be less accurate at the patch borders due to lack of context. With overlap, each internal voxel will be predicted multiple times by different patches, allowing us to composite a better result. We use a **Gaussian weighting** scheme when blending patches: assign higher weight to the center of patches and lower weight at the borders, then average the overlapping predictions weighted by this Gaussian maskemergentmind.com. This effectively smooths out any seams between patches. This technique, advocated by nnU-Net, ensures that no visible boundary exists in the final probability map even if the model had some boundary artifactsemergentmind.com.

For each volume, if it's very large, we can split the sliding window processing into chunks (e.g. process one quarter of volume's patches, then next) to fit in memory, but given 9h time, we can sequentially do it. We will utilize multiple CPU threads for data loading and augmentation if any (though at inference we usually skip augmentation except TTA). The heavy lifting is done on GPU. Each patch prediction is independent, so this could be parallelized if multiple GPUs were available; in a Kaggle notebook environment, typically one GPU is available, so we do it sequentially.

Test-Time Augmentation (TTA): To improve prediction robustness, we apply **TTA** by ensembling the model's outputs under multiple transformationsemergentmind.com. Specifically, we use **mirroring (flips) along axes and 90° rotations** as our augmentations, as these are nearly zero-cost symmetries for CT data. There are 8 possible combinations of flips (flip or not in each of x,y,z) – which correspond to the 8 octants of rotation by 180° – and additional 90° in-plane rotations (for instance, rotating the XY plane by 90° is like swapping axes). A common TTA set is the 8 transforms consisting of all flips and a 90° rotation of the XY-planeemergentmind.com. As reported by Kaggle solutions, this yields 8 predictions per patch, which are then averagedemergentmind.com. We will implement this by, for each patch:

1. Generate transformed versions (flipped/rotated) of the patch.
2. Run the model on each transformed patch.
3. Inverse-transform each output probability map back to the original orientation.
4. Average all the probabilities.

The resulting averaged probability is then used for thresholding. This ensembling cancels out some random prediction noise and biases (e.g. if the model is slightly biased to predict surfaces oriented a certain way, rotations will balance it). TTA can noticeably improve metrics – it's almost a free boost using the symmetry of the problem (CT physics unchanged under flips/rotations). We must be mindful of the time cost: 8x TTA multiplies inference time by 8. If a single pass for a volume took 1 hour, TTA would take 8 hours, which is borderline for 9h. However, typically a single pass might be shorter; we will measure it. If needed, we can reduce TTA (e.g. use only flips (×2 or ×4) without rotations to cut time). Alternatively, we can selectively apply TTA on patches near uncertain regions (but that complicates things). Given accuracy is priority, we lean towards using the full flips+rotations TTA if time permits, as done by top entrantsemergentmind.com.

Binarization of Output: After obtaining the probability map for the entire volume (through patch assembly and TTA averaging), we threshold it to get a binary segmentation. Normally a threshold of 0.5 is used for balanced classes, but since our foreground is sparse and we care about minimizing false negatives (missing surface) perhaps more than a few false positives (which might be pruned), we might choose a slightly lower threshold like 0.4. We can determine this threshold by analyzing the validation set – e.g. pick the threshold that maximizes the SurfaceDice or the combined score on val. This thresholding yields the initial binary mask of predicted surfaces.

