A Co-evolving Agentic AI System for medical imaging analysis

Professor: Zhaohan Xi

Research Intern: Shantanu Jaipurkar (MS Information Systems)

Index Page:

Sr.No	Topic	Page.No
1	Initial Plan (10/19/2025)	2

Initial Plan

1) Project Overview (what we're doing now):

We will build a **headless (no UI yet)**, modular pipeline that mirrors the core ideas in TissueLab:

- **Ask → Plan → Execute → Summarize** for medical images.
- Start with two flagship tasks from the paper:
  - **Depth of Invasion (DoI)** estimation on pathology whole-slide images (WSI).
  - **Lymph-node counting & positivity** (object detection + classification on pathology slides).
- All results are saved to disk as **artifacts** (masks/overlays) and **metrics** (JSON), so experiments are reproducible and easy to validate.
- This gives us a correct, testable backbone before we invest in desktop apps or UI.

2) Why this approach (rationale)

- **Focus on correctness first:** Remove UI distractions; verify the medical logic, masks, and measurements work end-to-end.
- **Stable interfaces:** Define a simple **tool contract** now, so we can swap heuristics for ML models later without changing the orchestration.
- **Reproducible science:** Every run writes a report, masks, and logs. This is crucial for validation, comparison, and academic reporting.
- **Privacy by design:** Local processing (no network required) fits clinical constraints and later hospital deployment.

3) Scope of the initial MVP (what's "in" vs "later")

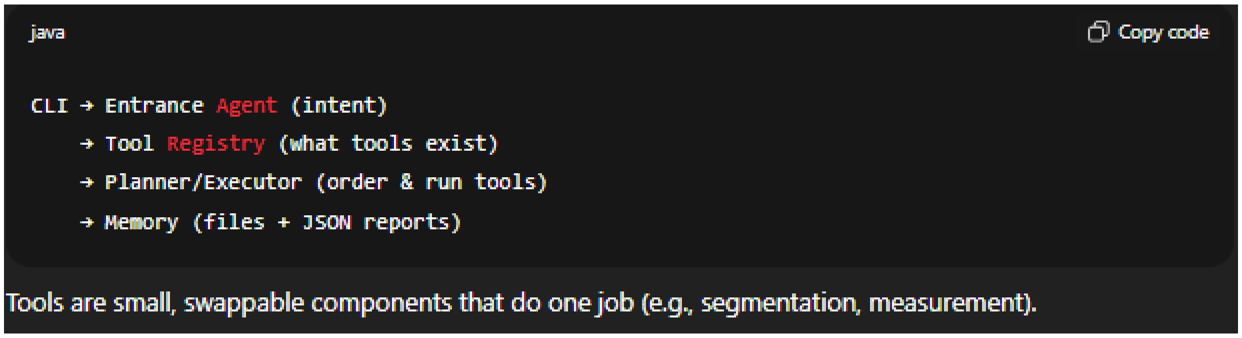
In (now):

- Command-line runner: tissuelab run --image ... --query "..."
- Simple "Entrance Agent" that maps a query to an intent (rule-based).
- A **Tool Registry** (manifest) and pluggable **Tool modules** with a uniform I/O.
- A **Planner/Executor** that builds an ordered list of tools and runs them.
- A **Memory** layer (folders + JSON) storing artifacts and metrics.

Later (once pipeline is solid):

- Real models (PyTorch/MONAI) replacing heuristics.
- Feedback capture and active learning.
- Desktop app (Electron/React) or web UI (Next.js) with a medical image viewer.
- Guideline-aligned diagnosis with online knowledge retrieval (MCP-style).

4) System at a glance (simple)



```
CLI → Entrance Agent (intent)
    → Tool Registry (what tools exist)
    → Planner/Executor (order & run tools)
    → Memory (files + JSON reports)
```

Tools are small, swappable components that do one job (e.g., segmentation, measurement).

5) Pipelines we will implement first

A) Depth of Invasion (DoI) — pathology

Goal: return a numeric depth (µm) and an overlay showing the deepest invasion.

Planned steps:

A) **Preprocess/Tiling** (optional for big WSIs): build a downsampled image/tiles.
B) **Epithelium/Tissue segmentation**: get the reference boundary (baseline).
C) **Tumor segmentation (or patch classifier → tumor mask)**.
D) **Geometry measurement**: compute the deepest distance from epithelium to tumor (px), convert using **pixel size (µm/px)**.
E) **Summarize**: save depth_um, depth_px, and an overlay (doi_overlay.png).

Outputs to expect:

A) report.json with depth_um, depth_px, pixel_size_um.
B) Artifacts: epithelium_mask.tiff, tumor_mask.tiff, doi_overlay.png.

B) B) Lymph-node counting & positivity

Goal: return total nodes, positives, and per-node labels.

Planned steps:

A) **Node detection** (on WSI or large thumbnails): candidate masks/boxes, node patches.
B) **Node positivity classification** (positive/negative; later: macro/micro/ITC per guidelines).
C) **Counting & summary** with overlays marking positives.

Outputs to expect:

A) report.json with node_count, positive_count, per-node scores.
B) Artifacts: nodes_overlay.png, node_patches/….

6) Tools & technologies (now → later)

Now (to get it working)

1. **Language/runtime:** Python 3.11
2. **Core imaging libs:**
    a. numpy, Pillow, scikit-image (thresholding, morphology, connected components)
    b. tifffile (TIFF/WSI access); OpenSlide if needed for pyramids
    c. SimpleITK (optional; DICOM/NIfTI later)
3. **Orchestration (headless):**
    a. CLI script (argparse/typer)
    b. Tool Registry as YAML/JSON
    c. File-based Memory (runs/<id>/…) + report.json
4. **Evaluation:** pandas, matplotlib (plots for MAE/RMSE, correlation), simple CSV logs

Later (to increase accuracy & scale)

- **Deep learning:** PyTorch + MONAI; ONNX for portable inference
- **Model management:** Git LFS for weights; version tags
- **Services:** FastAPI microservices per tool; WebSockets for progress
- **Data stores:** SQLite/Postgres for metadata; Zarr/HDF5 for arrays/tiles
- **UI:** Electron/React (desktop) with OpenSeadragon (WSI viewer) or a web Next.js app
- **Knowledge retrieval:** MCP-style guideline alignment and online criteria mapping

7) Step-by-step execution plan (with deliverables)

- **Phase 0 — Environment Setup & Repository Structure**
- **Phase 1 — Depth of Invasion (DoI) Pipeline Development**
- **Phase 2 — Lymph Node Counting & Classification Pipeline**
- **Phase 3 — Orchestration (Entrance Agent, Planner & Executor)**
- **Phase 4 — Validation & Reporting**
- **Phase 5 — Integration of Deep Learning Models**
- **Phase 6 — Feedback Loop & Co-evolution Mechanism**
- **Phase 7 — Summary Agent & Guideline Alignment (MCP Integration)**
- **Phase 8 — User Interface Development (Streamlit/Electron Prototype)**
- **Phase 9 — Performance Scaling & Deployment Readiness**