Viral Agent-Based Model Interface -- User Manual

Part 1: Using the Tool

Create a virtual environment

```
python -m venv abm source abm/bin/activate
```

Install the required dependencies

```
pip install -r requirements.txt
```

Starting the Application

From the project root, run:

```
python3 test.py
```

Important: All terminal commands provided in this manual must be executed from the **project root**.

The project root is the primary directory (e.g., `Viral-ABM-Interface`) containing the `test.py` file. If you are operating within subfolders like `View/` or `runs/`, configuration and setup commands will not function correctly.

How to check: Run `ls` (macOS/Linux) or `dir` (Windows) in your terminal. You are in the project root if both `test.py` and `requirements.txt` are listed.

The window opens on the Run Page, where you configure and launch simulations.

Selecting a Model

Use the Model dropdown at the top-left corner. Two options:

- Original Model -- Viral transmission (cell-to-cell / cell-free infection). Shows Initial Conditions, Rates, Distributions, and Transmission Modes panels.
- Gerg Model -- Syncytia formation (cell fusion). Shows the Fusion Parameters panel instead.

Switching models hides irrelevant fields automatically. Shared settings (Time Step, End Time) stay visible for both.

Configuring Parameters

Shared (both models):

Field	Range	Default
Time Step (hours)	0.0001 -- 1	0.005
End Time (hours)	0 -- 1000	480 (20 days)

Original Model:

Field	Range	Default
Total cells	1 -- 1,000,000	100,000
Eclipse cells	>= 0	0
Infectious cells	>= 0	100

Initial virus (MOI)	0 -- 100,000	1000
Infection rate (beta)	0 -- 10,000	0.001
Viral clearance rate (c)	0 -- 10	0.1
P(cell-to-cell)	0 -- 1	0.7
P(cell-free)	0 -- 1	0.3
Eclipse mean (TauE)	0 -- 1,000	10
Eclipse shape (ne)	0 -- 10,000	2
Infectious mean (TauI)	0 -- 1,000	12
Infectious shape (ni)	0 -- 10,000	3

At least one transmission mode (Cell-to-cell or Cell-free) must be enabled.

Gerg Model:

Field	Range	Default
Enable Cell Fusion	on/off	on
Fusion mean (Tauf)	0 -- 1,000	6
Fusion shape (nf)	0 -- 10,000	30
P(fusion)	0 -- 1	0.2

Running a Simulation

1. Fill in parameters. The Run Simulation button stays disabled until all validation passes. Any errors appear in red text below the button.
2. Click Run Simulation. The UI switches to the Running Page with a spinner.
3. When the CUDA backend finishes, the UI automatically moves to the Post-Simulation Page.
4. To abort mid-run, click Stop Simulation on the Running Page.

Behind the scenes, clicking Run writes `runs/Parameters.txt` at the project root, then compiles and executes the appropriate `.cu` file (`test/test/Viral_Transmission.cu` or `ABM_GERG/Syncytia_Formation.cu`).

Saving and Loading Configurations

- Save Config -- Exports the current model-specific parameters to a `.json` file. Only parameters relevant to the selected model are saved.
- Load Config -- Imports a `.json` file and updates the UI. Unknown keys are ignored; missing keys keep their current values.
- Load Defaults -- Resets all parameters to their defaults while preserving the current model selection.

Config Preview

Click Show Config Preview on the right panel to see a live JSON view of the parameters that will be saved or sent to the backend. This is useful for verifying what the backend will receive.

Part 2: Making Frontend Changes

The frontend is built with PySide6 (Qt for Python). All UI code lives in `View/`.

File Overview

File	Purpose
`test.py`	Entry point. Creates `MainWindow`, wires page transitions via `QStackedWidget`.
`View/RunPage.py`	Parameter configuration page. All input widgets, config state, validation, and simulation launch logic.
`View/RunningPage.py`	"Simulation in progress" page with spinner and stop button.
`View/PostSimulationPage.py`	Results display page shown after simulation completes.
`View/paramValidation.py`	Standalone validation function. Returns a dict of field-level errors.

