FNO-Diffusion: Fourier Neural Operator-Enhanced Diffusion Models for Brain-Tumor MRI Segmentation

Shuzhen Zhang, Zhiwei Tan, Songran Wang, Cindy Wang
University of Washington
{shuzhenz, zhiwei24, songranw,cinnddyy}@uw.edu

Abstract

Project Scope

The goal of our project was to evaluate and improve brain tumor segmentation on the BraTS 2021 dataset using a combination of Fourier Neural Operators (FNOs) and diffusion probabilistic models. We replicated two existing approaches—FNO segmentation and Diffusion with U-shape segmentation—and further proposed a hybrid architecture that integrates FNO into both the diffusion and supervised branches. Our hypothesis was that combining the global modeling ability of FNO with the generative refinement of diffusion would outperform each model individually.

Methodology

We used 2D slices derived from the BraTS 2021 dataset and implemented all models in PyTorch. For the baseline methods, we reimplemented FNO from scratch and adapted a publicly available diffusion-U-Net codebase. Our proposed FNO-Diffusion model replaced the UNet modules with spectral FNO blocks. All models were trained on an NVIDIA Tesla V100 GPU. Training time ranged from 2–8 hours depending on the model, with early stopping and mixed-precision training applied.

Results

The standalone Diffusion with U-shape model achieved the highest DICE score (69.44%), outperforming both the FNO baseline (65.13%) and the hybrid FNO-Diffusion model (57.03%). Contrary to our hypothesis, the combined model underperformed. We found that although FNO captures global features well, it struggles with fine-grained local structures essential for medical image segmentation.

What was Easy

Adapting and training the diffusion model with a U-shaped backbone was relatively straightforward. The modular structure of the code and clear separation of components enabled easy modification and debugging, allowing smooth integration with new architectures like FNO.

What was Difficult

The MedSegDiff-V2 model was difficult to adapt for multiclass segmentation. Despite working well for binary tasks, extending the architecture and loss functions to handle BraTS's four-class setup led to unstable training. Additionally, integrating FNO into the diffusion pipeline introduced complexity due to mismatched tensor dimensions and the need for custom timestep embeddings.

Github links

Diffusion with U-shape model, FNO and Diffusion with FNO model

CSE 599 , Project Report.

1 Introduction

Accurate delineation of brain tumor boundaries on MRI is critical for diagnosis, surgical planning, and treatment monitoring, yet today it still relies heavily on time consuming, error prone manual annotation by radiologists. We plan to address this bottleneck by building a system that automatically segments tumor regions in BraTS MRI images. We will explore deep-learning approaches to aim to identify a practical solution that can improve efficiency in clinical workflows and improve patient care.

A prominent approach to image segmentation, especially in the biomedical field, is the UNet family. We see variations such as the nnUNet (1) and TransUNet (2) that have recently improved global-context modeling. Diffusion probabilistic models have shown promising results compared to the UNet in regards to image segmentation (3). Fourier Neural Operators (FNOs) have excelled in learning PDE solution maps, and architectures such as FNOSeg3D have shown great results in medical image segmentation (4). Our proposal is to integrate FNOs with diffusion based models to exploit both frequency-domain representations and generative refinements.

In this work, we will replicate and adapt the Fourier Neural Operator (FNO) and diffusion with U-shape model for the task of brain tumor segmentation. Specifically, we will implement FNO to learn the mapping from MRI slices to segmentation masks in the frequency domain, and independently train a diffusion with U-shape model to generate pixel-wise tumor masks. In addition, we will explore the integration of a Fourier Neural Operator with a diffusion-based generative framework to segment different parts of the tumor.

2 Scope of the Project

Goal: evaluate and enhance the segmentation performance of brain tumors in BraTS MRI datasets by using both Fourier Neural Operators (FNOs) and diffusion with U-shape models.

