ODC AI Hackathon Journey: Brain Tumor Segmentation (BraTS)

A comprehensive story of our three-day journey tackling the Brain Tumor Segmentation challenge at the ODC AI Hackathon.

Day 1 — Data Preprocessing, Literature Review, and Architectural Exploration

1. Image Preprocessing and Data Preparation

On the first day, our primary focus was building a reliable preprocessing pipeline for the BraTS dataset. Since BraTS consists of multi-modal 3D MRI volumes, careful preprocessing was essential to ensure stable and efficient model training.

We implemented several preprocessing steps:

- **Multi-modal stacking**: Combined T1, T1ce, T2, and FLAIR modalities into a unified multi-channel 3D input to preserve complementary anatomical and pathological information.
- **Intensity normalization**: Applied modality-wise normalization to reduce variations caused by different scanners and acquisition protocols.
- **Spatial resizing and cropping**: Standardized volume dimensions and removed irrelevant background regions to reduce computational overhead.
- **Label encoding and class mapping**: Converted segmentation masks into structured tumor sub-regions (Whole Tumor, Tumor Core, Enhancing Tumor).
- **Patch-based sampling**: Extracted 3D patches from volumes to address GPU memory limitations and increase training efficiency.
- **Data augmentation**: Introduced spatial and intensity-based augmentations (e.g., flipping, rotation, noise injection) to improve model generalization.

This preprocessing pipeline was designed to balance data fidelity with computational feasibility, which is critical for large-scale 3D medical image segmentation.

---

2. Literature Review: Understanding the State-of-the-Art in BraTS

Before diving into implementation, we conducted an extensive literature review to understand what approaches have proven successful in BraTS challenges. We studied several key research papers that shaped our understanding of the problem landscape.

2.1 The Evolution of BraTS Architectures

The BraTS challenge has been running since 2012, and over the past decade,**convolutional neural networks (CNNs)** have emerged as the predominant approach. The **U-Net architecture** and its derivatives became foundational due to their encoder-decoder design with skip connections, which preserves spatial information critical for accurate medical image segmentation.

A major advancement came with **nnU-Net**, which has been the base model of winning solutions in BraTS 2020, 2021, and 2022. By 2023, the winning teams began employing ensembles of multiple architectures (like SwinUNETR + nnU-Net) to improve tumor subregion segmentation.

2.2 The Transformer vs. ConvNet Debate

One of the most significant debates in recent medical imaging literature is whether **Transformers** or **ConvNets** are better suited for segmentation tasks.

The Case for Transformers:

- Ability to learn long-range spatial dependencies through self-attention mechanisms
- Strong performance in natural image tasks (Vision Transformers, Swin Transformers)
- Scalability with larger datasets

The Critical Problem with Transformers in Medical Imaging: Our research revealed a fundamental challenge: *Transformers are plagued by the necessity of large annotated datasets to maximize performance benefits due to their limited inductive bias*. While such datasets are common in natural images (ImageNet-1k with millions of samples), medical image datasets suffer from the lack of abundant high-quality annotations.

> **Key Insight:** The BraTS-Africa dataset, for example, contains only 95 cases (60 training, 35 validation)—far too small for Transformers to reach their full potential.

2.3 Research Paper Analysis

We analyzed several state-of-the-art papers to inform our architecture decisions:

Paper 1: "Brain Tumor Segmentation in the Sub-Saharan African Population" (SPARK Academy 2025)

This paper explored an ensemble of three architectures: **MedNeXt**, **SegMamba**, and **Residual-Encoder U-Net**. Key findings:

Model	Architecture Type	Best LSD Score	Best NSD Score
MedNeXt	ConvNeXt-based CNN	0.865	0.810
SegMamba	State-space model (Mamba)	0.829	0.770
ResEnc U-Net	Residual CNN	0.822	0.765

Critical observations:

- **MedNeXt consistently outperformed** the other models across both Lesion-wise Dice (LSD) and Normalized Surface Distance (NSD) metrics
- Training duration had a notable impact—MedNeXt achieved its best results at **1000 epochs**
- The paper noted that MedNeXt's ability to preserve spatial information through its ConvNeXt residual blocks and efficient scaling was key to its success

---

Paper 2: "Optimizing Brain Tumor Segmentation with MedNeXt: BraTS 2024 SSA and Pediatrics" (MBZUAI)

