

**Natural-Language-Driven Transfer Function Optimization for  
Volumetric Visualization Using Multi-Agent Large Language  
Models**

## Abstract

This project presents a natural-language-driven system for volumetric visualization that uses an agentic workflow to design transfer functions aligned with user-specified objectives. Viewed as a research problem, it asks how multi-agent LLMs can be used to optimise transfer functions for volumetric visualization, enabling users to specify visual goals through natural language while the system plans and refines suitable mappings automatically. Instead of directly manipulating low-level transfer-function controls, a user issues instructions such as “highlight the metallic objects in the backpack” or “emphasise the vertebral column and suppress soft tissue,” and the system decomposes these requests into steps that resemble an expert’s workflow: analysing the intensity distribution, proposing a transfer function that maps intensities to colour and opacity, rendering multiple views, and assessing how well the results satisfy the stated objective.

In this project, we integrate a GPU-accelerated renderer based on VTK/PyVista, a knowledge agent for transfer-function generation, an evaluation agent for image-based quality assessment, and an orchestrator that coordinates the entire pipeline. In the current implementation, the knowledge agent parameterises the transfer function primarily through piecewise intensity–opacity segments, while the renderer applies a fixed plasma colour map; however, the overall design is agnostic to the specific colour model. A working prototype is deployed on a Debian server with AMD GPU support and accessed via a SvelteKit web client.

Case-study experiments on real-world volumetric datasets show that the system can converge to visually meaningful transfer functions within a small number of refinement cycles, without any manual editing of the underlying mapping. The results indicate that natural-language control, coupled with a multi-agent LLM workflow, is a viable interface for volume visualization and provides a practical platform for exploring intelligent, LLM-driven transfer-function planning and refinement.

## 1. Introduction

Volumetric visualization underpins many tasks in medicine, scientific computing, geosciences, and engineering. In CT and MRI, clinicians rely on volume rendering to inspect anatomical structures. In industrial and security settings, volumetric imaging is used to detect defects or identify materials inside closed containers.

Traditional practice often relies on experts designing and tuning transfer functions using specialised tools like Tomviz or ParaView. These tools provide transfer-function widgets with adjustable control points and, in many cases, a histogram view of the scalar values. Users drag and adjust these controls to emphasise or suppress specific intensity ranges. While powerful and flexible, this workflow typically assumes domain expertise, can be time-consuming and iterative, and can make it challenging to reproduce settings consistently across users and datasets.

### 1.1 Transfer Functions in Volume Rendering

Transfer functions determine which structures become visible in volume rendering. A transfer function is a mapping of scalar intensity values  $v$  to visual properties (colour and opacity). A 1D transfer function can be written as

$$T : \mathbb{R} \rightarrow \mathbb{R}^4, \quad v \mapsto (R(v), G(v), B(v), \alpha(v)),$$

where  $R(v), G(v), B(v)$  define colour and  $\alpha(v)$  defines opacity. During rendering, rays are cast through the volume; at each sample point, the scalar intensity  $v$  is looked up in the transfer function, and the resulting colour and opacity are composited along the ray. Regions with high opacity become visually prominent, while regions with low opacity fade into the background.

In practice, transfer functions are often designed using an intensity histogram generated from a volumetric dataset. Users place and adjust control points over ranges of intensity to create narrow, high-opacity peaks that highlight specific materials, broad, low-opacity regions that keep background structures visible but unobtrusive, and smooth ramps that avoid hard visual boundaries.

In this project, the transfer function is factored into a **colour map** and an **opacity map**. Focus is on modelling opacity as a piecewise function  $\alpha(v)$ , while colour is determined by a fixed plasma colour map  $C(v)$ , where

$$C : \mathbb{R} \rightarrow \mathbb{R}^3, \quad v \mapsto (R(v), G(v), B(v)),$$

and the full transfer function can be written as

$$T(v) = (C(v), \alpha(v)).$$

## 1.2 Research Question and Approach

This project explores an alternative approach: controlling volumetric visualization through natural language, with an agentic workflow that mimics how an expert would plan a transfer function for a specific objective. Rather than working directly with low-level transfer-function controls, users describe their intent in plain language, for example:

- “Highlight the metallic objects in the backpack and remove everything else.”
- “Show the vertebral column prominently and reduce surrounding soft tissue.”
- “Make high-density regions opaque and low-density regions semi-transparent.”

The system interprets these instructions, decomposes them into steps that resemble an expert’s workflow, and plans a corresponding transfer function that maps intensities to colour and opacity. It analyses the volume’s intensity distribution, proposes and refines a transfer function, applies it to the volume, and evaluates the resulting images against the stated objective. Users interact through text and images, while the system takes responsibility for transfer-function planning, refinement, and optimisation.

This leads to the central research question: **how can multi-agent LLMs be used to optimise transfer functions for volumetric visualization, enabling users to specify visual goals through natural language while the system plans and refines suitable mappings automatically?**

## 1.3 Project Objectives and Contributions

The project was guided by four main objectives:

1. **Natural-language interface for transfer-function design**  
Allow users to describe desired visual outcomes in plain language instead of tuning low-level rendering parameters.
2. **Interpretable transfer-function representation**  
Model transfer functions as structured, piecewise intensity–opacity mappings: the Knowledge Agent predicts segment bounds and opacity values, while the renderer applies a fixed plasma colour map. The result is human-readable, easy to validate, and serialisable to JSON.
3. **Multi-agent system architecture**  
Design a set of specialised agents (orchestrator, knowledge, renderer, evaluation) that collaborate to plan, render, and assess transfer functions in an iterative loop.
4. **End-to-end implementation and demonstration**  
Build and deploy a complete, usable prototype spanning backend services, a web client, and a GPU-enabled environment for running case studies on real volumetric datasets.

This project shifts the cognitive and tooling burden from the user to an agentic pipeline, while maintaining user control over the final visualization outcome. The broader impact includes making volumetric visualization more accessible to non-experts and enabling faster exploration of alternative visualizations for the same dataset.

## 2. Data and Methodology

### 2.1 Data Description

#### 2.1.1 Dataset Characteristics

This project uses real-world volumetric datasets in RAW format, which are common in medical imaging (CT, MRI) and industrial scanning applications. The datasets consist of three-dimensional arrays of scalar intensity values, where each voxel represents a measurement at a specific spatial location.

Two representative datasets were used for case-study experiments:

1. **Carp CT Dataset**

A computed tomography scan of a carp fish, containing anatomical structures including the vertebral column, ribs, skull, and soft tissue. The dataset is stored as a RAW file with `uint16` data type, requiring specification of three-dimensional volume dimensions for proper interpretation.

