Take-Home Assignment

Medical Image Segmentation + Continuous MLOps Tracking

Dataset: Medical Image Segmentation (Kaggle)
https://www.kaggle.com/datasets/modaresimr/medical-image-segmentation

Time expectation: ~6–10 hours
Submission: GitHub repo link (preferred) or zipped project

Goal

Train a medical image segmentation model and build an MLOps pipeline that makes it easy to **run experiments repeatedly**, **track results**, and **continuously improve model performance** over time.

Part 1 - Train a Segmentation Model

Train a segmentation model using the dataset above.

Requirements

- Use **Dice score** as the primary metric (IoU as secondary).
- Implement a clean **train/validation split** (explain your split strategy; avoid leakage if any grouping exists).
- Save the **best checkpoint** based on validation Dice.

Expected baseline

A standard UNet (or similar) trained with a reasonable loss (Dice/BCE combo is fine). You are not expected to achieve high-accuracy in this case but you need to provide a plan to improve accuracy.

Part 2 - Design & Implement an MLOps Pipeline for Continuous Improvement

Build a local, lightweight pipeline that supports iterative experimentation and tracking.

Must-have features

1) Config-driven experiments

All runs should be controlled by config (YAML/JSON/TOML), including:

- Dataset path
- Model choice and hyperparameters
- Augmentations
- Training params (epochs, LR, batch size)
- Output paths

Example:

```
python train.py --config configs/exp_001.yaml
python eval.py  --config configs/exp_001.yaml --ckpt <path>
```

2) Experiment tracking

Use **MLflow** (recommended) or an equivalent open-source tracker. Each run must log:

- Params (model, loss, LR, augmentations, etc.)
- Metrics (train/val Dice + IoU per epoch)
- Artifacts (best checkpoint, plots, sample predictions)

3) Reproducibility

- Fixed random seeds
- Deterministic settings documented (where applicable)
- Same config → same run behavior (within reason for GPU nondeterminism)

4) “Continuous improvement” workflow

Implement a simple mechanism that encourages iteration. Any one of these is acceptable:

- A `run_experiments.py` that executes multiple configs and compares results
- A `model_registry` concept (even a simple folder convention) where “best model so far” is tracked
- Automatic promotion of the best model based on a metric threshold
- A comparison report (markdown or JSON) summarizing experiments and deltas

---

Deliverables

Your repo should include:

- `train.py`, `eval.py`, `infer.py` (or equivalent)
- `configs/` (at least 2 experiment configs showing iteration)
- MLflow tracking (instructions to run locally; don't commit huge artifacts)
- Saved best checkpoint
- Sample predictions on a few validation images
- `README.md` with clear instructions and experiment summary

Recommended repo structure

```
.
├── configs/
├── src/
│   ├── data/
│   ├── models/
│   ├── training/
│   ├── utils/
├── train.py
├── eval.py
├── infer.py
├── run_experiments.py (optional but good)
├── requirements.txt
├── .github/workflows/ci.yml (optional but strong)
└── README.md
```

README expectations

Include:

- Diagrams
- Setup instructions
- How to run train/eval/infer
- Experiment results table like:

Experiment	Key change	Val Dice	Val IoU
exp_000	baseline UNet + BCE/Dice	0.xx	0.xx

exp_001	+ augmentations + LR schedule	0.xx	0.xx
exp_002	loss tweak / encoder / etc.	0.xx	0.xx

Evaluation criteria

Segmentation quality (40%)

- Sensible training pipeline
- Correct metrics
- Improvements explained clearly
- No obvious leakage issues

MLOps pipeline quality (60%)

- Config-driven workflow
- Solid tracking (params/metrics/artifacts)
- Clear “continuous improvement” mechanism
- Reproducibility and usability

Notes from Evaluators:

- You are allowed to use controlled support of AI tools.
- Your understanding of the requirement and solution is crucial, as you have to justify the decision taken to implement
- Additionally, you will be evaluated on code quality, documentation, and cost/computation efficiency of the code