## Dr. Hana Dobrovolny's Lab

# Viral Agent-Based Model Interface
# Software Requirements Specification

**Version 1.0**

# Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 12/5/25 | 1.0 | Filled out all desired material | Norwood, Ellion |
| | | | |
| | | | |
| | | | |

# Table of Contents

# Software Requirements Specification

## 1. Introduction

### 1.1 The Purpose of the Viral Agent-Based Model Interface

The Viral Agent-Based Model (ABM) Interface is being developed to make an existing CUDA-based viral infection simulation accessible to a broader range of users, including undergraduate students, graduate researchers, and faculty members. Currently, the ABM can only be used by manually editing C/CUDA code, adjusting parameters directly in source files, and running simulations via the terminal. This workflow is error-prone, difficult for non-programmers, and slows down experimentation.

The purpose of the system is to provide a graphical user interface (GUI) that allows users to:

- Configure all simulation parameters through intuitive input fields, sliders, and toggles
- Run the simulation without interacting directly with CUDA code or the command line
- Visualize results in real time through graphs and output summaries
- Save and export simulation results and parameter configurations
- Enable or disable different viral transmission mechanisms (cell-to-cell and cell-free)

By creating a user-friendly interface, the system expands the usability of the ABM beyond advanced programmers, supports classroom and research use, and enables faster iteration in scientific modeling.

### 1.2 The Purpose of this Document

The purpose of this Software Requirements Specification (SRS) is to define the functional and nonfunctional requirements for Version 1.0 of the Viral Agent-Based Model Interface. This document serves as a comprehensive reference for stakeholders, including project sponsors, developers, testers, and maintainers, ensuring a shared understanding of what the software must do and how it should behave.

This SRS describes:

- The intended users and use cases
- All functional requirements for the GUI and simulation workflow
- Validation rules for all parameter inputs
- System constraints imposed by the existing CUDA model
- Required performance, usability, and reliability characteristics

- Dependencies, assumptions, and future considerations

By consolidating requirements into a structured document, the SRS minimizes ambiguity, establishes the basis for testing and verification, and guides the software's design, implementation, and maintenance.

### 1.3 Document Conventions

This SRS follows these conventions:

- **Bold text** is used for section headers and requirement categories.
- *Italic text* is used for definitions or emphasis.
- **REQ-X.Y** identifies functional requirements (e.g., *REQ-2.3: Validate parameter inputs*).
- **NFR-X.Y** identifies nonfunctional requirements (performance, usability, etc.).
- GUI components are referenced using *CamelCase* identifiers similar to their internal names (e.g., InitialVirusInput, RunSimulationButton).

### 1.4 References

The following documents and resources support this SRS:

1. **Vision and Scope Document – Viral Agent-Based Model Interface**
    - Provides high-level goals, project constraints, and planned features.
2. **Use Case Specification – ABM Interface**
    - Describes actor interactions, event flows, and system boundaries.
3. **Viral Transmission Model Source Code (CUDA)**
    - Files:
        - Viral_Transmission.cu
        - Functions/Functions.h
        - Functions/Variables.h
        - Functions/Includes.h
    - Author: TCU Biology Department & TCU Graduate Researchers
    - Description: Core ABM simulation used as the computational backend.
4. **Client Requirements Email (Model Parameters & Expected GUI Features)**

- Received from project sponsors, includes required parameters, visualization expectations, and operational constraints.

5. **IEEE SRS Standard (IEEE 830-1998)**
    - Used as a guideline for structuring this document.

- Project Glossary: URL
- Vision and Scope: URL
- Use Cases: URL
- Business Rules: URL
- User Interface Wireframe/Prototypes: URL
- Business Domain Model: URL