2. **Backpack Security Scan Dataset**

A volumetric scan of a backpack containing various objects, including metallic bottle- or can-like items. This dataset represents a security screening scenario where high-density metallic objects need to be identified within lower-density materials (fabric, plastic, etc.).

#### 2.1.2 Data Preprocessing

The volumetric data undergoes minimal preprocessing in this system:

- **Format validation:** The system accepts RAW volumetric files with user-specified dimensions (Dim 1–3) and data type (e.g., `uint16`).
- **Intensity histogram computation:** For each dataset, the Renderer Agent computes a log-normalised histogram of intensity values. This histogram serves two purposes: (1) it provides visual feedback to the Knowledge Agent about the distribution of intensities in the volume, and (2) it is displayed in the client interface to help users understand the data characteristics.
- **Normalization:** Intensity values are normalized to the range of the dataset for consistent transfer-function application across different data types and scales.

No additional cleaning, filtering, or transformation is performed; the system operates directly on the raw intensity values, relying on the transfer function to map these values to visual properties.

#### 2.1.3 Exploratory Data Analysis

The primary exploratory analysis tool in this system is the intensity histogram, which reveals:

- The distribution of intensity values across the volume
- Potential peaks or modes that may correspond to different materials or anatomical structures
- The overall dynamic range of the dataset

The histogram is computed once per dataset and reused across all iterations for a given task, ensuring consistency in the Knowledge Agent’s decision-making process.

## 2.2 Methodology

### 2.2.1 System Architecture Overview

In this project, the system is implemented as a distributed multi-agent system with clear responsibilities.

- A **SvelteKit client** provides the user interface.
- Four **Python-based agents** handle orchestration, knowledge, rendering, and evaluation.
- A **shared infrastructure layer** provides storage and messaging.

**Client Application** The SvelteKit client allows users to:

- Upload volumetric data and enter a natural-language instruction,
- View a dashboard of tasks and their current status,
- Inspect task details, including:
  - Chronological action history,
  - Histogram images and rendered views,
  - Evaluation outcomes.

The client communicates with the Orchestrator Agent via HTTP APIs and periodically refreshes task status.

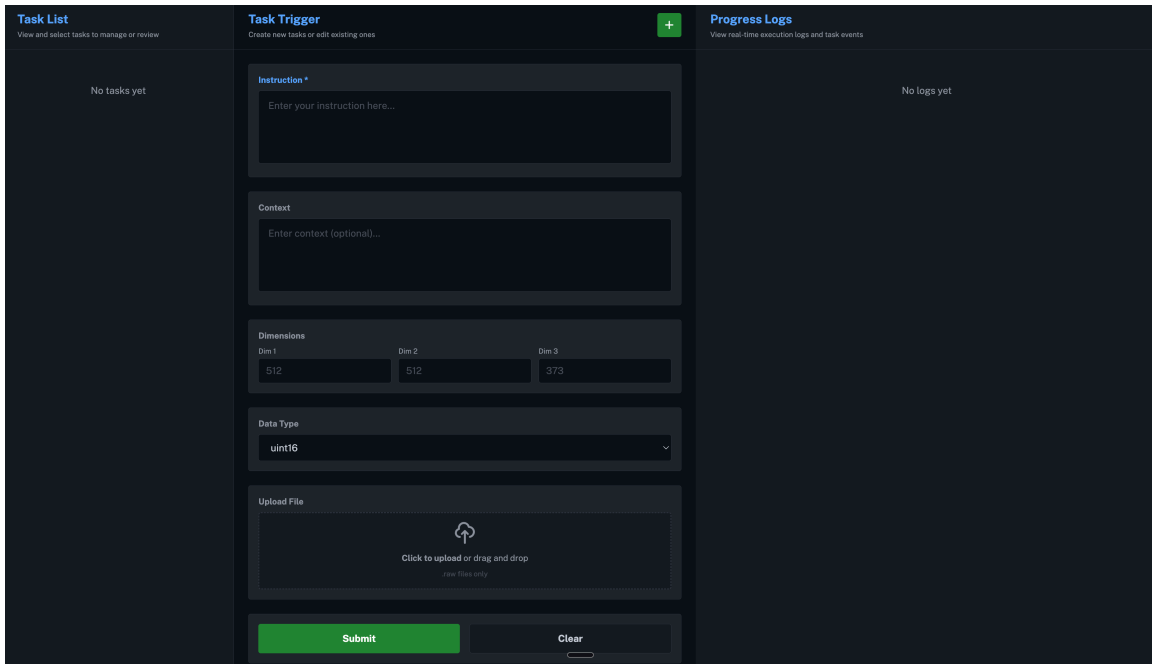


Figure 1: Client interface for task creation and monitoring.

The client is organised into three main vertical panels:

- **Left panel – Task List**  
This panel shows all existing tasks for the current user. When no tasks have been created yet, it displays a placeholder message (for example, “No tasks yet”). Once tasks exist, each entry in the list is clickable and loads the corresponding task details in the middle and right panels.
- **Middle panel – Task Trigger and Configuration**  
This panel is used to create a new task or modify the configuration for a selected task. It includes:
  - A text box for the natural-language instruction,

- An optional context or description field,
- Input fields for the volume dimensions (Dim 1–3),
- A data-type selector (for example, `uint16`),
- A file upload area for RAW volumetric data,
- Buttons to submit the task to the orchestrator or clear the form. This is the main place where users specify what they want the system to visualise and which dataset to use.

- **Right panel – Progress Logs and Status**

This panel shows the execution history and current status of the active task. It presents a scrollable log of messages summarising state transitions (for example, histogram computation, transfer-function inference, rendering, evaluation), agent-level actions, and any errors. When a task has not yet been run, the panel shows a placeholder message (for example, “No logs yet”).

**Orchestrator Agent** The Orchestrator Agent coordinates the overall workflow for each task. It:

- Accepts new task requests from the client,
- Stores task metadata and artefact references in MongoDB,
- Invokes other agents through RabbitMQ queues and gRPC calls,
- Tracks state transitions such as created, in progress, refining, completed, or failed,
- Uses a LangGraph-based control flow to decide the next action.

At each step, the orchestrator prompts an LLM with the current task state (instruction, existing artefacts, recent evaluations) and receives an action such as “compute histogram”, “generate transfer function”, “render volume”, or “evaluate renderings”. Actions and outcomes are logged as part of the task history.