2.1 Addressed Claims/Hypothesis from the Original Paper

1. Replicate the Fourier Neural Operator (FNO) model (5) that performs segmentation of brain tumors and calculate the DICE coefficient for multiclass
2. Replicate the diffusion with U-shape model (6) that performs segmentation of brain tumors and calculate the DICE coefficient for multiclass
3. The integrated model exceeds the results obtained by separately using the FNO model and diffusion with U-shape model for brain tumor segmentation

3 Methodology

3.1 Dataset

We use the **BraTS 2021 dataset**(7), which includes **1251** 3D MRI samples, each corresponding to a patient. For each patient, the dataset has 4 MRI modalities with 1 segmentation mask. Each MRI volume has a dimension of **240×240×155** (Height × Width × Depth). The 4 modalities are:

T1	Standard structural MRI, useful for viewing anatomical details
T1Gd	Post-contrast T1-weighted, highlighting active tumor regions with blood-brain barrier breakdown
T2	Sensitive to edema and fluid content, useful for assessing tumor extent
FLAIR	Fluid-attenuated inversion recovery, suppresses CSF signals to highlight peritumoral edema

Table 1: Imaging Data Description

The segmentation mask uses 4 class labels shown in figure 1 and table 2.:

3.2 FNO model background

The 2D FNO learns a mapping $u : \mathbb{R}^{H\times W} \to \mathbb{R}^{H\times W}$ in the frequency domain by applying Fourier transforms within each layer to capture global spatial dependencies (5).

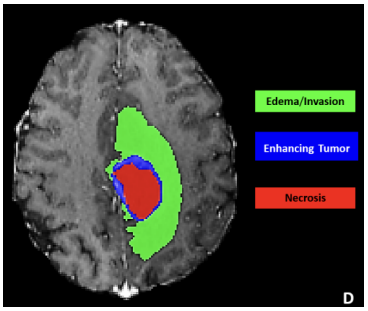


Figure 1: Class Labels (7)

Table 2: Class Labels

0	Background (Gray area)
1	Necrotic tumor core (NCR, Red area)
2	Peritumoral edema (ED, Green area)
4	Enhancing tumor (ET, Blue area)

Forward Mapping with Fourier Transforms

Given an input image $x \in \mathbb{R}^{C\times H\times W}$, and let $\mathcal{F}$ denote the Fourier transform of a function $f : D \to \mathbb{R}^{d_v}$ and $\mathcal{F}^{-1}$ its inverse, the FNO applies a 2D Fast Fourier Transform (FFT) to obtain frequency coefficients:

$$\hat{x} = \mathcal{F}[x]$$

The high-frequency components are truncated by selecting only the lowest $k_1 \times k_2$ Fourier modes. A learnable complex-valued weight tensor $W \in \mathbb{C}^{C_{\text{in}}\times C_{\text{out}}\times k_1\times k_2}$ is then applied:

$$\hat{y}_{b,o,k_1,k_2} = \sum_{i=1}^{C_{\text{in}}} \hat{x}_{b,i,k_1,k_2} \cdot W_{i,o,k_1,k_2}$$

After spectral filtering, the inverse FFT reconstructs the output in spatial domain:

$$y = \mathcal{F}^{-1}[\hat{y}]$$

This operation is embedded within each FNO layer, which includes additional pointwise (1×1) convolutions and activation functions (ReLU). The model follows a lift → spectral layers → projection pipeline.

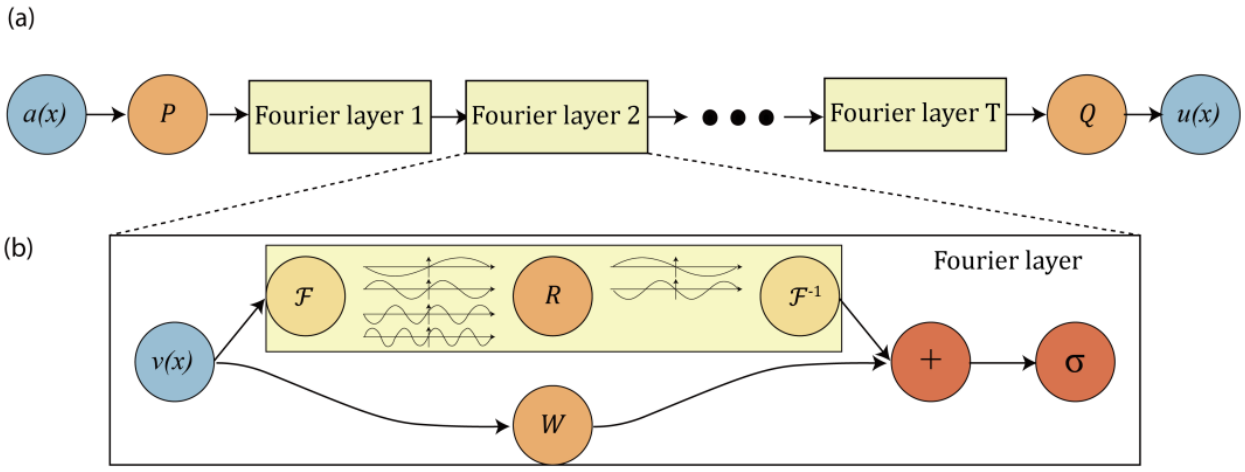


Figure 2: The model architecture of the FNO model (5)

3.3 Diffusion probabilistic model background

Diffusion models are generative models that learn to turn pure noise into structured data (images, audio, 3-D shapes, text embeddings, etc.) by reversing a gradual noising process.