This paper from Mohamed bin Zayed University of AI achieved state-of-the-art results on both BraTS-Africa and Pediatric datasets using MedNeXt. Key results:

Dataset	DSC Score	HD95 Score
BraTS-Africa	**0.896**	14.682
BraTS Pediatric	**0.830**	37.508

Critical insights from this paper:

1. **Finetuning strategy matters**: The best performance came from training on combined BraTS Adult Glioma + Africa datasets, then finetuning the final decoder layers on Africa-only data
2. **Schedule-free optimizer**: They introduced the novel Schedule-free AdamW optimizer from Meta AI, which eliminated the need for learning rate schedulers
3. **Ensemble limitations**: Interestingly, their MedNeXt Base + Medium ensemble (0.868 DSC) actually *underperformed* compared to individual finetuned models (0.896 DSC)

Paper 3: "MedNeXt: Transformer-driven Scaling of ConvNets for Medical Image Segmentation" (DKFZ, Original MedNeXt Paper)

This foundational paper explained *why* MedNeXt works so well. The key innovation is that MedNeXt is **Transformer-inspired but fully convolutional**.

The architecture addresses a fundamental question: *Can we get the benefits of Transformers without their data-hungry nature?*

The MedNeXt Block Design (Mirroring Transformers):

1. **Depthwise Convolution Layer**: A depthwise convolution with large kernel (k×k×k) that replicates the large attention window of Swin Transformers, while limiting computational cost
2. **Expansion Layer**: An overcomplete convolution layer with expansion ratio R, forming a transformer-like inverted bottleneck
3. **Compression Layer**: A 1×1×1 convolution performing channel-wise compression

Why Large Kernels Matter:

> Large convolution kernels approximate the large attention windows in Transformers. A 5×5×5 or 7×7×7 kernel can capture spatial dependencies across a much larger receptive field than traditional 3×3×3 kernels.

The UpKern Innovation: The paper introduced **UpKern (Upsampled Kernel Initialization)**—a technique to iteratively increase kernel sizes by initializing large kernel networks with trained small kernel networks through trilinear upsampling. This prevents performance saturation that typically occurs when training large kernel networks from scratch on limited medical data.

Benchmark Results from the Paper:

Network	Type	BTCV	AMOS22	KiTS19	BraTS21
nnUNet	CNN	83.56	88.88	89.88	91.23
UNETR	Transformer	75.06	81.98	84.10	89.65
SwinUNETR	Hybrid	80.95	86.83	87.36	90.48
nnFormer	Transformer	80.76	84.20	89.09	90.42
MedNeXt-L	ConvNeXt	**84.82**	**89.87**	**90.71**	**91.46**

The results were clear: **MedNeXt outperformed every Transformer-based architecture** on all four tasks, including UNETR, SwinUNETR, TransBTS, and nnFormer.

---

3. Key Conclusions from Our Literature Review

Why MedNeXt Beats Transformers (Despite Not Using Attention)

After extensive research, we identified several reasons why MedNeXt emerged as the superior choice:

1. Inductive Bias Advantage

> *"ConvNets have higher inductive biases and consequently, are easily trainable to high performance."* — MedNeXt Paper

Transformers assume no prior knowledge about spatial relationships—they must learn everything from data. ConvNets have built-in assumptions about locality, translation equivariance, and hierarchical features. In data-scarce medical settings, this inductive bias is a *massive advantage*.

2. Large Kernels ≈ Self-Attention (But Cheaper)

The MedNeXt paper demonstrated that large depthwise convolutions can replicate the benefits of self-attention:

- A 5×5×5 kernel captures dependencies across 125 voxels
- A 7×7×7 kernel captures dependencies across 343 voxels
- Combined with the inverted bottleneck design, this achieves transformer-like representation learning at a fraction of the computational cost

3. Compound Scaling Works Better Than Pure Depth

Unlike traditional approaches that just stack more layers (depth scaling), MedNeXt uses **compound scaling** across three dimensions:

- **Depth**: Number of MedNeXt blocks (B)
- **Width**: Expansion ratio (R) for more channels
- **Receptive Field**: Kernel size (k) for larger spatial context

This orthogonal scaling allows MedNeXt to be efficiently scaled for different computational budgets.