### Instruction-to-Visualization Loop

Operationally, the orchestrator drives an iterative loop that starts from a natural-language instruction and converges to a satisfactory visualization:

1. **Receive instruction**

The client submits a volumetric dataset and an instruction. The orchestrator records this as a new task with an initial `Created` state.

2. **Validate and refine instruction**

The orchestrator uses an LLM to refine the instruction, making implicit goals explicit (for example, emphasised structures, suppressed structures, or contrast preferences). The refined instruction is stored alongside the original text.

3. **Compute intensity histogram**

If no histogram exists for the dataset, the orchestrator dispatches work to the Renderer Agent to compute a histogram of intensity values and produce a log-normalised histogram image.

4. **Predict intensity–opacity mapping**

The orchestrator calls the Knowledge Agent with the (original and refined) instruction and the histogram. The agent predicts piecewise intensity bounds and opacity values, returning a JSON specification of the transfer function, which is stored as a new version.

5. **Render images**

The Renderer Agent loads the volume, applies the fixed plasma colour map and the predicted opacity mapping, and renders multiple views around the volume. Paths to the rendered images and histogram are attached to the task.

## 6. Evaluate images

The orchestrator forwards the instruction and images to the Evaluation Agent, which outputs a decision label (`accurate`, `needs_refinement`, or `inaccurate`) together with structured feedback about what should change in the next iteration.

## 7. Repeat or terminate

If the decision is `needs_refinement`, the orchestrator triggers another knowledge-render-evaluate cycle, passing the latest feedback back to the Knowledge Agent. If the decision is `accurate` or a stagnation rule is triggered (for example, repeated iterations without improvement), the task is marked as `Completed` or `Failed`, and the current images are treated as the final result.

This loop treats transfer-function design as an optimisation process driven by language and feedback, rather than manual parameter editing. The orchestrator is responsible for maintaining task state, choosing the next action, and wiring together the specialised agents that implement each step.

## Task Lifecycle States

The orchestrator drives each task through a series of lifecycle states, from creation to completion or failure. The diagram below summarises this control flow.

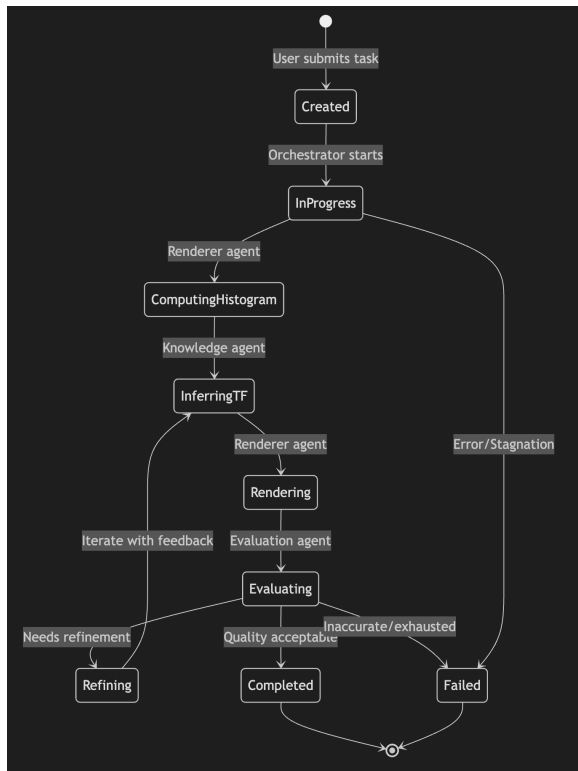


Figure 2: Task lifecycle for a task.

**Knowledge Agent** The Knowledge Agent is responsible for predicting the intensity–opacity transfer function. It receives:

- The user instruction (original and refined),
- The histogram image,
- The current or previous transfer function,

- Evaluation feedback from earlier iterations.

Using an LLM, it produces a structured JSON object describing piecewise segments of the transfer function. In the current prototype, this JSON has arrays of intensity intervals and corresponding opacity values, for example:

```
{
  "piecewise": {
    "intensity_intervals": [
      [0.0, 100.0],
      [100.0, 200.0],
      [200.0, 300.0]
    ],
    "opacity_values": [
      [0.0, 0.5],
      [0.5, 0.8],
      [0.8, 1.0]
    ]
  }
}
```

Here, each `intensity_intervals[i]` defines a range on the scalar axis, and the matching `opacity_values[i]` encodes the opacity behaviour over that range (constant or simple ramp). This JSON specification is consumed directly by the `Renderer Agent` to construct the piecewise opacity function used for volume rendering.

The output is validated for numerical consistency (bounds within the data range, no contradictory overlaps, opacity in  $[0,1]$ ) and then stored as a new transfer-function version that can be re-used, compared, or refined in subsequent iterations.

**Renderer Agent** The `Renderer Agent` performs volume rendering. It:

- Loads the volumetric dataset,
- Computes a log-normalised histogram (used by the knowledge agent and shown in the client),
- Applies the predicted intensity–opacity mapping  $\alpha(v)$ ,
- Applies a fixed plasma colour map  $C(v)$  to assign RGB colours,
- Renders the volume from a set of orbiting camera positions using GPU-accelerated ray casting.

The `renderer` writes histogram and image files to shared storage and reports their locations back to the `orchestrator`.

**Evaluation Agent** The `Evaluation Agent` assesses how well the rendered images match the instruction. It receives:

- The instruction (original and refined),
- The current set of rendered images,
- A summary of the current transfer function and recent history.

Using a multimodal LLM, it outputs:



- A decision: `accurate`, `needs_refinement`, or `inaccurate`,
- Structured feedback describing what should change (for example, “metal appears semi-transparent in several views; increase opacity for high intensities”).

This feedback is stored and used in subsequent iterations by the Knowledge Agent.

### 2.2.2 Transfer-Function Representation

The transfer function in this project is implemented as a set of piecewise linear opacity segments:

- The intensity axis is partitioned into intervals  $[v_{\min}, v_{\max}]$ .
- Each interval is associated with one or more opacity parameters (constant or simple ramp).
- The renderer combines:
  - $\alpha(v)$  from the knowledge agent,
  - The fixed plasma colour map  $C(v)$ .

This reduces the parameter space for the knowledge agent and increases robustness. The representation is stored as JSON, making it easy to version, inspect, and overlay on histograms.

### 2.2.3 Orchestrator Logic and Control Flow

The orchestrator uses a LangGraph-based graph to represent possible states (no histogram, histogram only, transfer function available, renders available, evaluated, etc.). At each step, the orchestrator:

1. Builds a summary of the current state,
2. Asks an LLM which action to take next,
3. Validates the response against a schema,
4. Executes the action and records the result.