3.3.1 Forward (diffusion) process

At each iteration of the forward process, Gaussian noise is added (3)

$$q(x_t|x_{t-1}) = N(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I_{n\times n})$$

We fix β_t as a constant and it is a variance schedule that defines how much noise is added. Thus, the forward process, defined by the approximate posterior q, has no learnable parameters. The forward process supports sampling at an arbitrary timestamp t, so after reparameterization, we obtain:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{(1-\bar{\alpha})}\epsilon, \epsilon \sim N(0, I_{n\times n})$$

with

$$\alpha_t = 1 - \beta_t, \bar{\alpha}_t = \prod_{s=0}^{t} a_s$$

3.3.2 Reverse (denoising) process and training objective

The reverse process steps are performed by taking small Gaussian steps, and it is defined as a Markov chain with learned Gaussian transitions starting from $p_\theta(x_T) = N(x_T; 0, I_{n\times n})$:

$$p_\theta(x_{t-1}|x_t) = N(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

after reparameterization:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)) + \sigma_t z$$

Finally, the term that will be minimized in the training process is

$$E_{x_0,\epsilon,t}[||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t, I)||^2] \quad (1)$$

where $\epsilon \sim N(0, I_{n\times n})$ and I is the input image.

3.4 Diffusion with U-shape Model: Dual-Path Segmentation with Diffusion-Guided Supervision

We use a dual-path segmentation framework that integrates a diffusion-based denoising branch and a supervised prediction branch to enhance medical image segmentation accuracy. The overall architecture is illustrated in Fig.3.

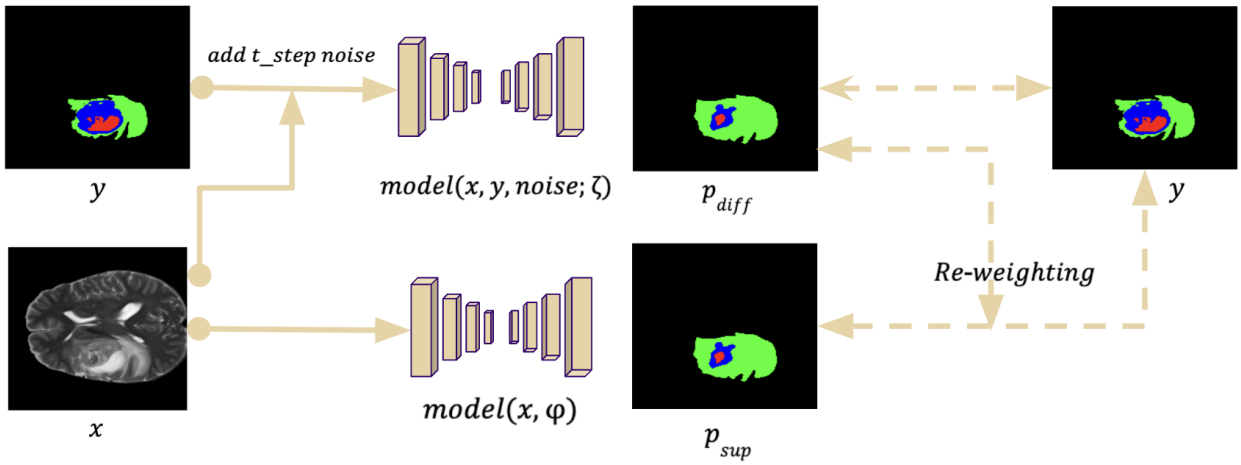


Figure 3: The model architecture of Dual-Path Segmentation with Diffusion-Guided Supervision

Diffusion branch: Using only the forward process mentioned in section 3.3.1. Given an input image x and its corresponding segmentation label y, we first inject noise into the label y at a predefined timestep t to simulate a corrupted version of the segmentation mask. This noisy label, together with the original image x and the timestep noise, is fed into a diffusion model parameterized by ζ denoted as:

$$p_{diff} = \text{model}(x, y, \text{noise}; \zeta)$$

The model aims to denoise the corrupted label conditioned on the input image, thereby producing a pseudo label p_{diff} that is guided by the structural context in x.

