SIIM CAIMI26 Conference on Artificial Intelligence in Medical Imaging
OCT 26-27 | University of Pennsylvania, Philadelphia, PA

ITK-SNAP as an Agent-Callable Tool: Expert Human Correction as a Resumable, Audited Pipeline Step

Jilei Hao, MS, University of Pennsylvania; Alison M. Pouch, PhD; Paul A. Yushkevich, PhD

Briefly describe what attendees will see during your demo.
A live demonstration of a single case, in six steps. First, a chat session with an agent is opened. Second, the agent creates an ITK-SNAP workspace from a selected image. Third, it opens that workspace in an interactive ITK-SNAP session. Fourth, it requests an automatic segmentation from our model server and applies the returned structure into the live session, where the result appears in the image views. Fifth, the proposal is corrected manually with the paintbrush. Sixth, the agent queries the audit records and reports what changed at each step, distinguishing the changes it made from those made by the operator. A recorded walkthrough of the same sequence is available as a fallback.

Problem & Motivation: What clinical or operational problem does your MVP address?
Automatic segmentation models are increasingly accurate but remain imperfect, so research and clinical pipelines continue to depend on a person to review and correct their output. That review takes place inside an interactive program: the operator opens the image, paints a correction, saves a file, and moves on. The corrected file persists; little else does. No durable record is kept of what was changed, by how much, or by whom. This has two consequences. First, an automated pipeline has no mechanism for handing a case to a person and receiving a machine-readable result in return. The human step is a discontinuity in the workflow, bridged manually and tracked outside the imaging software. Second, expert corrections are a valuable training signal, and they are routinely discarded: the edit is flattened into a new mask that cannot be distinguished from the model's own output. We set out to determine whether the human step could be made callable — invoked by a pipeline on the cases that require it, suspended, resumed, and answered in a structured form — without requiring any change to how segmentation is actually performed.

What You Built: Describe your MVP, tool, or workflow in plain language
We made ITK-SNAP — a 20-year-old open-source segmentation application with over a million downloads — callable by an AI agent, and arranged for every edit to return a structured record of what changed. It accepts any image format ITK-SNAP can read. It produces an ITK-SNAP workspace containing the segmentation, together with a log of every change made to it. The agent has roughly a dozen commands: list the available models, create a workspace, request an automatic segmentation, write a proposed structure into the segmentation, open a workspace for a person, set label names and colors, and read back the change log. It operates in either of two modes. In the first, it works on the saved workspace with no window open, so a pipeline can prepare cases before anyone is involved, and work can be suspended and resumed because all state resides on disk. In the second, it connects to a running ITK-SNAP over a local channel and applies changes into the open session, where the result appears immediately in the image views and can be corrected in place. The same commands are available and the same record is produced in both modes. Each committed edit returns the same fields: the operation, a timestamp, whether an agent or a person made it, the

number of voxels changed, the bounding box of the change, and the voxel counts per label before and after.

Tech Stack & Development Approach: What tools, frameworks, or methods did you use to build it?

The agent layer is Python 3.10 or later, packaged as a server for the Model Context Protocol — a standard that allows an assistant to call external tools — using the official SDK. Any client implementing the protocol can drive it; in our demonstration the client is Claude Code. Runtime dependencies are limited to requests, numpy, SimpleITK and PyYAML. The model layer is a FastAPI server hosting TotalSegmentator, nnInteractive and SAM2 on a CUDA GPU under PyTorch. The agent communicates with it over HTTP. The application layer is ITK-SNAP itself: C++17, Qt6, ITK and VTK. Two additions were required: a Unix-domain socket carrying newline-delimited JSON-RPC, which allows an external process to communicate with a running window, and the audit record, which resides in the interface-independent logic layer and is covered by a unit test that links against that layer alone. The record is reconstructed at a single commit point from the undo data ITK-SNAP already maintains, so no editing tool required modification. Workspaces are written by ITK-SNAP's own command-line tool rather than composed in Python. Images may use any format ITK-SNAP supports; records are JSON. Development followed a timeboxed sprint using Claude Code, gated on two empirical checkpoints before the design was fixed: serving the segmentation model, and demonstrating that an external process could drive a live window. An adversarial code review, supplied with the requirement but not our reasoning, identified four defects in the provenance field.