If repeated iterations fail to improve the evaluation label, a simple stagnation rule terminates the task and marks it as non-converged.

### 2.2.4 LLM Models and Prompt Engineering

In this project, we use large language models as the main reasoning components of the system. Different roles are assigned to different model configurations: vision-capable models for transfer-function inference and image-based evaluation, and a lighter text configuration for orchestration. Prompt design and explicit schemas control how these models interact with one another and with the rest of the pipeline.

#### Model roles and capabilities:

LLMs appear in three places, but there are two primary reasoning roles:

- A **vision-capable Knowledge Agent** that plans and refines the transfer function. It consumes the textual instruction, prior feedback, and visual artefacts such as the intensity histogram (and optionally thumbnails from previous renderings). Its main output is a JSON description of the piecewise intensity–opacity mapping that the Renderer Agent turns into an opacity function over the scalar range.

- A **vision-capable Evaluation Agent** that inspects rendered images alongside the instruction and a short description of the current transfer function. It decides whether the visualization satisfies the stated objective and, if not, what should change. Its output is a discrete decision label (`accurate`, `needs_refinement`, or `inaccurate`) plus structured feedback that can be fed back into the Knowledge Agent.

In addition, the **orchestrator logic** reuses a compact text LLM configuration from the same family, enabled with structured tool calling. It receives a summarised task state (original and refined instruction, available artefacts, recent evaluations) plus a list of allowed actions and returns a single tool choice with arguments as a small JSON object. This keeps control flow explicit and machine-readable without introducing a separate “orchestration-specific” model.

### Prompt structure and output schemas:

Prompt engineering in this project centres on explicit output schemas rather than free-form prose. For each LLM role, the prompt:

- Separates **inputs** (instruction, images, state) from **required outputs**,
- Defines a **strict JSON schema** for the output, and
- States the **role** of the model and what it must not do.

For the **Knowledge Agent**, the prompt makes clear that its sole responsibility is to produce an intensity–opacity mapping consistent with the objective and histogram. It is given examples of valid and invalid segment definitions and instructed to emit only a JSON object that matches the expected schema. Any output that fails validation (for example, overlapping intervals, values outside the data range, or opacity outside  $[0, 1]$ ) is rejected, and the model is asked to correct its response.

For the **Evaluation Agent**, the prompt is split into two clearly marked parts: a decision and an explanation. The model must (1) choose exactly one label from the allowed set and (2) provide a short, actionable description of what should change in the transfer function to better satisfy the instruction. This keeps answers compact and makes them easy for the orchestrator and Knowledge Agent to consume.

For the **orchestration logic**, the prompt lists the available tools (compute histogram, call Knowledge Agent, render views, run Evaluation Agent, and so on) with concise descriptions, and requires the model to choose exactly one tool plus any necessary arguments. The tool-calling interface ensures that control decisions are always expressed as structured function calls.

### Vision models and the role of images:

Vision-capable LLMs are necessary for the Knowledge and Evaluation Agents because their decisions depend on visual artefacts:

- The **Knowledge Agent** uses the histogram image (and optionally thumbnails from previous renderings) to understand how intensities are distributed and how earlier transfer functions behaved. This supports behaviours such as “push opacity towards the high-intensity tail to emphasise metal” or “suppress mid-range opacity where soft tissue dominates,” which would be difficult to capture from text alone.
- The **Evaluation Agent** must directly inspect rendered images to judge whether the instruction has been satisfied. It needs to recognise, for example, whether metallic objects stand out from fabric in the backpack dataset or whether the vertebral column is clearly visible in the spine dataset. A text-only model would not be able to perform this assessment.

Prompts for these agents explicitly ask the model to ground its reasoning in visual features (“bright, opaque structures,” “low-opacity background,” “structures still overlapping”) rather than generic language. This grounding is important for producing feedback that can be mapped back to concrete edits in the intensity–opacity segments.

### 2.2.5 Deployment Environment

The prototype is deployed on a single Debian-based server with AMD GPU support. The key elements are:

- A Python environment with VTK/PyVista and supporting libraries, configured for headless GPU-accelerated rendering,
- A MongoDB instance and RabbitMQ broker running as services,
- Separate processes for orchestrator, knowledge, renderer, and evaluation agents,
- A SvelteKit client served via a Node.js process or reverse proxy.

### 2.2.6 Evaluation Metrics

The system uses qualitative evaluation through the Evaluation Agent, which assesses rendered images against the user’s natural-language instruction. The evaluation produces:

- A discrete decision label: `accurate`, `needs_refinement`, or `inaccurate`
- Structured feedback describing what should change in the transfer function

This approach is appropriate for this problem because the goal is to satisfy user intent expressed in natural language, which is inherently subjective and difficult to quantify with traditional metrics. The iterative refinement loop allows the system to converge toward satisfactory visualizations through multiple cycles of generation, rendering, and evaluation.

## 3. Results

This section summarises how the system behaved in this project on two representative tasks: highlighting the vertebral column of a carp in a CT volume, and revealing metallic bottle- or can-like objects in a backpack scan. In both cases, the system started from a single natural-language instruction and relied on the multi-agent loop to refine a transfer function until the Evaluation Agent judged the result to be satisfactory.

### 3.1 Experimental Configuration

For both tasks, the same model configuration was used:

- Orchestration logic: text LLM with tool-calling support (`gpt-oss:120b`)
- Knowledge Agent: vision-capable model (`qwen2.5v1:32b`)
- Evaluation Agent: vision-capable model (`qwen2.5v1:32b`)

Each experiment followed the standard pipeline:

1. Upload RAW volumetric data and enter a single textual instruction in the SvelteKit client.
2. Let the orchestration agent drive the loop of histogram computation, transfer-function inference, rendering, and evaluation.
3. Stop when the Evaluation Agent returned an `accurate` decision or a stagnation rule prevented further improvement.
4. Save the final transfer function, the orbit of 12 rendered views, and the last evaluation report as the task artefacts.

## 3.2 Carp Dataset – Highlighting the Vertebral Column

### Instruction

> “Can you highlight the carp fish’s inner spine (vertebral column)?”