Supervised prediction branch: Simultaneously, we employ a supervised segmentation network parameterized by ϕ which directly takes the input image x and predicts the segmentation mask:

$$p_{sup} = \text{model}(x; \phi)$$

Re-weighted supervision mechanism: To leverage the complementary strengths of the diffusion-based pseudo labels and the direct supervised predictions, we introduce a re-weighting strategy. Both p_{diff} and p_{sup} are compared against the ground-truth label y, and a dynamic weighting mechanism is used to balance their contributions during training. This mechanism encourages the model to emphasize more reliable predictions, mitigating the noise in pseudo labels while preserving the benefits of diffusion-guided refinement.

3.5 Diffusion with FNO Model: Dual-Path Segmentation with Diffusion-Guided Supervision

To further explore the potential of operator-based architectures in segmentation, we propose an enhanced variant of our framework by replacing both core components with Fourier Neural Operators (FNOs). Specifically, in the diffusion-guided pseudo labeling branch, we substitute the original UNet-based diffusion model with an FNO-based model, denoted as:

$$p_{diff}^{fno} = \text{FNO}(x, y, \text{noise}; \zeta')$$

This modification allows the diffusion process to better capture global spatial dependencies through spectral representations, potentially yielding higher-quality pseudo labels.

In the supervised prediction branch, we similarly replace the conventional segmentation network with an FNO model parameterized by ϕ', which directly outputs the predicted segmentation mask:

$$p_{sup}^{fno} = \text{FNO}(x; \phi')$$

These replacements are motivated by the global receptive field and resolution-agnostic capabilities of FNOs, which can complement both local denoising in the diffusion branch and spatially coherent prediction in the supervised path. The re-weighted supervision mechanism remains unchanged, dynamically balancing contributions from p_{diff}^{fno} and p_{sup}^{fno} during training.

3.6 Experimental Setup

1. Hardware: 1 x NVIDIA Tesla V100 GPU
2. Training Strategy: At each epoch, both training and validation losses are recorded to monitor convergence.
3. We use 70% of the samples for training, 20% for validation, and 10% for testing.

3.6.1 Dataset and Preprocessing

We used the dataset put forward in section 3.1. Each MRI volume (240×240×155) is reshaped to slice-wise 2D images. We then apply random horizontal and vertical flips as data augmentation. The labels are converted to one-hot format with three semantic classes corresponding to the tumor subregions.

3.7 Evaluation Metric

We will be using the DICE coefficient to evaluate the BraTS dataset.

$$DICE(y_i, \hat{y}_i) = \frac{2TP(y_i, \hat{y}_i)}{2TP(y_i, \hat{y}_i) + FN(y_i, \hat{y}_i) + FP(y_i, \hat{y}_i)}$$

3.8 FNO training and experiment

3.8.1 Model Architecture

The model first lifts the input features to a higher-dimensional space using a 1×1 convolution. Each output branch consists of three repeated FNO blocks, where spectral convolution is applied using the lowest $k_1 = k_2 = 10$ Fourier modes, followed by inverse FFT and a residual bypass connection via 1×1 convolution. The final projection layer maps the feature space back to a single-channel output per class.

3.8.2 Training Configuration

The training is conducted using the Adam optimizer with a learning rate of 3×10^{-4} and a batch size of 8 for 50 epochs. The hybrid loss function combines Dice loss and Cross-Entropy loss, where the total loss is given by:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{Dice}} + \lambda \cdot \mathcal{L}_{\text{CE}},$$

with $\lambda = 0.5$ balancing the two terms. The GELU activation function is used throughout the network. Detailed hyperparameter settings are listed in Table 3.

Parameter	**Value**
Input Channels	4 (4 modalities: t1, t1ce, t2, flair)
Output Channels	4 (Background, NCR, ED, ET in one-hot format)
Fourier Modes	$k_1 = k_2 = 10$
Width	16
Activation	GELU
Batch Size	8
Epochs	50
Loss function	DICE loss + Cross entropy loss
Number of repeated FNO blocks in each output branch	3
Optimizer	Adam (lr $= 3 \times 10^{-4}$)

Table 3: FNO Hyperparameters

3.9 Diffusion with U-shape Model training and experiment

3.9.1 Model Architecture

The model uses a U-shaped encoder-decoder backbone tailored for diffusion-based segmentation. The input image and an associated timestep are first embedded and fused using convolutional and linear layers. The encoder consists of repeated convolutional blocks with normalization (batch, group, or instance norm) and dropout, with downsampling performed via strided convolutions. As features are downsampled, skip connections store intermediate outputs at each encoder stage.