4. Residual Inverted Bottlenecks Preserve Semantic Richness

The paper's ablation studies showed that using standard up/downsampling resulted in significantly worse performance (83.13 DSC vs 84.01 DSC on BTCV). The residual inverted bottleneck design preserves contextual richness while resampling—critical for dense segmentation tasks where every voxel matters.

5. Medical Data is Too Scarce for Transformers to Shine

The numbers speak for themselves:

- ImageNet-1k: 1.2 million images
- ImageNet-21k: 14 million images
- BraTS-Africa: **60 training samples**

Transformers need massive datasets to overcome their lack of inductive bias. In medical imaging, we simply don't have that luxury.

---

4. Initial Architectural Hypothesis: Diffusion-Inspired U-Net with Attention

Before finalizing our architecture choices, we explored potential model architectures suitable for the BraTS challenge.

Our first idea was inspired by **stable diffusion architectures**, which combine:

- ResNet-based encoders,
- Self-attention mechanisms,
- Multiple U-Net blocks operating at different scales.

The motivation behind this idea was to leverage the strong representation learning capabilities of diffusion-style U-Nets, which have shown remarkable performance in vision tasks.

However, after deeper analysis, we identified major limitations:

1. **Dimensional mismatch**: Stable diffusion models are primarily designed and pretrained on 2D natural images, while BraTS data is inherently 3D volumetric. Adapting such architectures to 3D would require extensive redesign and retraining.
2. **Computational cost**: Training a diffusion-style architecture from scratch on 3D MRI volumes would be prohibitively expensive in terms of GPU memory and training time, especially within the constraints of a hackathon.
3. **Lack of pretrained 3D diffusion models**: Without reliable pretrained weights in the 3D medical domain, the expected performance gain did not justify the complexity.

As a result, we decided to omit this approach and shift toward architectures specifically optimized for 3D medical segmentation.

---

5. Baseline Approach: U-Net

Our second idea was to adopt the classical **U-Net architecture**, which remains a cornerstone in medical image segmentation.

Reasons for choosing U-Net:

- Efficient encoder–decoder design with skip connections.
- Strong localization capability for pixel/voxel-level segmentation.
- Feasible training on 3D data with moderate computational requirements.

U-Net served as:

- A strong baseline for performance comparison.
- A stable and interpretable model for understanding the behavior of the dataset.

However, we anticipated that U-Net alone might struggle to capture complex tumor structures due to its limited depth and representational capacity.

---

6. Enhancing Feature Representation: SegResNet

To overcome the limitations of vanilla U-Net, we explored **SegResNet**, a residual-based segmentation architecture.

Key advantages of SegResNet over U-Net:

- **Residual connections** enable deeper networks without gradient degradation.
- **Improved hierarchical feature extraction**, which is crucial for modeling complex tumor morphology.

- **Better training stability**, especially for deep 3D architectures.

SegResNet represented a middle ground between simplicity and expressive power, making it a strong candidate for the BraTS challenge.

---

7. Moving Toward State-of-the-Art: ConvNeXt and MedNeXt

Finally, based on our literature review, we identified **MedNeXt** as the most promising architecture for our hackathon.

Why We Chose MedNeXt:

1. **Proven performance**: State-of-the-art on 4 major benchmarks (BTCV, AMOS22, KiTS19, BraTS21)
2. **Data efficiency**: The ConvNeXt inductive bias works well with limited training samples
3. **Scalability**: Compound scaling allows us to adjust for Kaggle's computational constraints
4. **No Transformers needed**: We get transformer-like representation learning without the data requirements

The MedNeXt Architecture at a Glance:

```
┌─────────────────────────────────────────────────────────┐
│                    MedNeXt Block                        │
├─────────────────────────────────────────────────────────┤
│  1. Depthwise Conv (k×k×k) + GroupNorm                  │
│     → Captures local spatial context (like attention)   │
│                                                         │
│  2. Expansion Conv (1×1×1) × Expansion Ratio R + GELU   │
│     → Overcomplete representation (transformer bottleneck)  │
│                                                         │
│  3. Compression Conv (1×1×1)                            │
│     → Channel compression back to C channels            │
│                                                         │
│  + Residual Connection                                  │
└─────────────────────────────────────────────────────────┘
```

---

8. Strategic Decision and Roadmap

After evaluating these architectural directions and our literature review, we finalized our strategy:

- **U-Net** as a baseline model (low complexity, high bias)
- **SegResNet** as a deeper residual-based alternative (medium complexity)
- **MedNeXt** as a modern, high-performance architecture (high complexity, maximum performance)

This spectrum approach allowed us to:

1. Have a fallback if complex models overfit
2. Compare performance across architectural paradigms
3. Validate our literature review conclusions empirically

These choices defined our roadmap for the next two days of the hackathon, where we focused on training, optimization, and comparative evaluation of these models.