For the carp dataset, the default configuration was overridden for testing purposes, so that all three agents (orchestrator, Knowledge, and Evaluation) used the same underlying private model (`gpt-5`). Starting from the single instruction above, the orchestration logic drove two complete iterations of the histogram  $\rightarrow$  transfer-function inference  $\rightarrow$  rendering  $\rightarrow$  evaluation loop. No manual editing of the transfer function was performed; both the initial mapping and its refinement were produced by the Knowledge Agent and accepted (or rejected) based on the Evaluation Agent’s feedback. The task was marked complete after the second iteration, when the Evaluation Agent judged the visualisation to meet the stated objective.

The two evaluations were:

### Iteration 1 – Evaluation summary

Inputs are incomplete—no individual evaluations or raw responses were provided. From the available view, the tapered volume shows faint internal orange features and basal bands but low contrast against the dominant purple, making target structures hard to distinguish. Overall visibility appears insufficient to confidently meet the objective across angles.

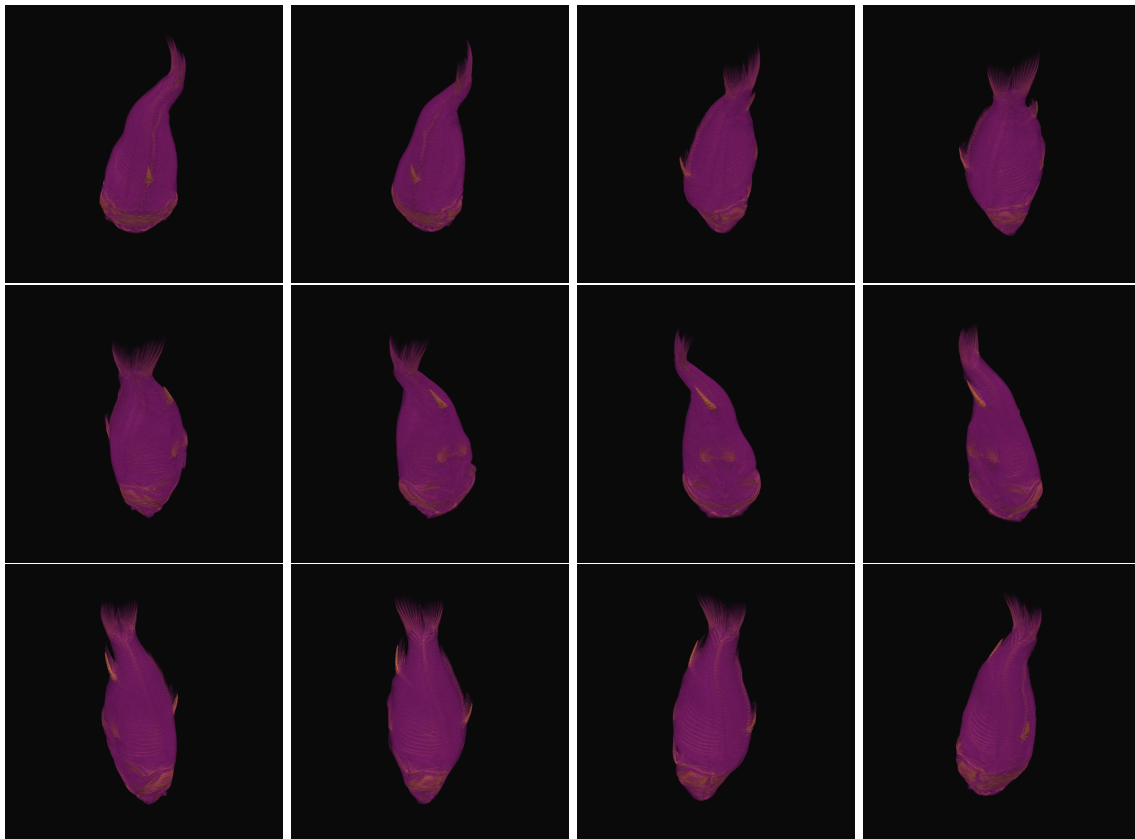


Figure 3: Carp dataset – iteration 1 orbit views (12-camera sweep around the volume).

### Iteration 2 – Final evaluation summary

The vertebral column is clearly visible as a continuous, bright, high-opacity structure running from head to tail, distinct from surrounding tissue throughout the rotation. High-density bony voxels (spine, ribs, skull) are highlighted with a bright warm colour while soft tissue appears semi-transparent purple, providing context without obscuring the spine. Across all viewing angles, the spine remains identifiable and distinguishable, meeting the objective to highlight the carp fish’s inner spine. Minor polish (for example, slightly lowering soft-tissue opacity or adding a cutaway to reduce rib/skull overlap) could increase contrast further, but the current visualization is already functional for the stated goal.