In the bottleneck, additional convolutional processing is applied. The decoder mirrors the encoder, successively upsampling feature maps using transposed convolutions and fusing them with corresponding encoder features via skip connections. Temporal embeddings are injected at multiple stages in both the encoder and decoder to condition the network on the diffusion timestep.

Two parallel decoder branches are implemented for different prediction heads, and a separate denoising diffusion model branch provides consistency and regularization. The final layer is a 1×1 convolution mapping the decoder output to the required number of segmentation classes. Swish $(x \cdot \text{sigmoid}(x))$ and ReLU activations are used throughout, and normalization and dropout are applied after each convolutional unit.

3.9.2 Training Configuration

The training is conducted using the Adam optimizer with a learning rate of 1×10^{-2} and a batch size of 32 for a maximum of 300 epochs. An early stop of 50 epochs is in place. The hybrid loss function combines Dice loss and Cross-Entropy loss, where the total loss is given by:

$$\mathcal{L}_{\text{total}} = \phi \cdot \mathcal{L}_{\text{Dice}} + \lambda \cdot \mathcal{L}_{\text{CE}},$$

The loss function does not use fixed weights, it uses per-class weights that adapt dynamically based on the learning progress of each class, reflecting how difficult each class currently is for the model. Detailed hyperparameter settings are listed in Table 4.

Parameter	**Value**
Optimizer	Adam($\text{lr} = 1 \times 10^{-2}$)
Batch size	32
EMA Weight (α)	0.99
Loss function	DICE loss + Cross entropy loss
Weight of unsupervised loss	10

Table 4: Diffusion with U-shape model Hyperparameters

3.10 Diffusion with FNO Model training and experiment

3.10.1 Model Architecture

The model adopts a diffusion-based encoder-decoder structure with integrated spectral and convolutional components. Input images and noisy segmentation masks are first fused with a sinusoidal timestep embedding. The encoder follows a standard hierarchical downsampling design with normalization, dropout, and temporal conditioning at each stage.

Two decoders are used: a convolutional decoder Decoder_θ with skip connections, and a Fourier Neural Operator (FNO) decoder Decoder_ψ that performs spectral convolution over spatial dimensions to capture global context. A separate denoising model D_ξ reconstructs clean masks from noise, also conditioned on timestep and image. The final segmentation logits are obtained via a 1×1 convolution. This hybrid design allows the model to leverage both local spatial details and long-range dependencies under the diffusion framework.

3.10.2 Training Configuration

The training is conducted using the SGD optimizer with a base learning rate of 1×10^{-3}, momentum of 0.9, and a batch size of 32 for a maximum of 300 epochs. An early stopping patience of 50 epochs is applied to prevent overfitting. Mixed-precision training is enabled to accelerate computation.

The loss function is the same as that in 3.9.2

Additionally, an Exponential Moving Average (EMA) mechanism is applied to the main decoder to stabilize predictions using both the denoising branch and the FNO-based decoder. All experiments are performed using a GPU with CUDA acceleration. The complete set of training hyperparameters is summarized in Table 5.

Parameter	**Value**
Optimizer	SGD (momentum = 0.9, weight decay = 3e-5)
Learning Rate	0.001
Batch Size	32
EMA Weight (α)	0.99
Diffusion Timesteps	1000 (sampling uses 10)
Loss Function	DICE loss + Cross entropy loss
FNO Modes	$[16, 16]$
FNO Width	32
FNO Blocks per Channel	3
Time Embedding Dim	512
Dropout Rate	0.5

Table 5: FNO Diffusion Training Hyperparameters

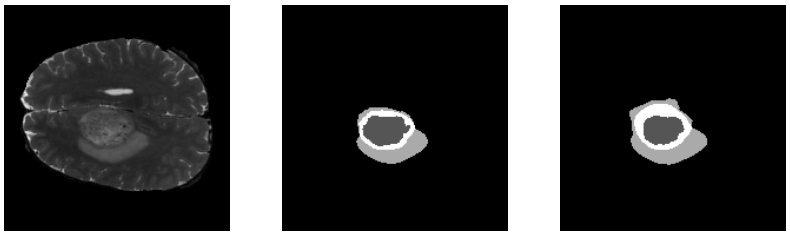
4 Results and Summary

4.1 Result 1: FNO

(a) Input Image (b) Ground Truth (c) FNO Prediction

Figure 4: Results for the FNO-based segmentation model. (a) shows the original MRI slice, (b) the ground-truth segmentation mask, and (c) the prediction output from the FNO model. Achieving a DICE score of 65.13%.