Day 2 — From Research to Implementation

On Day 2, we transitioned from research mode to **hands-on implementation**. Our focus was on applying the knowledge we gathered from the literature review and getting actual models running on our dataset.

1. Cloning and Adapting Research Repositories

Our first task was to clone the repositories we found during our literature review. However, we quickly discovered that **research code is rarely plug-and-play**. Each repository had its own assumptions about data format, directory structure, and preprocessing steps.

Challenges we encountered:

Issue	Description	Our Solution
Data format mismatch	Repos expected different input structures	Created custom data loaders
Modality naming	Different naming conventions (t1ce vs t1c vs T1Gd)	Standardized to t2f, t1n, t1c, t2w
Dependency conflicts	PyTorch/MONAI version mismatches	Fixed compatibility issues

Issue	Description	Our Solution
Missing preprocessing	Repos assumed pre-processed data	Built our own preprocessing pipeline

We spent considerable time debugging import errors, fixing deprecated function calls, and adapting the training scripts to work with the BraTS-Africa dataset structure.

---

2. The I/O Bottleneck Problem

A critical insight from Day 1's research was that **I/O overhead can dominate training time**. The BraTS dataset comes in `.nii.gz` (compressed NIfTI) format, which requires:

1. Decompression on every data load
2. Complex header parsing
3. Affine transformation handling

Loading a single 4-modality volume from `.nii.gz` takes **300-500ms**, which becomes a significant bottleneck when training for hundreds of epochs.

Our Solution: Offline Preprocessing to NumPy

We decided to preprocess all data offline and save as `.npy` (NumPy) format:

Format	Load Time	Compression	Training Impact
`.nii.gz`	~400ms	High	Slow epochs, CPU-bound
`.npy`	~10ms	None	Fast epochs, GPU-bound

This **40x speedup** in data loading allowed us to complete more training epochs within Kaggle's 30-hour execution limit.

---

3. Building the Preprocessing Pipeline

We implemented a comprehensive preprocessing pipeline using **MONAI transforms** and custom functions. Here's what each step accomplishes:

Step 1: Multi-Modal Loading and Stacking

```
# Load all 4 MRI modalities and stack into a single 4-channel volume
modalities = ["t2f", "t1n", "t1c", "t2w"]  # T2-FLAIR, T1, T1-contrast, T2-weighted
vol = np.stack([get_data(mod) for mod in modalities], axis=-1)
```

What this does: Combines T2-FLAIR, T1, T1ce, and T2w into a **4-channel 3D tensor** (shape: `[H, W, D, 4]`), preserving complementary anatomical and pathological information from each modality.

Step 2: Foreground Cropping

```
# Remove background (air/skull) to reduce computation
bbox = transforms.utils.generate_spatial_bounding_box(image)
image = transforms.SpatialCrop(roi_start=bbox[0], roi_end=bbox[1])(image)
```

What this does: Uses MONAI's bounding box detection to find the **smallest 3D box containing all non-zero voxels**. This removes ~60-70% of the volume (empty background), dramatically reducing memory usage and computation.

Why we save metadata: We store the original bounding box coordinates so we can reconstruct the full volume during inference for proper submission formatting.

Step 3: Percentile-Based Intensity Rescaling (Optional)

```
# Clip to 2nd-98th percentile and rescale to [0, 255]
lower = np.percentile(image, 2)
upper = np.percentile(image, 98)
image = np.clip(image, lower, upper)
image = (image - lower) / (upper - lower) * 255
```

What this does: Removes extreme intensity outliers (scanner artifacts, noise) by clipping to the 2nd-98th percentile range. This increases contrast and normalizes intensity distributions across different scanners/protocols.

Step 4: Channel-Wise Z-Score Normalization

```
# Normalize each modality independently (zero mean, unit variance)
image = transforms.NormalizeIntensity(nonzero=True, channel_wise=True)(image)
```

What this does: For each of the 4 modalities separately:

- Computes mean (μ) and standard deviation (σ) **only on non-zero voxels**
- Applies z-score normalization: `(x - μ) / σ`

Why channel-wise? Each MRI modality has different intensity characteristics. T1 and T2 images have inverted contrasts, so normalizing them together would be incorrect.

