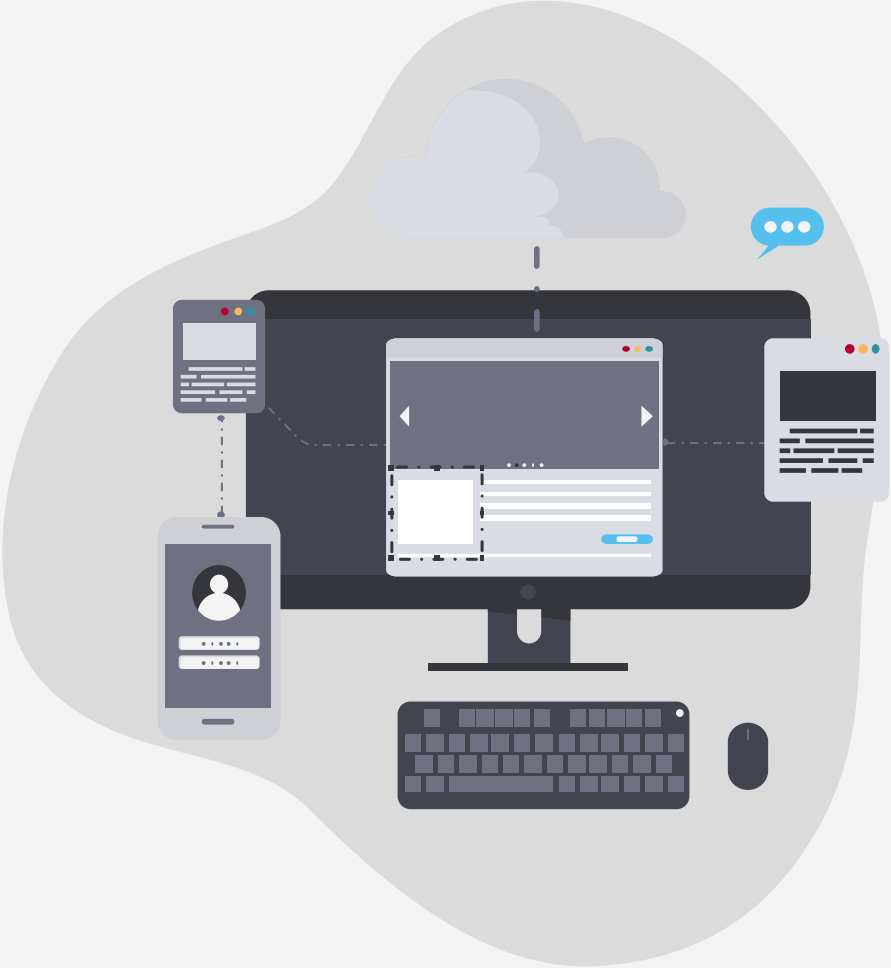
Viral Agent-Based Model Interface

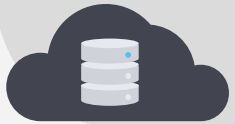
Hebert Alvarez, Gabby Campos, Aqil Dhanani, Derek Le, Bereket Mezgebu, Stefan Saba, Ellion Norwood



Problem Statement

Researchers at TCU are using a CUDA-based viral agent-based model (ABM) to simulate how viruses spread across a 2D layer of biological cells.
But the current model can only be used by editing code, changing parameters manually, and running simulations through the terminal.

Project Goals

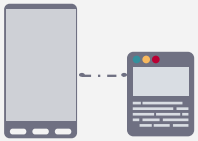


User Interface & Experience

- **User-Friendly GUI:** Design an intuitive interface for simulation parameters without code manipulation.
- **Multi-Model Integration:** Seamlessly switch between and execute two distinct simulation models.
- **Interactive Visualization:** Present results through interactive charts and graphs for viral dynamics.
- **Data Portability:** Built-in capabilities to export simulation data and visualizations.