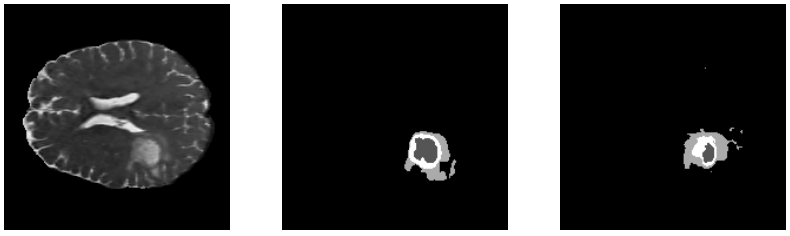
4.2 Result 2: Diffusion with U-shape Model



(a) Input Image (b) Ground Truth (c) Diffusion-UNet Prediction

Figure 5: Results for the diffusion model using a UNet backbone. (a) shows the input MRI slice, (b) the ground-truth segmentation mask, and (c) the output predicted by the diffusion-based model. Achieving a DICE score of 69.44%.

4.3 Result 3: Diffusion with FNO Model



(a) Input Image (b) Ground Truth (c) FNO-Diffusion Prediction

Figure 6: Results for the diffusion+FNO model. (a) shows the input MRI slice, (b) the ground-truth segmentation mask, and (c) the output predicted by the FNO-Diffusion model. Achieving a DICE score of 57.03%.

5 Discussion

Our results showed that the combined Diffusion with FNO model performed worse than using FNO alone or Diffusion with U-shape model.

One likely reason is that FNO lacks attention mechanisms, which are important for capturing local features and refining boundaries—critical in segmentation tasks. In contrast, the U-shape model inherently models both local and global context via skip connections and convolutional layers.

5.1 What was Easy

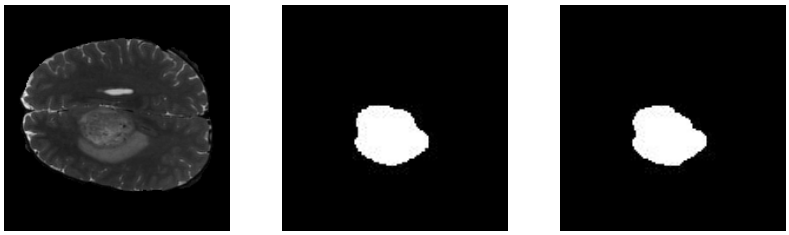
Compared to MedSegDiff-V2 (8), we found that working with the diffusion-based segmentation pipeline using a U-shaped architecture was relatively easier. The overall codebase for this model was more modular and easier to follow, with clear naming conventions and explicit separation between the forward diffusion process, the denoising model, and the loss computation. This made it easier to isolate components for modification or debugging.

While the diffusion mechanism itself still involves non-trivial concepts (e.g., timestep embedding and noise scheduling), the clarity of the U-shaped implementation allowed us to experiment with model replacements—such as swapping in Fourier Neural Operators—with more confidence than we had when trying to extend the MedSegDiff-V2 code.

5.2 What was Difficult

One of the major challenges we encountered was attempting to reproduce the results of MedSegDiff-V2 (8). The authors provided code, and the original model was designed for a wide range of medical segmentation tasks. The binary segmentation setting (e.g., tumor vs. non-tumor) was successfully implemented and produced reasonable results 7. However, when extending the model to the multiclass segmentation task, the training became unstable, and the segmentation quality degraded significantly. Despite several attempts to tune the architecture and loss functions, we were unable to obtain satisfactory multiclass results.

We originally planned to use MedSegDiff-V2 as a baseline and then replace its UNet backbone with a Fourier Neural Operator (FNO). Unfortunately, due to the poor performance in the multiclass setting and the complexity of integrating timestep embeddings with FNO blocks, this plan was not successful.



(a) Input Image (b) Ground Truth (Binary) (c) MedSegDiff-V2 Prediction

Figure 7: Binary segmentation result using MedSegDiff-V2 on BraTS. While reasonable performance was achieved in the binary setting, the model failed to generalize to multiclass segmentation.