Why non-zero only? Background voxels (value 0) would skew the statistics, leading to incorrect normalization.

Step 5: Padding to Minimum Patch Size

```
# Pad to at least 128×128×128 for consistent patch extraction
patch_size = [128, 128, 128]
pad_shape = [max(min_s, img_s) for min_s, img_s in zip(patch_size, image.shape)]
```

What this does: Ensures all volumes have a minimum size of 128×128×128 by symmetrically padding with zeros. This guarantees that patch-based training can extract at least one full patch from every sample.

Step 6: Foreground Mask Encoding

```
# Create binary foreground mask and concatenate as 5th channel
mask = np.zeros(image.shape[1:], dtype=np.float32)
for i in range(image.shape[0]):  # For each modality
    ones = np.where(image[i] > 0)
    mask[ones] += 1.0
mask[mask > 0] = 1.0
image = np.concatenate([image, np.expand_dims(mask, 0)])
```

What this does: Creates a **binary foreground mask** indicating which voxels contain brain tissue (value=1) vs. background (value=0). This mask is concatenated as the **5th channel**, giving the model explicit information about the brain boundary.

Why this helps: The model can use this mask to:

- Focus attention on brain tissue
- Avoid predicting tumors in background regions
- Learn better boundary representations

Final Output Format

After preprocessing, each sample is saved as three `.npy` files:

File	Shape	Contents
`{id}_x.npy`	`[5, H, W, D]`	4 modalities + 1 foreground mask
`{id}_y.npy`	`[1, H, W, D]`	Segmentation label
`{id}_meta.npy`	`[3, 3]`	Bounding box + original shape

---

4. Median Shape Analysis

We also implemented a utility to analyze the distribution of cropped volume sizes:

```
def collect_cropped_image_sizes(directory, example_ids):
    shapes = []
    for example_id in example_ids:
        _, _, _, image_shape = preprocess_sample(directory, example_id)
        shapes.append(image_shape)

    median_shape = np.median(shapes, axis=0).astype(int)
    print("The median shape is:", median_shape)
    return median_shape
```

Purpose: Determines the optimal patch size for training. If most volumes are smaller than 128×128×128 after cropping, we might use a smaller patch size to preserve more spatial context without excessive padding.

---

5. Deep Supervision Implementation

With the preprocessing pipeline complete, we focused on the **MedNeXt architecture** and implemented **Deep Supervision** to enhance training stability and convergence speed.

> **The Technical Advantage of Deep Supervision:** By adding auxiliary loss branches at multiple decoder stages, we effectively mitigated the 'vanishing gradient' problem. This forced the shallower layers of the network to learn more discriminative features early on, leading to better spatial representation and a more robust final segmentation mask compared to standard single-loss training.

---

6. Kaggle Optimization Strategy

To navigate the **Kaggle 30-hour execution limit**, we implemented several optimizations:

1. **Pre-computed `.npy` files**: Eliminated I/O bottleneck (40x faster loading)
2. **Small learning rate + capped epochs**: Ensured stable convergence without timeout risk
3. **Gradient checkpointing**: Reduced GPU memory usage to fit larger batches
4. **Mixed precision training**: Used FP16 to speed up computation by ~2x

This combination of preprocessing innovation and resource management provided the performance boost we needed heading into the final day.

Day 3 — Benchmarking, Diagnostics, and Strategic Optimization

On our final day, we moved to full-scale inference and submission. We implemented an **RLE (Run-Length Encoding)** pipeline to format our segmentation masks and ran parallel benchmarks to evaluate our model ensemble.

Model Selection Strategy

Before diving into the results, it is important to address our **model selection strategy**.

You might ask: *'Why spend time on simpler models like U-Net when state-of-the-art architectures like MedNeXt exist?'*

Our reasoning was based on **risk management**. In a worldwide competition like this, the intrinsic complexity of the test data is often 'blurred'—we cannot know strictly how noisy or heterogeneous the private test set is. A highly complex model runs a higher risk of **overfitting** if the target data is simpler than expected.

To mitigate this, we deliberately tested a **spectrum of model complexities**:

1. **U-Net:** Low complexity (High bias, low variance baseline).
2. **SegResNet:** Medium complexity.
3. **MedNeXt:** High complexity (Low bias, high variance risk).