System Architecture

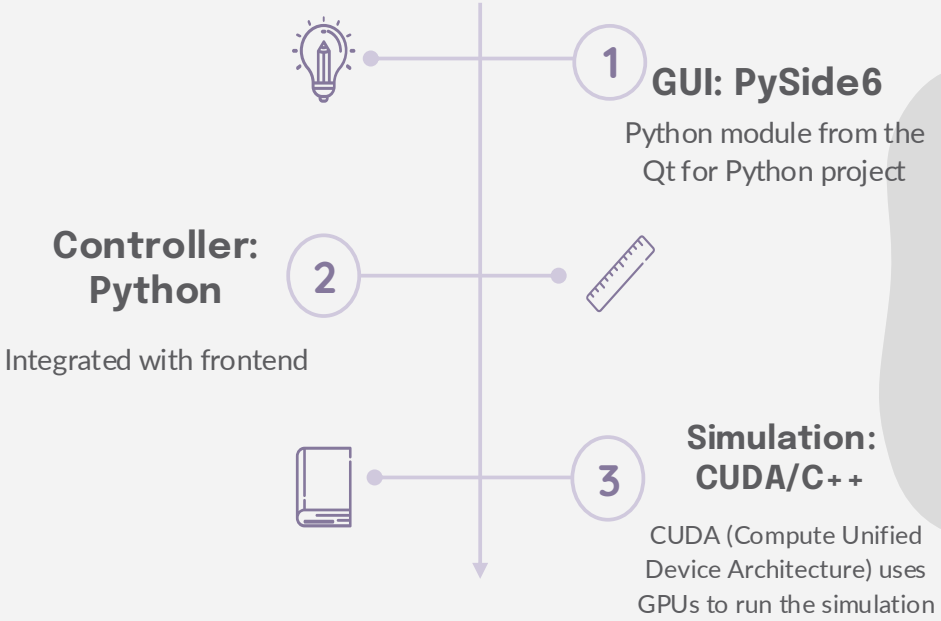
- **Codebase Refactoring:** Reorganize Python and C++/CUDA logic for improved maintainability.
- **Modularization:** Decouple the simulation engine from the GUI for scalability and updates.



Tech Stack

Our system combines a PySide6 desktop interface with a high-performance C++/CUDA backend. The UI allows users to configure parameters, start simulations, and view results, while the backend runs the core agent-based model on the GPU for speed and scalability. The aim was a simple and responsive simulation engine optimized for large populations and complex interactions.

Summary



Tech Stack

Front End + Controller

- Cross-platform Qt framework for building responsive desktop UIs in Python.
- Acts as the front end and the main controller layer: users set parameters, start/stop runs, and visualize outcomes.
- Handles user input validation, parameter presets, and real-time updates from the simulation.
- Lightweight bridge between user actions and backend compute: sends config → waits for results → displays plots/metrics.

Backend Simulation Engine

- Core simulation implemented in high-performance C++ for deterministic, optimized logic (provided by client).
- CUDA accelerates computation steps by running agent interactions on the GPU.
- Designed for scalability: supports large populations, time-step progression, and parallel computations.
- Exposes results to the GUI through slightly modified existing output formats for clean interaction and minimal overhead.

Workflow Diagram

01. User Setup

User starts new simulation

- Loads main dashboard/simulation set-up screen
- [User] configures simulation parameters

02. Configuration

Controller receives configuration

- Validates inputs
- Builds a simulation job
- Sends config to simulation engine

03. Execution

Simulation Engine executes

- Initializes agents & environment
- Applies infection & movement logic
- Records time series snapshots

04. Results

UI displays result page

- Statistics/infection curves/snapshots
- Exports/saves results
- Adjusts and re-run

Viral Agent-Based Model Interface Overview

Overview

A simple clean interface to configure, run, and visualize viral simulations

- Clean MVC design
- Easy parameter input with validation
- Direct connection to CUDA-based engine
- Built-in graphing for results
- Switch between multiple models

Configure Page

Set parameters like infection rate, mobility, population size, and runtime.