## 2. Project Glossary

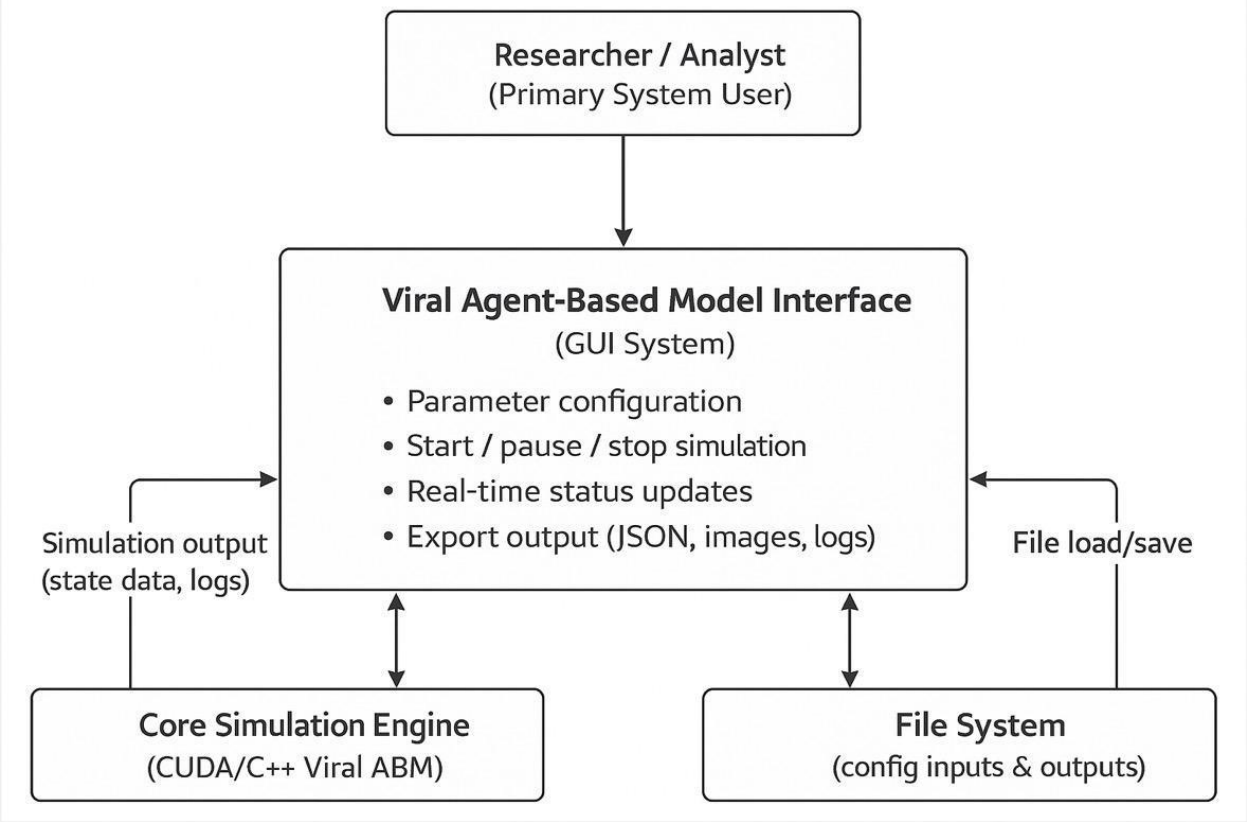
The project glossary is available here: URL.

## 3. Vision and Scope

The vision and scope document is available here: URL.

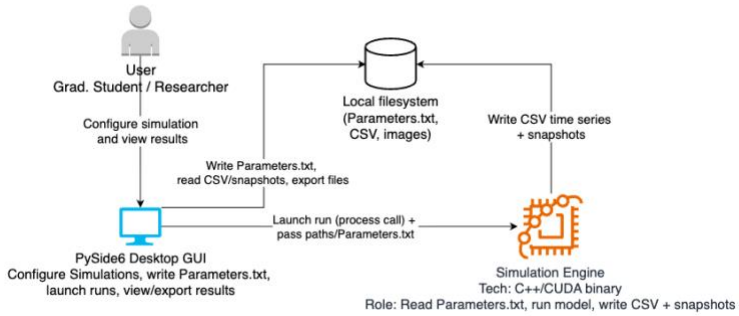
# 4. Software Architecture

## 4.1 System Context Diagram




The Level 1: Context Diagram for the Viral Agent-Based Model Interface presents a high-level view of the system and the external entities that interact with it. At the center is the ABM Interface, which researchers use to set up simulations, adjust parameters, and review results. The system interacts with the Core Simulation Engine, which runs the underlying viral model, and with the File System, which stores and retrieves configuration files and output data. This diagram highlights the primary user and the main external systems involved, providing a clear, non-technical overview of how the interface fits into the larger simulation workflow.