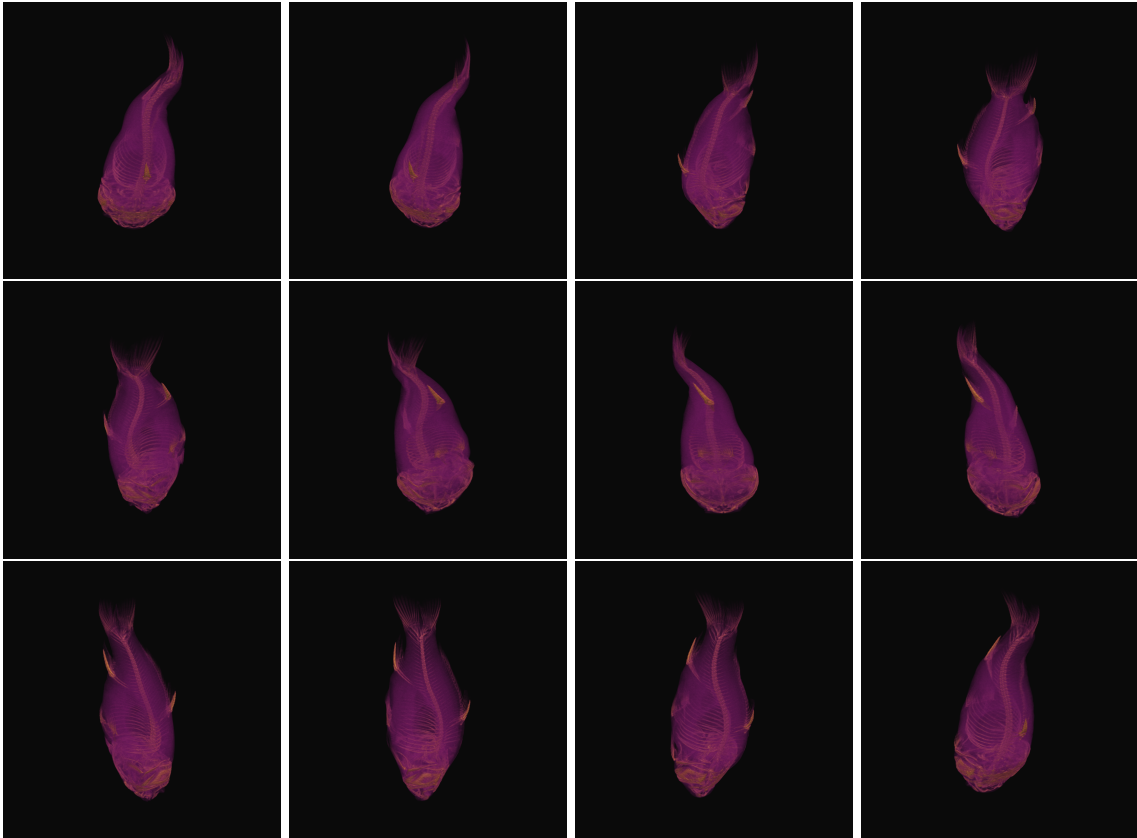


Figure 4: Carp dataset – iteration 2 orbit views (12-camera sweep around the volume).

Taken together, the two iterations show the agentic loop converging from an initial, partially satisfactory mapping to a final transfer function that clearly isolates the carp's vertebral column while keeping surrounding soft tissue as a low-opacity context.

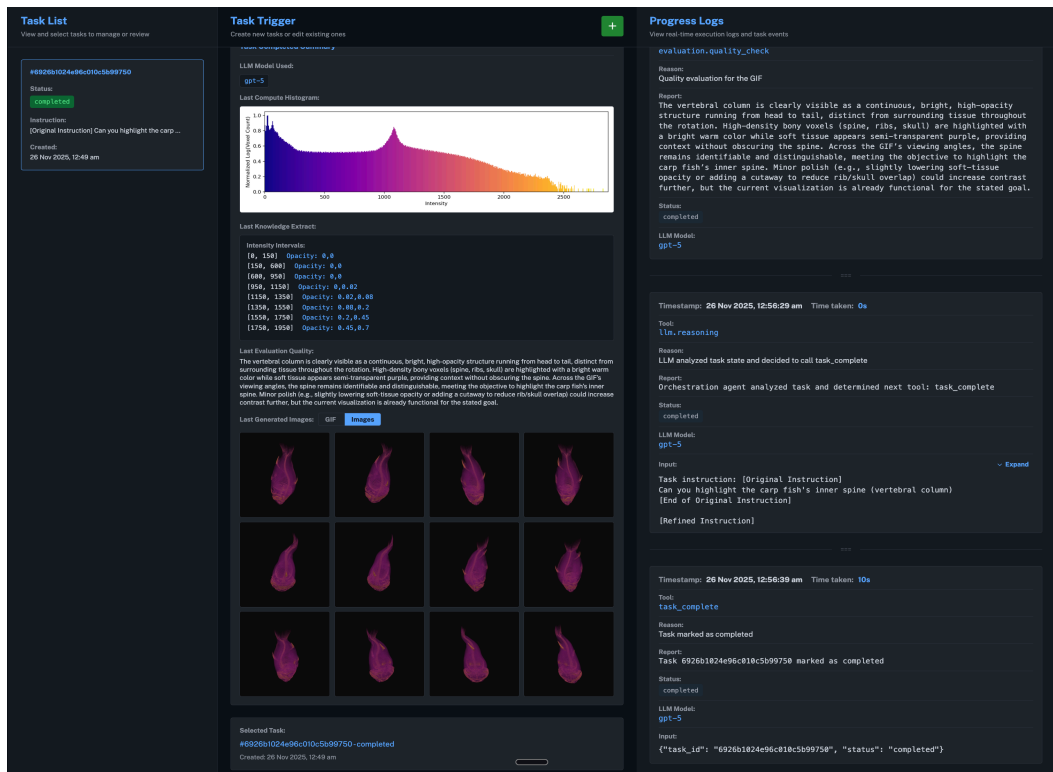


Figure 5: Dashboard of Carp Fish Dataset

### 3.3 Backpack Dataset – Metallic Bottle or Can Detection

#### Instruction

> “Can you identify any object that appears like a bottle or a can made of metal?”

For the backpack dataset, the orchestration agent ran the same iterative loop: infer a transfer function, render multiple views, and ask the Evaluation Agent whether metallic bottle- or can-like objects were clearly visible against the rest of the backpack contents. The process continued until the Evaluation Agent labelled the result as accurate or further iterations stopped improving the evaluation. The task was marked complete after the first iteration, when the Evaluation Agent judged the visualisation to meet the stated objective.

The evaluation was:

#### Iteration 1 – Evaluation summary

The rendering clearly emphasizes very high-density material in a bright yellow/orange with high opacity against a black background. Two cylindrical, can-like metal objects are prominently visible, with rims and sidewalls distinguishable as the view rotates, allowing the user to identify bottle/can-shaped items. The transfer function effectively separates dense metal from surrounding lower-density structures, which appear more translucent magenta. Minor clutter from small metallic fragments and mid-density wisps remains but does not obscure the targets. Overall, the visualization is functional for the stated objective; suggested tweaks would be incremental polish to reduce noise and further isolate the metallic cylinders.