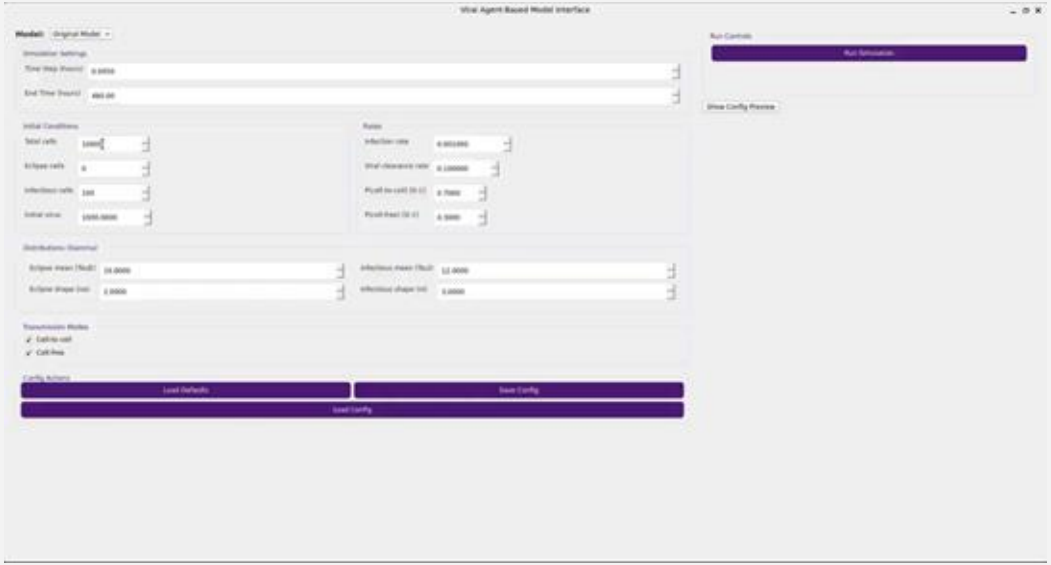
Choose between available simulation models.

Run & Results Page

View simulation progress and logs.

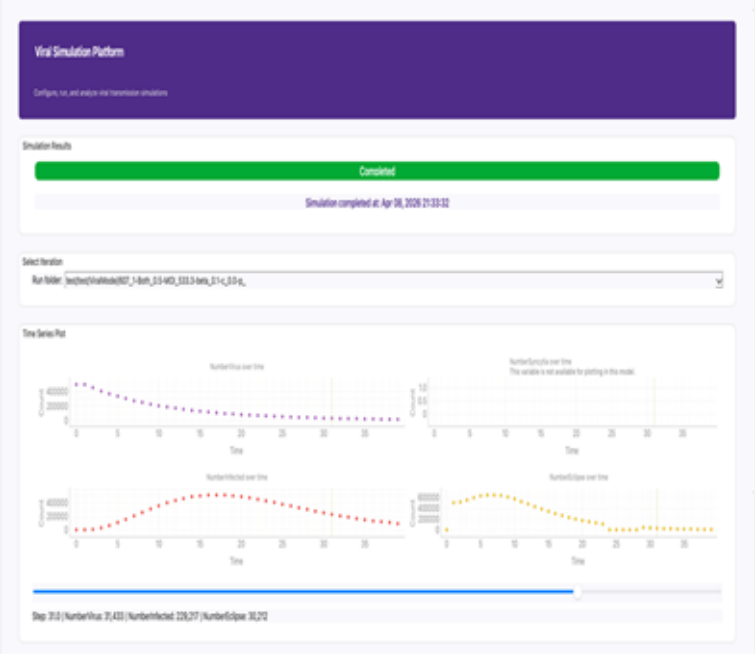
Analyze results with real-time graphs.

Configure Page



- Set initial conditions (total, infected, eclipse cells)
- Adjust rates (infection, clearance, transmission)
- Configure timing distributions (eclipse & infections periods)
- Select transmission mode (cell-to-cell or cell-free)
- Choose model + output settings (plots, file path)
- Save, load, or reset configurations

Results Page



- View run status and completion time
- Select and load simulation runs
- Interactive time-serties graphs (virus, infected, eclipse cells)
- Step through simulation timeline
- Access output data and files
- Export results or rerun with new settings

Demo

What We Learned

Fullstack Development

QT, Foreign Function Interface, Reusable Code

SWE Best Practices

MVC, Multi-Modal Archetecture, State Management

Agile

Self-Paced Stories, Constant Communication, Documentation

Questions?

Thanks!