### 4.2 Container Diagram




The Container Diagram for the Viral Agent-Based Model Interface shows the internal architecture of the system and how the main containers interact. The system consists of three primary containers: the **PySide6 Desktop GUI**, the **Simulation Engine**, and the **File-Based Results Store**. The PySide6 Desktop GUI, implemented in Python with PySide6 (Qt for Python), provides a user-friendly interface for configuring simulations, launching runs, and viewing results. The Simulation Engine, implemented as a C++/CUDA binary, executes the viral agent-based model using parameters produced by the GUI and writes time-series and optional spatial outputs to disk. The File-Based Results Store, implemented as local filesystem artifacts (parameter files, CSV files, image snapshots), acts as the interchange layer between the GUI and the Simulation Engine, persisting all inputs and outputs for later analysis or re-loading. During development, the Simulation Engine is built using the NVIDIA CUDA toolkit and NVCC, but this toolchain is not part of the deployed runtime system.

Here are the detailed steps of a typical simulation run:

1. The user launches the Viral Agent-Based Model Interface and configures simulation inputs (initial conditions, transmission modes, rate parameters, output locations) through the PySide6 Desktop GUI.
2. The GUI validates the inputs and generates a Parameters.txt file whose keys match the variables consumed by the C++/CUDA Simulation Engine.
3. The GUI invokes the Simulation Engine as a separate process, passing the path to the parameter and output files.
4. The Simulation Engine reads Parameters.txt, executes the viral agent-based model on the GPU/CPU, and writes time-series CSV files and any configured spatial snapshots into a per-run results directory in the File-Based Results Store.
5. When the simulation completes, the GUI either automatically opens the corresponding results view or allows the user to select a run directory to inspect.
6. The GUI reads the CSV outputs and snapshots from the File-Based Results Store, renders plots and tables summarizing the simulation, and optionally previews images.
7. The user may export plots, copy configuration information (e.g., JSON or text summaries), or open the results folder in the OS file browser; all run artifacts remain on disk for future re-loading and analysis.

### 4.3 Operating Environment

OE-1: The Viral Agent-Based Model Interface (GUI) will operate on a Linux-based operating system.

OE-2: The system requires use of an Nvidia GPU to run the underlying CUDA simulation code.

OE-3: The GUI is expected to run on designated workstations within the TCU research lab/department, and all

deployment and access must comply with TCU research lab security and access protocols.

### 4.4 Design and Implementation Constraints

CO-1: The systems front end design and code should follow Python PEP-8 standards.

CO-2 The GUI must follow and be implemented with the PySide 6 framework standards.

CO-3: The deployed solution is fundamentally dependent on the presence of the NVIDIA CUDA Toolkit.

s

### 4.5 Assumptions and Dependencies

AS-1: The lab Linux workstations will maintain a stable version of the operating system, necessary Python and PySide libraries, and the required CUDA toolkit/driver versions throughout the development and deployment phase.

AS-2: The ,txt for passing parameters to and receiving time-series results from the CUDA backend is easily parsable by the Python GUI.

DE-1: Dr. Hana and Anthony must provide the biologically reasonable limits for all simulation parameters so that the GUI's input validation logic can be correctly implemented.

# 5. Functional Requirements

## 5.1 Use Cases

The use cases are available here: URL.

## 5.2 Non-Use Case Functional Requirements

- When parameter values are modified or loaded, the system shall validate all parameters against their constraints and display validation results in real-time, preventing simulation execution if any parameter is invalid.
- When a simulation configuration is saved or loaded from a file, the system shall ensure all parameters are persisted in the correct format and validated upon retrieval.
- When a simulation is running, the system shall prevent concurrent simulations and parameter modifications, maintaining state tracking throughout execution.
- When a simulation completes or is terminated, the system shall gracefully halt all computational processes, preserve output data, and return to an idle state ready for new simulations.
- When simulation output is generated, the system shall automatically create required directory structures, write all output files in a consistent format, and ensure unique file naming prevents data overwriting across multiple runs.
- When reading or writing configuration and output files, the system shall handle parsing errors gracefully, report specific format violations, and prevent corrupted data from being used in subsequent operations.
- When saving and reading time-series data, the system shall record data at specified intervals, maintain chronological ordering, validate data integrity, and handle missing or malformed records appropriately during visualization and analysis.
- When a spatial simulation is configured, the system shall generate the appropriate grid structure, initialize cell positions according to the spatial model, apply boundary conditions correctly, and exclude obstacle cells from calculations.
- When the system interfaces with the backend simulation engine, the system shall accept and manage simulation parameters, delegate computation to the backend, monitor execution status, and retrieve results without requiring detailed knowledge of underlying viral transmission model specifics.
- When saving or loading configurations, the system shall persist parameters in a structured format, validate all required parameters, handle corrupted files gracefully, and track modification state for user notifications.
- When runtime errors or I/O failures occur, the system shall log errors with diagnostic detail, attempt recovery where appropriate, gracefully halt affected processes, and preserve all available data for user access or export.
- When writing output files, the system shall ensure all related files are synchronized, parameter values match the simulation configuration, and incomplete or interrupted writes are detected and handled appropriately.
- When the UI and backend communicate, the system shall serialize parameters correctly, verify successful file operations, signal execution status changes, and validate data integrity before visualization, with appropriate timeout and error reporting mechanisms.
- When generating preview visualizations, the system shall read output files asynchronously without blocking the UI, validate file completeness, apply data selection parameters, and handle missing or incomplete data gracefully.