Connected Component Analysis: Now, we apply a key post-processing step: **connected component filtering**. From experience, if the model mistakenly segments an isolated blob that is not actually part of a papyrus surface (e.g. a noise artifact), it will likely be very small or oddly shaped compared to true surfaces (which are large continuous sheets). We compute 3D connected components of the binary prediction and remove any components that are clearly spurious. For example, we can drop any component with volume less than a certain threshold (perhaps <100 voxels, tuned to be smaller than any real papyrus patch) or that doesn't have a sheet-like shape. The challenge is that a broken papyrus layer could also appear as two components if a section is missed. We don't want to remove a legitimately large piece that was only disconnected due to a model mistake – ideally, we'd fill that gap instead. But if the model completely missed a section, post-processing can't magically recover it, whereas it's safer to remove tiny false positives (likely noise). As a rule, **all true papyrus surfaces in a chunk should be fairly large continuous areas**, so removing tiny bits is safe. nnU-Net also advocates removing small isolated components if the target anatomy is known to be one pieceemergentmind.com. In our case, each scroll layer (recto surface) is one continuous piece spanning the chunk (unless the chunk cuts it). So we expect maybe one or few components, not dozens. Therefore, after connected-component analysis, we keep only the largest few components that account for the expected number of surfaces and discard the restemergentmind.com. If multiple layers are present in one chunk, we keep that many largest components (if known), or we keep everything above a size threshold.

Morphological Smoothing: Next, to address any small holes or gaps *within* a predicted surface, we apply morphological operations. A predicted papyrus sheet might have occasional one-voxel holes (perhaps where the model was uncertain or where there was an unlabeled region). These holes can harm TopoScore (because they introduce an unnecessary handle/topology feature) and are not desired since a real papyrus sheet is continuous (except maybe small physical damage, but those might not be labeled as holes unless they truly are missing papyrus). We perform a **binary closing** operation in 3D: basically, a dilation followed by an erosion with a small spherical structuring element (like radius 1 voxel). This will fill in tiny voids in the segmentation while largely preserving the outer shape. We choose the structuring element small enough to not accidentally connect two distinct surfaces that are a few voxels apart. If two surfaces are separate by, say, >3 voxels of air, a radius 1 closing won't join them (it only fills holes up to 2 voxels across). But if the model left a 1-voxel gap in a surface, this will seal it. This approach is supported by the pipeline described in a related study, where after binarizing the volume, they *smoothed pages and filled holes iteratively* to produce continuous page segmentsnature.com. That ensures topologically correct surfaces (no holes). We do have to be cautious not to over-smooth edges; however, since SurfaceDice allows a 2-voxel tolerance, a slight smoothing is not detrimental and might even help by aligning surfaces more cleanly.

We can also apply a slight **median filter** on the binary mask to remove salt-and-pepper noise (individual outlier voxels). A 3×3×3 median filter will eliminate single-voxel false positives or negatives. However, if our connected component removal already took care of small positives and the closing took care of small negatives (holes), a median filter might be redundant.

After these steps, we might end up with a prediction that slightly overestimates the surface extents (closing can add a voxel layer on the inside of a surface). If needed, we can intersect the prediction with a mask of "possible papyrus region" to trim any parts that clearly go into invalid areas. For example, if the scroll is roughly within a certain radius, we wouldn't expect surface far outside that. But such a mask is only available if we pre-computed something from the volume (like where attenuation is high, etc.), and is not strictly necessary.

Resource Optimization: We design the inference pipeline to be efficient:

- Use **half-precision** for model inference as well (most frameworks allow it easily), to speed up compute and reduce memory. This is safe because our model outputs probabilities and a minor precision loss does not affect final binary much.
- Use **CPU multithreading** to preload the next patch while GPU is processing the current patch. This overlaps IO/computation.
- If 9h is still tight, consider using a slightly smaller model or fewer TTA augmentations to save time.
- Ensure we only write out final results (maybe as a compressed format) to avoid slow disk writes of huge volumes.

We also consider that the final submission might require results in a certain format (perhaps run-length encoding or specific file format). We will include the conversion in the pipeline and ensure it's also optimized (vectorized where possible).

Validation of Inference: We simulate the inference on our validation set to ensure the pipeline yields good metrics. We particularly look at TopoScore – if we see any topological errors in validation predictions, we examine them. If, for instance, a surface was predicted as two pieces (TopoScore hit), we see if the gap was small – if yes, maybe increasing the closing radius or adjusting threshold could join them. If the gap was large (model truly missed a part), only improving the model (or labeling those unlabeled regions if any) can fix it. If we see false surface fragments, we might tighten the component size threshold. This tuning, however, must not be overfit to validation – just ensure general settings that make sense.