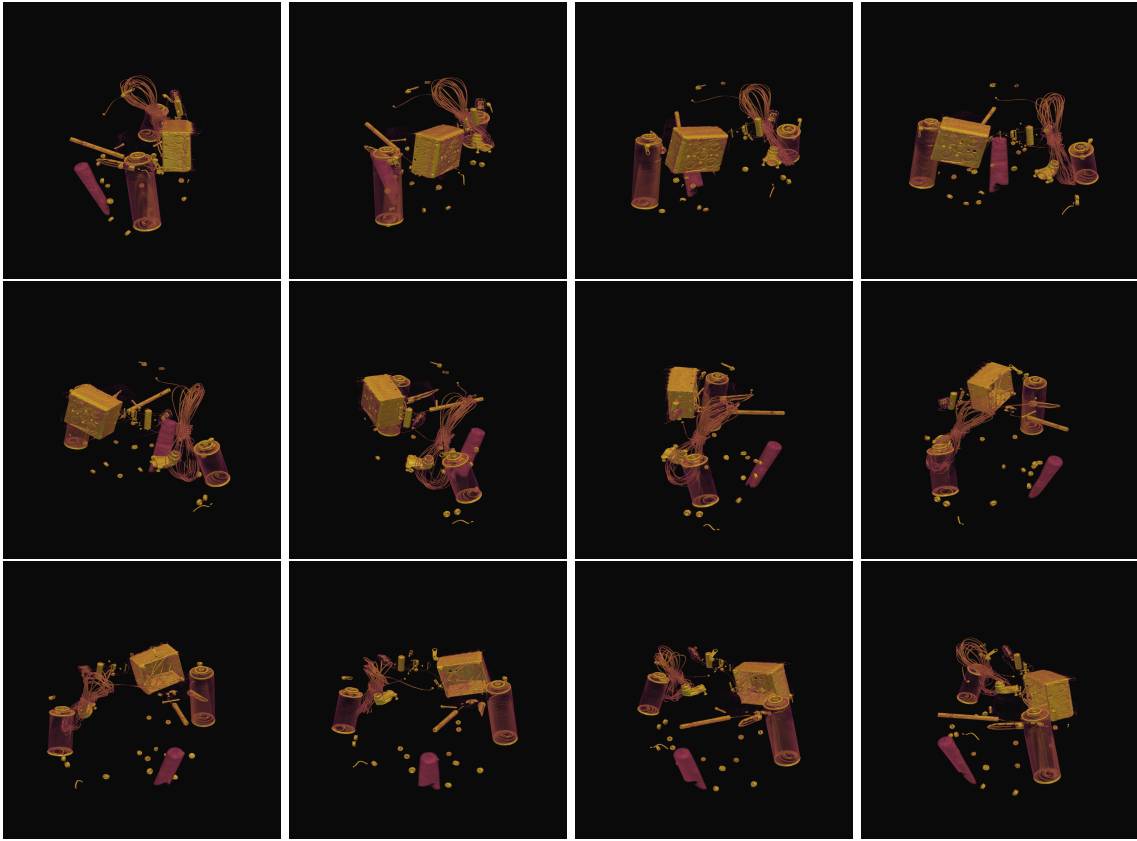


Figure 6: Backpack dataset – iteration 1 orbit views

### Task List

View and select tasks to manage or review

- #6926b8964e96c010c5b99752

Status: Completed

Instruction: [Original Instruction] Can you identify any object...

Created: 26 Nov 2025, 01:21 am
- #6926b8744e96c010c5b99751

Status: Failed

Instruction: [Original Instruction] Iterations placeholder [End...

Created: 26 Nov 2025, 01:21 am
- #6926b1024e96c010c5b99750

Status: Completed

Instruction: [Original Instruction] Can you highlight the corp...

Created: 26 Nov 2025, 12:49 am

### Task Trigger

Create new tasks or edit existing ones

LLM Model Used: gpt-5

Last Compute Histogram

Last Knowledge Extract

Intensity Intervals:

(9, 288)	Opacity: 0.0
(289, 864)	Opacity: 0.0
(865, 1340)	Opacity: 0.05
(1341, 1788)	Opacity: 0.05, 0.1
(1789, 2088)	Opacity: 0.1, 0.15
(2089, 2288)	Opacity: 0.15, 0.2
(2289, 2488)	Opacity: 0.2, 0.25
(2489, 2688)	Opacity: 0.25, 0.3

Last Evaluation Quality:

The rendering clearly emphasizes very high-density material in a bright yellow/orange with high opacity against a black background. Two cylindrical, can-like metal objects are prominently visible, with rims and sidewalls distinguishable as the view rotates, allowing the agent to identify distinctive-shaped items. The transfer function effectively separates dense metal from surrounding lower-density structures, which appear more translucent/magenta. Minor clutter from small metallic fragments and mid-density wire systems that does not obscure the targets. Overall, the visualization is functional for the stated objective; suggested tweaks would be incremental: adjust opacity ramps to further isolate the metallic cylinders.

Last Generated Images: [GIF](#) [Images](#)

Selected Task: #6926b8964e96c010c5b99752 - completed

Created: 26 Nov 2025, 01:21 am

### Progress Logs

View real-time execution logs and task events

Success: use opaque ramp and ramps only: ramps: opacity in low ranges for context, while ramping steeply from 2688 upward to emphasize metals; the very top densities (>3888) reach near-opaque. For color, apply a distinct cool hue (e.g., cyan/teal) to the highest intervals to separate them visually from other structures. No prior attempt exists; if a previous one had failed, it likely kept too much mid-density opacity, causing clutter and masking metallic shapes.

Status: completed

LLM Model: gpt-5

Input: You are the Knowledge Agent for GenVolvis.

Task ID: 6926b8964e96c010c5b99752

Input Histogram:

Output Piecewise Data:

Interval: (9, 288)	Opacity: 0.0
Interval: (289, 864)	Opacity: 0.0
Interval: (865, 1340)	Opacity: 0.05
Interval: (1341, 1788)	Opacity: 0.05, 0.1
Interval: (1789, 2088)	Opacity: 0.1, 0.15
Interval: (2089, 2288)	Opacity: 0.15, 0.2
Interval: (2289, 2488)	Opacity: 0.2, 0.25
Interval: (2489, 2688)	Opacity: 0.25, 0.3
Interval: (2689, 3088)	Opacity: 0.35, 0.4
Interval: (3089, 3488)	Opacity: 0.4, 0.4
Interval: (3489, 3888)	Opacity: 0.4, 0.55
Interval: (3889, 4071)	Opacity: 0.5, 1

Timestamp: 26 Nov 2025, 01:23:50 am Time taken: 0s

Tool: `llm_reasoning`

Reason: LLM analyzed task state and decided to call `renderer_generate_volume_images`

Reason: Orchestration agent analyzed task and determined next tool: `renderer_generate_volume_images`

Status: completed

LLM Model: gpt-5

Input: Task instruction: [Original Instruction] Can you identify any object that appears like a bottle or a can made of metal?

Figure 7: Dashboard of Backpack Dataset

## 3.4 Summary of Results

Both case studies demonstrate that the system can successfully converge to visually meaningful transfer functions within a small number of refinement cycles. The carp dataset required two iterations to achieve satisfactory visualization of the vertebral column, while the backpack dataset achieved acceptable results in a single iteration. In both cases, no manual editing of transfer functions was required; the system autonomously planned and refined the mappings based on natural-language instructions and iterative feedback.

## 4. Discussion and Limitations

### 4.1 Interpretation of Results

The results indicate that natural-language control, coupled with a multi-agent LLM workflow, is a viable interface for volume visualization. The system successfully translated high-level user instructions into concrete transfer-function parameters, demonstrating that LLMs can effectively bridge the gap between natural language and low-level rendering controls.