## 6. Business Rules

The business rules are available here: URL.
BR-1: The Program must be made available to the authorized lab participants.
BR-2: All parameters must be validated before the simulation begins
BR-3: The system's viral agent model must implement the approved computational algorithms
BR-4: System instructions must be consistent with faculty-approved instruction
BR-5: Any experimental configurations must be documented according to lab protocols
BR-6: Simulation data must be logged without modification to ensure reproducibility.

# 7. Data Requirements

## 7.1 Business Domain Model

The domain model is available here: URL.

## 7.2 Data Acquisition, Integrity, Retention, and Disposal

The system is a local desktop tool that reads and writes plain-text simulation files (e.g., Parameters.txt, PerTimeStep.txt, spatial grid files) on the user's machine. It does not use a central database, does not communicate over a network, and does not implement backups, archival, or automated deletion. All retention and disposal of files are the user's responsibility.

DI-1: The system shall store simulation inputs and outputs only as local files in directories chosen or confirmed by the user.

DI-2: The system shall not transmit simulation data over a network or to external services.

DI-3: The system shall not enforce automatic data retention or deletion; creation, backup, and disposal of simulation files are handled outside the application.

# 8. External Interface Requirements

## 8.1 User Interfaces

UI-1: **Configuration Page**: The Configuration Page provides the primary interface through which users define all simulation parameters before execution. It follows the PySide6/Qt layout conventions used throughout the product, including consistent placement of action buttons (Save, Load Preset, Reset, Run), standardized form controls, and uniform error-message displays for invalid or missing inputs. Users configure initial conditions (e.g., total cells, eclipse cells, infected cells), adjust rate parameters, select transmission mode, and specify distribution characteristics for eclipse and infection periods. The page includes built-in validation that highlights erroneous fields, tooltips for parameter explanations, and keyboard-accessible navigation consistent with lab accessibility requirements.

UI-2: **Run & Result Page**: The Run & Results Page presents a unified interface for monitoring simulation progress and inspecting output artifacts after completion. The page adheres to the same GUI standards as the Configuration Page, including consistent typography, navigation controls, and error-display conventions. Users can view run metadata (configuration hash, timestamps, runtime duration), track execution status, and visualize results through dynamically rendered time-series plots. Snapshot images generated by the Simulation Engine are listed or previewed in a standardized display panel, and export functions. Users may rerun simulations, open output directories, or copy configuration data using clearly labeled controls with tooltips and keyboard shortcuts.

## 8.2 Software Interfaces

SI-1: **C++/CUDA simulation engine**: The PySide6 GUI interfaces with the external C++/CUDA Simulation Engine by generating a Parameters.txt file, launching the engine as a separate OS process, and passing the necessary input and output paths as command-line arguments. The engine reads the parameter file, executes the viral agent-based model on the GPU, and writes CSV outputs, optional spatial snapshots, and logs into a designated results directory. The GUI monitors the engine's exit code and captures stdout/stderr to provide status updates and error reporting to the user.

SI-2: **File-based Results store (Local Filesystem)**: The File-Based Results Store serves as the shared data exchange layer between the GUI and the Simulation Engine, consisting of local filesystem artifacts such as parameter files, CSV outputs, images, and logs. The GUI writes run configurations into per-run folders, and the Simulation Engine populates those folders with results. After execution, the GUI reads CSVs and snapshots for visualization, allows users to browse or export results, and supports reloading historical runs by scanning directories for valid output patterns.

SI-3: **Linux Operating System:** The Viral ABM Interface relies on Linux OS services for file I/O, process creation, and integration with the desktop environment. The GUI manages run directories and uses OS process-management facilities (e.g., subprocess or Qt wrappers) to execute and track the Simulation Engine. The GUI may also invoke OS utilities such as opening folders in the default file manager. All interactions must conform to TCU lab security, permissions, and deployment policies.