Validation & Evidence: Have you done any informal testing or validation? If so, what did you find?

No clinical validation has been performed. What follows is bench testing of the mechanism on a single dataset. The full sequence was executed on a GPU. TotalSegmentator applied to a body CT returned 48 anatomically correct structures. The agent wrote one of them, the left upper lung lobe, into a workspace and read back the record: 1,169,665 voxels changed, bounding box [84,2,0] to [247,189,180], attributed to the agent. A person then opened that workspace and corrected the proposal with the paintbrush. The record returned for the correction carried the same fields with its own values, and differed in the actor field, which read human rather than agent. What was tested is that both producers emit the same schema, not the same measurements. Two results are worth reporting. First, the single-capture-point design held. A unit test exercises the audit engine against both a plain image type and the run-length-encoded type used in production, and links against the logic layer alone, confirming the record carries no dependency on the user interface. Headless tests over the socket assert exact voxel counts. Second, the design failed under realistic use in a way we had not anticipated. The initial read-back command reported only the most recent edit, concealing all but the final correction when a person fixed several structures in one session; we had implicitly assumed one correction per case. Records also did not persist when the window was closed. Both have since been corrected.

Feedback You're Seeking: What specific feedback, help, or collaboration are you hoping to get from the CAIMI26 community?

We are seeking input in four areas. 1) Routing. The system records what changed but does not yet determine which cases require a person. A stability heuristic has been written and unit-tested but is not connected to the pipeline, because we do not know which signal would be trusted in practice. For those who triage automatic segmentations today: what causes you to open a case? 2) Record contents. The record currently carries the operation, timestamp, actor, voxels changed, bounding box, and per-label counts before and after. These fields were chosen by judgment rather than by stated requirement. For retraining or quality auditing, what is missing, and what could be removed? 3) Integration. We would welcome comment from groups running