Metric Computation: We then compute the SurfaceDice@2, VOI, and TopoScore on validation. SurfaceDice we can compute by extracting the surface voxels (on prediction and truth) and checking distances (<=2). VOI requires computing entropies of the confusion matrix of clusters (which for binary segmentation can be simplified to foreground vs background partitions). TopoScore, if not provided, we approximate by comparing Euler characteristics or connected component counts. For instance, a simple TopoScore definition could be: 1 minus (normalized count of topological differences). But since the competition likely has a specific formula, we may implement their provided evaluator for validation (if available from the competition notebooks). Either way, our validation aims to mirror the leaderboard metrics.

Summary of Inference: Using **sliding-window with overlap and TTA**, we ensure a high-quality probability map covering the whole volumeemergentmind.com. Then, through **thresholding, component filtering, and morphological cleaning**, we refine the binary mask to be as continuous and accurate as possiblenature.comemergentmind.com. These steps are common in production segmentation systems – e.g., nnU-Net's post-processing includes removing small stray components and ensuring known structures are single connected piecesemergentmind.com. By doing this, we directly address the TopoScore (by eliminating extra components and closing holes) and slightly improve SurfaceDice (by filling near-misses within 2 voxels). The end result is a set of segmented papyrus recto surfaces that can be used for downstream tasks like virtual unwrapping.

Validation Protocol and Evaluation Metrics

Because the ultimate goal is performance on the competition metrics, our validation strategy must reliably estimate those metrics and guide model development. We likely have a limited number of labeled scroll volumes (e.g., perhaps 2–3 training scrolls split into chunks). Using all for training might risk overfitting without an external check, so we adopt either a **cross-validation** or a dedicated hold-out approach:

- If there are, say, 3 scrolls worth of data, we perform a 3-fold cross-validation: train on 2 scrolls, validate on the 1 left-out, rotating each time. This gives three models and three validation results. We can average their scores to estimate leaderboard performance. We could also ensemble these models for final submission if time allows, but baseline will likely just use one (perhaps the one with best val or a model fine-tuned on all data).
- If there is a distinct validation set provided or if we choose to hold out part of one scroll's chunk as validation (less ideal due to distribution shift), we use that.

During validation, we compute **SurfaceDice@2, VOI, and TopoScore** exactly as the competition defines. If an official evaluation script is given (often Kaggle provides one in a notebook), we will use it. Otherwise:

- *SurfaceDice@2:* We compute the set of surface voxels for prediction and ground truth. A voxel is considered surface if it's labeled as papyrus and is adjacent to background (in 26-neighborhood) – essentially the boundary of the segmentation. Then we find, for each predicted surface voxel, the distance to the nearest ground truth surface voxel (using a distance transform) and vice versa. We mark those within 2 voxels. SurfaceDice is `2 * |{pred surface voxels within 2 of GT}| / (|pred surface| + |GT surface|)` pure.uva.nl. We will implement this carefully, taking into account anisotropic spacing if any (using physical distance). The tolerance τ=2.0 means 2 voxels in physical terms (post spacing alignment).
- *Variation of Information (VOI):* This is the sum of two conditional entropies between the predicted and true segmentation. For binary segmentation, it simplifies to a function of false negative and false positive rates. We can compute VOI by:
  - Compute the probabilities p(i,j) of a randomly chosen voxel falling into category (pred=i, true=j) for i,j ∈ {0,1}. From these, compute H(Pred|True) and H(True|Pred). The competition likely provides a direct way to compute VOI_score as 1 - (VOI / log 2?) since they mention VOI_score ∈ [0,1]. Possibly they normalize it. We'll follow their formula.
  - Alternatively, treat segmentation as clustering: all foreground voxels vs background cluster, then VOI measures the information difference. We might rely on the given script or a library like `scikit-image` (which has VOI in some form as variation_of_information).
- *TopoScore:* This likely checks the correctness of topology. It might be defined such that 1.0 means the Euler characteristic of prediction equals that of ground truth (or specifically connected components count equals, and no hole differences). A possible implementation:

    - Compute the number of connected foreground components in pred and GT (maybe also the number of holes (handles) in each). If both numbers match, TopoScore = 1, if not, degrade.
    - The Kaggle discussion might have formula, but since not accessible, we assume TopoScore is some normalized count. One way: If there are N_gt surfaces in ground truth for the chunk and N_pred in prediction, then component score = exp(-|N_gt - N_pred|). Similarly for holes (which likely should be 0 ideally). Then combine. However, since TopoScore is weighted 0.30 in final, it's probably an average of some per-structure matching score.
    - We can explicitly check common topological errors: If prediction has an extra surface piece (false extra component) or missed one (false split), that's a topological error. Same for any loop (hole) in a surface that ground truth doesn't have or vice versa. We will design TopoScore in val to penalize those events heavily.

Given the complexity, we will rely on verifying qualitatively as well: we visualize a few slices or 3D renderings of the prediction vs ground truth. This helps catch issues like slight offsets (which could hurt SurfaceDice if >2 voxels) or small holes.

Validation Frequency: We perform validation periodically (e.g. every epoch or every few epochs, since it might be heavy to run full volume inference). To speed up, we might validate on a sub-sampled volume or only on crucial slices during training. But ultimately, before finalizing, we validate on the full set. We then pick the model epoch that had the highest *combined score* (0.30*Topo + 0.35*SurfDice + 0.35*VOI) on validation. That model will be used for test inference.

Metric-driven Decisions: If during training/validation we observe that one metric is lagging (e.g. VOI is high meaning poor, or TopoScore is low), we adjust accordingly:

- Low TopoScore might mean the model is fragmenting surfaces. We could increase the weight of TopoLoss/clDice in loss, or refine post-processing (maybe our closing radius was too small to join fragments).
- Low SurfaceDice with decent Dice could mean boundaries are off by >2 voxels often. This might suggest maybe our threshold is wrong or the model is slightly under/over segmenting. We might then adjust threshold or consider incorporating a boundary-aware loss. Possibly, computing distance transform of surfaces and using a regression loss on that could help align surfaces to within tolerance.
- VOI is related to over-/under-segmentation as well. A high VOI (worse) could indicate either we have too many false positives (increasing entropy of prediction given truth) or too many misses. Monitoring precision and recall separately would help. If false positives (FPs) are an issue (maybe model paints some areas as papyrus that aren't), we could raise the classification threshold or apply more aggressive component removal. If false negatives (misses) are the issue, perhaps decrease threshold or ensure augmentation covers more variations to not miss faint surfaces.

We will also adhere to **patient-level (scroll-level) validation splits** to avoid any leakage of correlated slices between train and valemergentmind.com. This ensures our validation truly tests generalization to a new scroll. We also track metrics per volume if multiple, since one scroll might be easier than another (due to preservation state or scanning differences). This can inform if our model is overfitting to a specific scroll's texture.

Finally, we note that the **evaluation on the platform** will be done on hidden test scans. Our whole pipeline must therefore be deterministic and robust. We include assertions or checks in the code (e.g. ensure no patch left unsegmented, ensure output dimensions match input, etc.) to avoid any runtime issues. And we profile the inference speed on validation to confirm it's within the limit (with some margin).

By validating thoroughly and using the competition metrics as our guide, we align the baseline with the **leaderboard objectives from the start**. This metric-driven development is key in competitions: for instance, if TopoScore is novel, many top competitors would implement something like TopoLoss early (as we did) to directly optimize it. We believe our validation approach and loss formulation have positioned us well in that regard.