SI-5: **Python Runtime and GUI/Plotting Libraries**: The PySide6 GUI is built on a Python runtime and uses Python libraries such as PySide6 for UI components, NumPy/Pandas for CSV parsing, and Matplotlib for visualization. These libraries enable the GUI to load simulation data, compute basic statistics, and render plots, tables, and images. Although internal to the GUI, these dependencies must be installed and version-compatible in the lab environment to ensure responsive performance, correct rendering, and graceful handling of malformed or incomplete result files.

## 8.3 API Document

This project does not involve any API.

### 8.4 Hardware Interfaces

HI-1: GPU Hardware: The Viral Agent-Based Model GUI shall run on Linux workstations equipped with NVIDIA GPUs capable of executing CUDA-based simulations.

HI-1.1: The software shall require a CUDA-compatible GPU to execute the backend simulation engine.

HI-1.2: The GUI shall communicate with the GPU indirectly through the CUDA/C++ simulation backend; the GUI itself will not issue direct GPU commands.

HI-1.3: The GPU must support the CUDA toolkit version used by the existing agent based model codebase.

### 8.5 Communications Interfaces

This project does not involve any communication interfaces.

# 9. Quality Attributes

## 9.1 Usability

USE-1: The interface shall allow users to configure all simulation parameters (cell counts, infection rate, transmission mode toggles, eclipse duration, infectious duration, etc.) using sliders, dropdowns, or buttons without editing code.

USE-2: 95% of first-time users (undergraduates with no prior exposure) shall be able to successfully run a simulation within 10 minutes, using only the GUI and without external guidance.

USE-3: Users shall be able to load the previous set of parameters (most recent run) with a single interaction (e.g., “Load Previous Settings”).

USE-4: All input fields shall prevent out-of-range or invalid values, reducing user errors and preventing crashes.

USE-5: All displayed plots (cells over time, virus amount) shall include labeled axes, legends, and units to ensure interpretability

## 9.2 Performance

PER-1: For typical experimental sizes (<200k cells), simulations shall begin executing within 3 seconds after the user presses “Run Simulation.”

PER-2: The interface shall respond to user interactions (sliders, toggles, graph selection) within 200 milliseconds to ensure a responsive experience.

PER-3: The system shall generate and render time-series plots within 2 seconds of simulation completion.

PER-4: Saving time-series output files shall complete within 1 second for typical simulation lengths.

## 9.3 Security

SEC-1: The system shall run only on authorized lab machines configured by the TCU biophysics group.

SEC-2: Only authorized users with lab workstation access may run simulations; no anonymous access is permitted.

SEC-3: The system shall store saved simulation data only in designated user-accessible directories and shall not access restricted system files.

## 9.4 Safety

SAF-1: The interface shall warn users when parameter selections fall outside typical biological ranges (e.g., infection rate > 1.0, eclipse duration = 0).

SAF-2: Any hard-to-reverse actions (e.g., clearing parameters, deleting saved outputs) shall require confirmation.

SAF-3: The system shall prevent simulation execution when required parameters are missing or invalid.

## 9.5 Availability

AVL-1: The interface shall be available 98% of scheduled lab workstation hours, excluding OS, GPU driver, or CUDA updates.

AVL-2: The system shall start up within 10 seconds on supported workstations.

AVL-3: The system shall not require internet access to run simulations or generate outputs.

### 9.6 Robustness

ROB-1: If a simulation fails due to invalid parameters or backend errors, the interface shall display a clear error message and preserve the user's previously entered values.

ROB-2: The interface shall gracefully handle missing or corrupted parameter files when users attempt to load previous configs.

ROB-3: The system shall validate all numeric inputs before attempting to run a simulation.

### 9.7 Expandability

MOD-1: Adding a new simulation parameter (e.g., immune response strength, directed motion speed) shall require modifying no more than one configuration file and one UI widget file.

MOD-2: New output plot types shall be integrable without restructuring existing plotting code.

MOD-3: The interface shall store parameter definitions in a structured, editable format (e.g., JSON, YAML, or a config class).

# 10. Deployment

This section describes how the Viral Agent-Based Model Interface will be deployed, operated, and maintained in its intended environment. Because the system integrates a Python-based GUI with a CUDA-based simulation backend, deployment must accommodate both software components and ensure compatibility with the laboratory's GPU workstations. The deployment strategy aims to provide a reliable, reproducible, and user-friendly installation process for researchers and students.