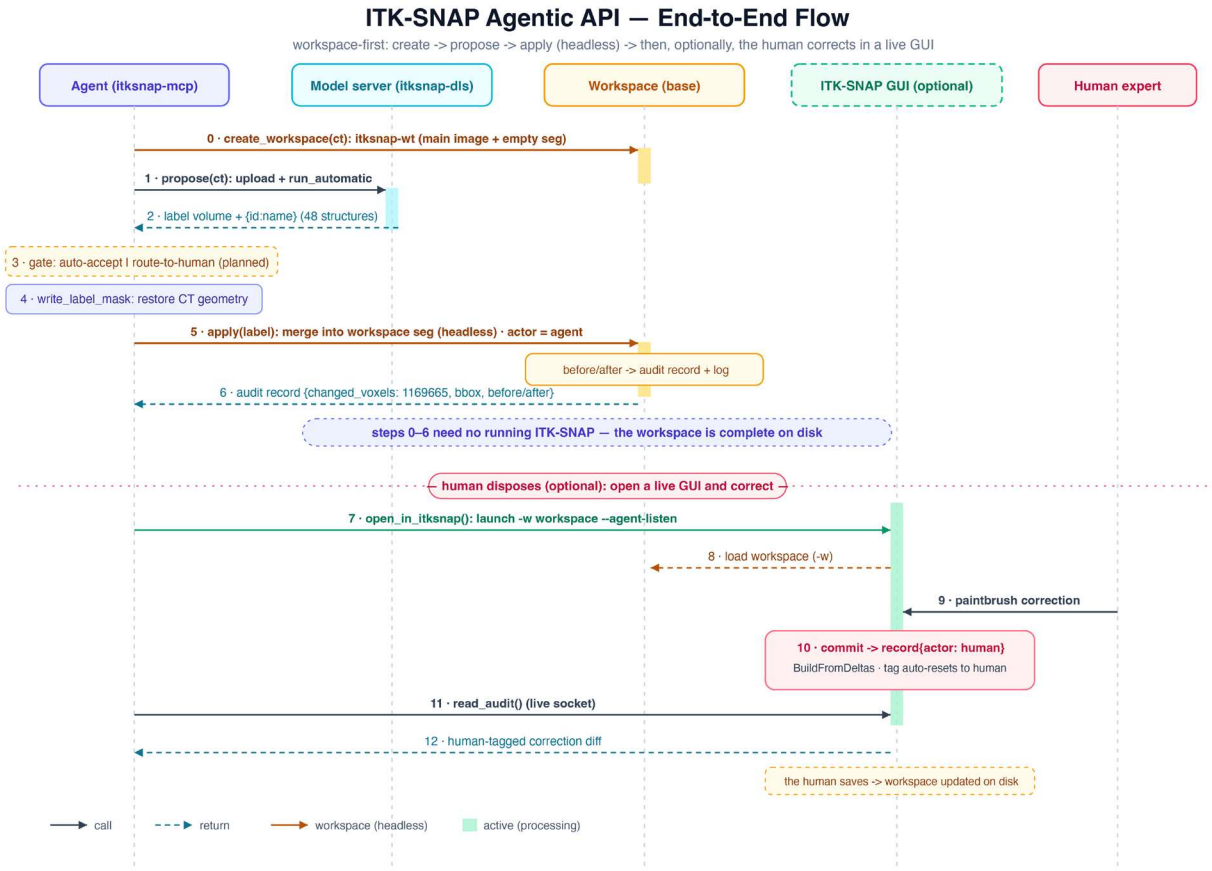
annotation or quality-control workflows at scale on where a callable correction step would sit relative to existing tooling, and whether a workspace file is an appropriate unit of exchange. 4) Clinical collaboration. The correction in our demonstration was made by a developer. The mechanism has been exercised; clinical judgment has not. We are seeking a collaborator who performs this work routinely and can identify failure modes we are not positioned to see. We would also welcome experience with distributing a Qt-based desktop application through the Python package index.

Known Limitations & Honest Failures: What isn't working yet, or what have you already tried that failed?

1) The system cannot yet decide which cases need a person. That decision is the point of the design, but the code for it — a measure of the model's confidence — exists only as a tested function that nothing calls. In the demonstration, a person picks the case. 2) Corrections are not explicitly committed. After correcting a proposal, the operator has no way to accept or reject it: the outcome is implied by whether the file is saved. Accepting and discarding should be deliberate, recorded actions, and are not yet. 3) Installing and configuring the agent server is awkward. It can be registered at either project or user scope, and choosing correctly is a manual step most people will get wrong at least once. This should reduce to a single setup command. 4) The architecture reversed during development. We first built the agent to drive a live ITK-SNAP window, then added the background workspace path and made it the default, because only a saved file can be suspended and resumed. Both are retained, so one operation has two implementations that must stay in agreement.

Potential for Impact: If this MVP were fully developed and deployed, who would benefit and how?

Three groups would benefit. 1) Research groups and clinical services that segment more images than they can inspect. Their correction records would stop being discarded. At present an expert fix is saved as a new mask, indistinguishable from model output. If every correction records who made it, how large it was, and where, a year of routine review becomes a labeled record of the cases the model gets wrong — training data that is expensive to obtain and seldom collected. The same record answers quality questions that are currently difficult to ask: how often the model is corrected, on which structures, and whether that is changing over time. For trainees, an attributable record of corrections is also a teaching record. 2) Groups building agent-driven analysis pipelines. Automated agents are increasingly used to run image analysis. ITK-SNAP can now sit inside such a pipeline as a step the agent calls directly, rather than a manual detour at the end of it. 3) Developers building on ITK-SNAP. The socket interface and the headless commands open the application to outside control. Others can drive its interface, reuse its logic components, or build tools that work alongside it — which previously meant modifying and rebuilding the application itself.



GitHub Repo:

- https://github.com/jilei-hao/itksnap-mcp
- https://github.com/jilei-hao/itksnap