5.3 Recommendations for Future Work

If we were to start this project again, we would prioritize establishing stable baselines. We recommend future teams begin with a UNet-based diffusion baseline, such as the one we adopted in our dual-path framework, and ensure it achieves consistent performance on binary or multiclass segmentation tasks before extending it.

For building on this project, we suggest that:

- Start working with 2D slices. Once the pipeline is stable, extension to 3D can be considered.
- Carefully balance the loss functions: When using both supervised and diffusion-based losses, dynamic weighting strategies can help stabilize training. Ablation studies on weighting schemes are recommended.

References

[1] Fabian Isensee, Paul F. Jaeger, Simon A. A. Kohl, Jens Petersen, and Klaus H. Maier-Hein. nnu-net: a self-configuring method for deep learning-based biomedical image segmentation. *Nature Methods*, 18(2):203–211, Feb 2021. doi:10.1038/s41592-020-01008-z.

[2] Jieneng Chen, Yongyi Lu, Qihang Yu, Xiangde Luo, Ehsan Adeli, Yan Wang, Le Lu, Alan L. Yuille, and Yuyin Zhou. Transunet: Transformers make strong encoders for medical image segmentation, 2021. URL: https://arxiv.org/abs/2102.04306, arXiv:2102.04306.

[3] Tomer Amit, Tal Shaharbany, Eliya Nachmani, and Lior Wolf. Segdiff: Image segmentation with diffusion probabilistic models. *arXiv preprint arXiv:2112.00390*, 2021. URL: https://arxiv.org/abs/2112.00390.

[4] Ken C. L. Wong, Hongzhi Wang, and Tanveer Syeda-Mahmood. Fnoseg3d: Resolution-robust 3d image segmentation with fourier neural operator. In *2023 IEEE 20th International Symposium on Biomedical Imaging (ISBI)*, page 1–5. IEEE, April 2023. URL: http://dx.doi.org/10.1109/ISBI53787.2023.10230586, doi:10.1109/isbi53787.2023.10230586.

[5] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021. URL: https://arxiv.org/abs/2010.08895, arXiv:2010.08895.

[6] Haonan Wang and Xiaomeng Li. Towards generic semi-supervised framework for volumetric medical image segmentation, 2023. URL: https://arxiv.org/abs/2310.11320, arXiv:2310.11320.

[7] Ujjwal Baid, Satyam Ghodasara, Suyash Mohan, Michel Bilello, Evan Calabrese, Errol Colak, Keyvan Farahani, Jayashree Kalpathy-Cramer, Felipe C. Kitamura, Sarthak Pati, Luciano M. Prevedello, Jeffrey D. Rudie, Chiharu Sako, Russell T. Shinohara, Timothy Bergquist, Rong Chai, James Eddy, Julia Elliott, Walter Reade, Thomas Schaffter, Thomas Yu, Jiaxin Zheng, Ahmed W. Moawad, Luiz Otavio Coelho, Olivia McDonnell, Elka Miller, Fanny E. Moron, Mark C. Oswood, Robert Y. Shih, Loizos Siakallis, Yulia Bronstein, James R. Mason, Anthony F. Miller, Gagandeep Choudhary, Aanchal Agarwal, Cristina H. Besada, Jamal J. Derakhshan, Mariana C. Diogo, Daniel D. Do-Dai, Luciano Farage, John L. Go, Mohiuddin Hadi, Virginia B. Hill, Michael Iv, David Joyner, Christie Lincoln, Eyal Lotan, Asako Miyakoshi, Mariana Sanchez-Montano, Jaya Nath, Xuan V. Nguyen, Manal Nicolas-Jilwan, Johanna Ortiz Jimenez, Kerem Ozturk, Bojan D. Petrovic, Chintan Shah, Lubdha M. Shah, Manas Sharma, Onur Simsek, Achint K. Singh, Salil Soman, Volodymyr Statsevych, Brent D. Weinberg, Robert J. Young, Ichiro Ikuta, Amit K. Agarwal, Sword C. Cambron, Richard Silbergleit, Alexandru Dusoi, Alida A. Postma, Laurent Letourneau-Guillon, Gloria J. Guzman Perez-Carrillo, Atin Saha, Neetu Soni, Greg Zaharchuk, Vahe M. Zohrabian, Yingming Chen, Milos M. Cekic, Akm Rahman, Juan E. Small, Varun Sethi, Christos Davatzikos, John Mongan, Christopher Hess, Soonmee Cha, Javier Villanueva-Meyer, John B. Freymann, Justin S. Kirby, Benedikt Wiestler, Priscila Crivellaro, Rivka R. Colen, Aikaterini Kotrotsou, Daniel Marcus, Mikhail Milchenko, Arash Nazeri, Hassan Fathallah-Shaykh, Roland Wiest, Andras Jakab, Marc-Andre Weber, Abhishek Mahajan, Bjoern Menze, Adam E. Flanders, and Spyridon Bakas. The rsna-asnr-miccai brats 2021 benchmark on brain tumor segmentation and radiogenomic classification, 2021. URL: https://arxiv.org/abs/2107.02314, arXiv:2107.02314.