Optimization and Deployment Considerations

To ensure the baseline is **production-ready** under competition constraints, we address a few practical considerations:

- **Runtime efficiency:** The 9h notebook limit requires efficient use of time. Our pipeline of sliding-window inference with TTA is the most time-consuming part. We will use optimized libraries (PyTorch's CUDA ops, perhaps MONAI for sliding window which is highly optimized) and vectorized numpy for any post-processing. We also consider splitting work: for example, after one volume's patches are processed, we could start writing results or computing metrics while the next volume is processing in parallel threads. However, in Kaggle's single GPU, multi-thread on CPU is the main concurrency we get. As mentioned, mixed precision speeds up each patch inference ~2x on tensor cores, so that is enabled. If we have a large GPU (like 24GB or more), we might increase patch size to reduce the number of patches (thus fewer model runs). It's a trade-off between single-run cost and total runs. We will experiment with the largest patch that fits.
- **Memory usage:** We ensure to delete or reuse large arrays when done (especially probability maps for large volumes). We process volumes sequentially to not hold multiple volumes in memory. If needed, we write intermediate predictions to disk and reload for post-processing one by one, to avoid cumulative memory usage. The code will be structured to handle one volume at a time.
- **Reproducibility:** We encapsulate preprocessing and training steps such that the same environment yields the same result. We use fixed seeds for splitting and augmentation where possible. We also plan to release the code (in compliance with competition rules) with clear documentation, so that others can reproduce the baseline on their systems.
- **Robustness:** The baseline should not assume anything too specific; e.g., we assume each chunk has at least some papyrus. If a test volume had no papyrus at all, our model might predict none or some noise. The metric convention likely says that if both pred and truth

are empty, that's a perfect score for that casekaggle.com (SurfaceDice=1 by convention in empty case). We might implement a check: if the model predicts almost nothing and maybe the volume intensity indicates nothing (all air), we output empty directly to save time. But careful: it's possible a scroll chunk could have just unlabeled because maybe no ground truth region – but in test, unlabeled unknown. Hard to say. We'll rely on model output.

- **Leaderbord relevance:** Our choices (3D U-Net, Dice+CE loss, heavy aug, TTA, component filtering) are all well-known ingredients in winning solutions for segmentation challenges. For example, the nnU-Net baseline itself won multiple biomedical competitions with essentially these componentsemergentmind.comemergentmind.com. The Vesuvius Challenge winners also used ensembles of segmentation models and careful surface extractionscrollprize.org. By building a strong baseline, we put ourselves in a competitive position. Any additional improvements (like using multiple models or refining with larger networks) can be added on top of this baseline.

In conclusion, this baseline strategy provides a **comprehensive, production-quality segmentation pipeline** for papyrus surface detection. It emphasizes data normalization, a powerful 3D model architecture, tailored loss functions for topology, extensive augmentations, and careful inference processing – all backed by academic literature and competition experience. We expect this approach to yield a high SurfaceDice (by capturing surfaces within tolerance), low VOI (by accurate overall segmentation), and high TopoScore (by maintaining connectivity), thereby achieving a strong combined score. By following this plan, one can train a model and deploy it within the given constraints, serving as a solid foundation for further enhancements in the quest to virtually unroll and read the ancient Herculaneum scrolls.

Sources:

- Kaggle Vesuvius Challenge Descriptionkaggle.comkaggle.com
- nnU-Net Framework Best Practices (Isensee et al.)emergentmind.comemergentmind.comemergentmind.com
- Milletari et al., *V-Net: Fully Convolutional Neural Nets for Volumetric Segmentation*arxiv.orgarxiv.org
- Hatamizadeh et al., *Swin UNETR: Swin Transformers for 3D Segmentation*arxiv.orgarxiv.org
- Hu et al., *Topology-Preserving Deep Image Segmentation (TopoLoss)*papers.neurips.ccpapers.neurips.cc
- Shit et al., *clDice – Topology-Preserving Loss for Tubular Structures*arxiv.orgarxiv.org
- Ottesen et al., *2.5D vs 3D Segmentation of Brain Metastases*frontiersin.org
- Scientific Reports, *Virtual Unwrapping of Sealed Manuscripts Pipeline*nature.comnature.com