The Benchmarking Results

Our results confirmed the hierarchy, but also justified the risk management approach:

Model	Score	Complexity Level
U-Net	0.62	Low (baseline)
SegResNet	0.71	Medium
MedNeXt	0.76	High

- **U-Net** provided a safe baseline of **0.62**.
- **SegResNet** jumped to **0.71**, proving that residual connections were necessary to capture deeper features without signal degradation.
- **MedNeXt** ultimately led with **0.76**.

Validating Our Literature Review: These results aligned perfectly with what the papers predicted. The gap between U-Net (0.62) and MedNeXt (0.76) demonstrated the power of modern ConvNeXt-based architectures. Interestingly, our MedNeXt score was constrained not by architecture, but by training time—the papers showed that MedNeXt truly shines at 500-1000 epochs, while we were limited by Kaggle's execution constraints.

While MedNeXt was our winner, the score was lower than anticipated. Our diagnosis revealed that this architecture is 'data-hungry' in terms of *epochs*, and typically requires a **high-regime training schedule (500–1000 epochs)** to converge—a luxury we did not have due to Kaggle's execution timeouts.

Bridging the Performance Gap: Three Targeted Optimizations

To bridge this performance gap, we deployed three targeted optimizations:

1. Advanced Inference (TTA)

We upgraded our pipeline to use **Test Time Augmentation**. Instead of a single pass, the model predicted on flipped and rotated versions of the input volume. We then averaged these predictions to smooth out decision boundaries and reduce errors.

2. Input Optimization

We retrained the model using fully **preprocessed and normalized data** rather than raw inputs. This removed the computational bottleneck of on-the-fly processing, allowing us to complete more training epochs within the same time limit.

3. Smart Transfer Learning (Differential Learning Rates)

Our most critical technical optimization was a strategy called **Discriminative Layer-Wise Learning Rates**.

We hypothesized that the **early layers** of the model (the stem and first encoder blocks) had already learned robust, universal features—like edge detection—from the pre-training dataset. Aggressively retraining these layers would cause **'catastrophic forgetting,'** where the model overwrites useful prior knowledge. However, the **deep layers and decoder** needed to adapt quickly to the specific tumor regions in the BraTS dataset.

To solve this, we split the model into two parameter groups:

- **For the Deep Layers:** We assigned a base learning rate of **1e-4** to encourage rapid learning of high-level semantic features.
- **For the Early Layers:** We applied a **dampening factor of 0.01**, reducing their learning rate to **1e-6**.

We managed this using an **AdamW optimizer** paired with a **Linear Warmup and Cosine Annealing scheduler**. This ensured the model could fine-tune its decision heads quickly without destabilizing the robust feature extractors at the input level.

Key Takeaways

Our three-day journey at the ODC AI Hackathon taught us valuable lessons about tackling complex medical imaging challenges under time constraints:

1. **Literature review is essential** — Understanding the state-of-the-art through papers gave us the confidence to choose MedNeXt over Transformers.
2. **Preprocessing is foundational** — A robust data pipeline is essential for stable training.
3. **Transformers aren't always the answer** — In data-scarce medical settings, ConvNets with large kernels can match or exceed Transformer performance.
4. **Inductive bias matters** — MedNeXt's inherent spatial priors made it trainable on just 60 samples.
5. **Balance complexity with risk** — Testing a spectrum of architectures protects against overfitting.
6. **Optimize strategically** — Techniques like TTA, data optimization, and differential learning rates can significantly boost performance.
7. **Time management is critical** — Working within platform constraints (like Kaggle's 30-hour limit) requires careful planning.

References

1. Ankomah, C.T., et al. (2025). *Brain Tumor Segmentation in the Sub-Saharan African Population Using Segmentation-Aware Data Augmentation and Model Ensembling*. arXiv:2510.03568v2
2. Hashmi, S., et al. (2024). *Optimizing Brain Tumor Segmentation with MedNeXt: BraTS 2024 SSA and Pediatrics*. MBZUAI. arXiv:2411.15872v2
3. Roy, S., et al. (2024). *MedNeXt: Transformer-driven Scaling of ConvNets for Medical Image Segmentation.* DKFZ. arXiv:2303.09975v5

This hackathon was an incredible learning experience in brain tumor segmentation, and we're proud of the progress we made in just four days!