[8] Junde Wu, Wei Ji, Huazhu Fu, Min Xu, Yueming Jin, and Yanwu Xu. Medsegdiff-v2: Diffusion-based medical image segmentation with transformer. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(6):6030–6038, March 2024. URL: https://ojs.aaai.org/index.php/AAAI/article/view/28418, doi:10.1609/aaai.v38i6.28418.


```python
import torch
import torch.nn as nn
from torch.optim import Optimizer
from torch.utils.data import DataLoader, TensorDataset
from tqdm.auto import tqdm
from accelerate import Accelerator

import optuna
from FNO_torch.Diffusion.diffusion_basic import Diffusion

accelerator = Accelerator()

def train_epoch(model: nn.Module,
                train_loader: DataLoader,
```

```
                optimizer: Optimizer,
                device: torch.device,
                *,
                x_name: str | None = None,
                y_name: str | None = None) -> float:
    model.train()
    running = 0.0
    total = 0
    pbar = tqdm(train_loader, desc='Train', position=1, leave=False, dynamic_ncols
    =True)
    for batch in pbar:
        if isinstance(batch, dict):
            if x_name is None or y_name is None:
                raise ValueError("When batches are dictionaries, x_name and y_name
     must be provided.")
            xb = batch[x_name].to(device)
            yb = batch[y_name].to(device)
        elif isinstance(batch, (list, tuple)):
            xb = batch[0].to(device)
            yb = batch[1].to(device)
        else:
            raise TypeError(f"Unsupported batch type: {type(batch)}")
        optimizer.zero_grad()
        base_model = model.module if isinstance(model, nn.DataParallel) else model
        loss = base_model.cal_loss(xb, yb)
        accelerator.backward(loss)
        optimizer.step()
        running += loss.item() * xb.size(0)
        total += xb.size(0)
    return running / total

def valid_epoch(model: nn.Module,
                test_loader: DataLoader,
                device: torch.device,
                *,
                x_name: str | None = None,
                y_name: str | None = None) -> float:
    model.eval()
    val_running = 0.0
    total = 0
    with torch.no_grad():
        pbar = tqdm(test_loader, desc='Valid', position=1, leave=False,
    dynamic_ncols=True)
        for batch in pbar:
            if isinstance(batch, dict):
                if x_name is None or y_name is None:
                    raise ValueError("When batches are dictionaries, x_name and
    y_name must be provided.")
                xb = batch[x_name].to(device)
                yb = batch[y_name].to(device)
            elif isinstance(batch, (list, tuple)):
                xb = batch[0].to(device)
                yb = batch[1].to(device)
            else:
                raise TypeError(f"Unsupported batch type: {type(batch)}")
            base_model = model.module if isinstance(model, nn.DataParallel) else
    model
            loss = base_model.cal_loss(xb, yb)
            val_running += loss.item() * xb.size(0)
            total += xb.size(0)
    return val_running / total

def train_model(model: nn.Module,
                train_loader: DataLoader,
                test_loader: DataLoader,
```

```
                optimizer: Optimizer,
                epochs: int,
                device: torch.device,
                x_name: str = None,
                y_name: str = None) -> dict:
    # Prepare model, optimizer, and data loaders for multi-GPU
    model, optimizer, train_loader, test_loader = accelerator.prepare(
        model, optimizer, train_loader, test_loader
    )
    device = accelerator.device
    history = {'train_loss': [], 'val_loss': []}
    pbar = tqdm(range(epochs), desc='Epoch', unit='epoch', leave=True,
dynamic_ncols=True, position=0)
    for epoch in pbar:
        train_loss = train_epoch(model, train_loader, optimizer, device, x_name=
x_name, y_name=y_name)
        val_loss = valid_epoch(model, test_loader, device, x_name=x_name, y_name=
y_name)
        history['train_loss'].append(train_loss)
        history['val_loss'].append(val_loss)
        pbar.set_postfix(train_loss=train_loss, val_loss=val_loss)
    return history
```