The iterative refinement loop proved effective in both case studies. For the carp dataset, the first iteration produced a partially satisfactory result with low contrast, but the second iteration successfully isolated the vertebral column while maintaining context through semi-transparent soft tissue. The backpack dataset achieved its objective in a single iteration, suggesting that the system can sometimes converge quickly when the instruction aligns well with the intensity distribution of the target structures.

### 4.2 What Worked and What Didn't

#### Strengths:

- The multi-agent architecture successfully decomposed the complex task of transfer-function design into manageable, specialized components.
- The vision-capable LLMs (Knowledge and Evaluation Agents) effectively utilized visual artifacts (histograms and rendered images) to make informed decisions.
- The structured JSON output format for transfer functions enabled validation and versioning, reducing errors and improving robustness.
- The iterative refinement loop allowed the system to converge from initial approximations to satisfactory results.

#### Challenges:

- The system's performance is highly dependent on model choice and prompt design. Early experiments with weaker models produced unstable results.
- The evaluation process is qualitative and subjective, making it difficult to establish objective success criteria.
- Some tasks may require multiple iterations, increasing computational cost and latency.

### 4.3 Limitations

#### 4.3.1 LLM-Driven Nature and Reliability

In this project, the system is fully LLM-driven: every transfer-function update, rendering decision, and stopping criterion is ultimately controlled by the interaction between the Knowledge Agent and the Evaluation Agent. This has important consequences for reliability.

First, there is no guarantee that the agentic loop will always follow a “winning path”. The Evaluation Agent may occasionally produce **false positives**, labelling a result as **accurate** even when the visualisation is only partially aligned with the user's intent. Conversely, it can generate **false negatives**,



asking for further refinement when the current images are already acceptable. The Knowledge Agent can also become stuck in local optima, making small adjustments that do not materially improve the visual outcome.

Second, the quality of the behaviour is tightly coupled to both **model choice** and **prompt design**. Early experiments with weaker models and less constrained prompts produced unstable transfer functions, noisy evaluations, and inconsistent convergence behaviour. After iterating on the prompts and switching to more capable models for both the Knowledge and Evaluation Agents, the system’s behaviour became noticeably more stable: evaluations were more consistent across similar images, and tasks usually converged in a small number of iterations.

Overall, across the carp and backpack experiments and additional informal tests, this model configuration (orchestration logic plus `qwen2.5v1:32b` for knowledge and evaluation) performed **fairly well** once prompts and schemas were tuned. However, the results should still be interpreted as outcomes of a probabilistic, learning-based pipeline rather than deterministic guarantees. In practical use, human inspection of the final images remains important, especially for safety-critical or clinically relevant scenarios.

### 4.3.2 Data Quality and Assumptions

The system assumes that volumetric datasets are provided in RAW format with accurate dimension and data-type specifications. It does not handle corrupted data, missing values, or format inconsistencies. The intensity histogram computation assumes that the entire volume is relevant; no spatial filtering or region-of-interest selection is performed.

### 4.3.3 Computational Constraints

The system requires GPU acceleration for efficient volume rendering. The iterative refinement loop can be computationally expensive, especially when multiple iterations are needed. Each iteration involves LLM inference (which may have latency and cost implications), volume rendering from multiple camera positions, and image evaluation.

## 5. Conclusion

This project demonstrates that multi-agent LLMs can be effectively used to optimize transfer functions for volumetric visualization, enabling users to specify visual goals through natural language while the system plans and refines suitable mappings automatically. The key contributions of this work are:

1. **Natural-language interface for transfer-function design:** The system successfully translates high-level user instructions into concrete transfer-function parameters, making volumetric visualization more accessible to non-experts.
2. **Multi-agent architecture:** The specialized agent design (orchestrator, knowledge, renderer, evaluation) effectively decomposes the complex task of transfer-function design into manageable, co-operating components.
3. **Iterative refinement workflow:** The system demonstrates that an iterative loop of generation, rendering, and evaluation can converge to satisfactory visualizations within a small number of cycles.
4. **End-to-end prototype:** A complete, deployable system was built and tested on real-world volumetric datasets, demonstrating the practical feasibility of this approach.
5. **Interpretable transfer-function representation:** The piecewise intensity–opacity mapping provides a human-readable, versionable representation that can be validated and refined.

The case-study experiments on carp CT and backpack security scan datasets show that the system can successfully identify and highlight target structures (vertebral column, metallic objects) based on natural-language instructions, without requiring manual parameter tuning. While the system’s performance depends on model choice and prompt design, the results indicate that this approach is viable for practical applications.

The research question—**how can multi-agent LLMs be used to optimise transfer functions for volumetric visualization, enabling users to specify visual goals through natural language while the system plans and refines suitable mappings automatically?**—has been answered affirmatively through the design, implementation, and evaluation of the prototype system.

For practitioners, this work suggests that natural-language interfaces can reduce the cognitive and tooling burden associated with traditional transfer-function design workflows, potentially enabling faster exploration of alternative visualizations and making volumetric visualization more accessible to domain experts who may not be familiar with specialized visualization tools.

## 6. Appendix

### 6.1 Source Code and Repository Access

The prototype in this project is implemented across several private Git repositories:

- **Orchestrator Agent**  
<https://csgit.ucalgary.ca/joshua.ogunbo/genvolvis-orchestrator-agent>
- **Evaluation Agent**  
<https://csgit.ucalgary.ca/joshua.ogunbo/genvolvis-evaluation-agent>
- **Renderer Agent**  
<https://csgit.ucalgary.ca/joshua.ogunbo/genvolvis-renderer-agent>
- **Knowledge Agent**  
<https://csgit.ucalgary.ca/joshua.ogunbo/genvolvis-knowledge-agent>
- **Shared Packages (common utilities and models)**  
<https://csgit.ucalgary.ca/joshua.ogunbo/genvolvis-shared-packages>
- **Client (SvelteKit UI)**  
<https://csgit.ucalgary.ca/joshua.ogunbo/genvolvis-client>

These repositories are hosted on the University of Calgary Git service and are not publicly visible. For access requests or further technical details, please email [joshua.ogunbo@ucalgary.ca](mailto:joshua.ogunbo@ucalgary.ca) or [ualim@ucalgary.ca](mailto:ualim@ucalgary.ca).

### 6.2 Development Workflow

The system is split into:

- A core application repository containing the agents, shared Python packages, and the SvelteKit client,
- An infrastructure repository containing configuration and scripts to start services on the GPU server.

This separation makes it easier to change infrastructure without modifying the application code. Branches and pull requests track incremental changes across both repositories.