X.1 Deployment Environment

The system will be deployed in three primary environments:

1. Development Environment

- **Used by:** Senior design team
- **Components:**
    - Python 3.x
    - PySide6
    - Required Python packages (NumPy, Matplotlib, etc.)
    - NVIDIA CUDA Toolkit (matching GPU compute capability)
    - C/C++ compiler (e.g., nvcc, gcc)
- **OS:** Linux (Ubuntu), optionally Windows Subsystem for Linux (WSL2)
- **Purpose:** Development, debugging, and integration testing of GUI + backend.

2. Testing/Staging Environment

- **Used by:** Team testers, client graduate researchers
- **Components:**
    - Same as development environment
    - Mirrors GPU hardware configuration in the biology lab
- **Purpose:**
    - Validate correct system behavior
    - Verify parameter validation, file generation, and visualization
    - Ensure the GUI properly launches and monitors real CUDA simulations

3. Production Environment (Lab Workstations)

- **Used by:** TCU graduate researchers, undergraduate students
- **Hardware:**
    - NVIDIA GPU-enabled workstation (e.g., Quadro 4000 or newer)
    - CUDA-compatible GPU drivers
- **Software Requirements:**
    - CUDA Toolkit installed
    - Python runtime + packaged GUI
    - Compiled version of the CUDA simulation backend (program.out)

The GUI will run directly on these machines and execute the backend locally, eliminating the need for remote compute access.

X.2 Deployment Architecture

The system uses a **two-part architecture**:

1. **Frontend (Python / PySide6 GUI)**
    - Packaged as a standalone Python application or pyinstaller-generated executable
    - Handles parameter entry, validation, simulation launch, and visualization
2. **Backend (CUDA ABM Simulation)**
    - Precompiled executable (e.g., program.out) located on the workstation
    - GUI invokes the backend with user-defined parameters
    - Simulation generates output files in designated directories
    - GUI reads these files to display graphs and summaries

Communication is file-based and process-based; no networking or distributed components are required.

X.3 Deployment Procedure

Initial Deployment

1. Install Python 3.x environment on target workstation

2. Install required Python dependencies
3. Install CUDA Toolkit and verify GPU compatibility
4. Place the compiled simulation backend executable in the designated directory
5. Deploy GUI application (Python project folder or packaged executable)
6. Configure file paths in GUI settings (optional)

User-Level Deployment

- Users launch the GUI from the workstation
- GUI allows simulation parameter configuration
- When the user starts a run, the GUI launches the backend and monitors progress

X.4 Update Strategy

- **GUI Updates:**
    - Delivered as updated Python files or packaged executables
    - Version control via Git ensures traceability
- **Backend Updates:**
    - If simulation logic changes (e.g., new fusion model), the biology lab compiles a new binary
    - GUI remains compatible as long as parameter file format remains consistent
- **Parameter Expansion:**
    - New parameters added by researchers can be plugged into the GUI with minimal refactoring

X.5 Rollback Strategy

- Maintain previous working versions of:
    - Python GUI package
    - CUDA backend binary
- If an update introduces defects, revert to last known stable version
- Git history guarantees recoverability of past releases

X.6 Maintenance and Support

- **Routine maintenance** includes:
    - Updating CUDA driver compatibility
    - Verifying GUI dependencies
    - Cleaning or archiving simulation output directories
- **Long-term maintenance** will be handled by graduate researchers or designated lab personnel
- Documentation is provided to assist new lab members with running and updating both components

X.7 Security Considerations

- System runs locally; no network exposure
- User input validation prevents malformed parameters from breaking the backend
- File permissions on lab workstations ensure simulation output cannot be modified by unauthorized users

## 11. Internationalization and Localization Requirements

The Viral Agent-Based Model GUI is intended solely for use within the TCU Biophysics Lab, so only minimal localization support is required. The interface will be provided in English and will use U.S. numerical formatting standards.

## 12. Other Requirements

Because the Viral Agent-Based Model GUI is used solely within the TCU Biophysics Lab for internal research and education, there are no external legal, regulatory, or financial compliance requirements. Installation and configuration are limited to setting up the GUI and ensuring CUDA and GPU drivers are properly installed on designated lab workstations, which will be handled by lab administrators.

## 13. Appendix A

There are no additional materials or supplementary documents to include in the appendix at this time.