How RunPage Works Internally

RunPage keeps a flat Python dict called `self._config` that holds every parameter. The UI is built in `_build_ui()`, and the dict stays in sync with the widgets through signal handlers.

Lifecycle of a parameter value:

1. User changes a widget (e.g., a spin box).
2. The widget emits a signal (e.g., `valueChanged`), which calls a handler like `_on_value_changed`.
3. The handler reads the widget's value and writes it into `self._config`.
4. The handler calls `_update_validation()` and `_update_preview()`.

When loading a config (from file or defaults), the reverse happens: `self._config` is updated first, then `_sync_ui_from_config()` pushes values into all widgets. Signals are blocked during sync to prevent cascading updates.

Adding a New Parameter (Step by Step)

Here is the concrete process, using a hypothetical "Diffusion Coefficient" field as an example.

1. Add the key to `self._config` in `__init__`:

```
# in RunPage.__init__, inside self._config = { ... }
"diffusionCoefficient": 0.5,
```

2. Create the widget in `_build_ui`:

Find the group box where the field belongs (e.g., `gb_rates`) and add:

```
self.dsb_diffusion = QtWidgets.QDoubleSpinBox(gb_rates)
self.dsb_diffusion.setObjectName("diffusionCoefficient")
self.dsb_diffusion.setRange(0.0, 100.0)
self.dsb_diffusion.setDecimals(4)
self.dsb_diffusion.setSingleStep(0.01)
self.dsb_diffusion.valueChanged.connect(self._on_value_changed)
form_rates.addRow("Diffusion coefficient", self.dsb_diffusion)
```

Key points:

- Use `QSpinBox` for integers, `QDoubleSpinBox` for floats, `QCheckBox` for booleans.
- Always set `setRange()` to enforce bounds at the widget level.
- Connect the appropriate signal (`valueChanged` for spin boxes, `toggled` for checkboxes) to the matching handler.

3. Read the widget in the signal handler:

In `_on_value_changed`, add:

```
self._config["diffusionCoefficient"] = float(self.dsb_diffusion.value())
```

4. Push config to the widget in `_sync_ui_from_config`:

Add the widget to the `widgets` list (for signal blocking), then set its value in the `try` block:

```
self.dsb_diffusion.setValue(float(c.get("diffusionCoefficient", 0.5)))
```

5. Add it to `_on_load_defaults`:

Inside the defaults dict reset, include the default value:

```
"diffusionCoefficient": 0.5,
```

6. Add it to `_get_model_specific_config`:

Under the correct model branch (Original or Gerg), add:

```
"diffusionCoefficient": c.get("diffusionCoefficient", 0.5),
```

7. Add validation in `View/paramValidation.py`:

```
if not _range_num(c.get("diffusionCoefficient"), 0, 100):
    errors["diffusionCoefficient"] = "Diffusion coefficient must be between 0
and 100"
```

8. If the backend needs this parameter, add it to `_write_backend_params_file`:

Under the correct model branch:

```
f.write(f"diffusionCoefficient:
{float(self._config.get('diffusionCoefficient', 0.5))}\n")
```

Adding a Model-Specific UI Section

To add a new group box that only appears for one model:

1. Create the `QGroupBox` and its child widgets inside `_build_ui`, same as the existing `gb_gerg` block.
2. Store a reference: `self.gb_mygroup = gb_mygroup`.
3. In `_on_model_changed`, toggle its visibility: `self.gb_mygroup.setVisible(...)`.
4. Do the same in `_sync_ui_from_config` so it's correct on config load.

Adding a New Page

1. Create a new file in `View/` (e.g., `View/MyNewPage.py`) with a class extending `QtWidgets.QWidget`.
2. In `test.py`, import it, instantiate it, and add it to `self.stack` with `addWidget`.
3. Wire navigation by connecting signals. For example, to navigate from RunPage to your new page, emit a signal from RunPage and connect it in `MainWindow` to call `self.stack.setCurrentWidget(self.my_new_page)`.

Quick Reference: Common Widget Types

Widget	Use for	Key signal
`QSpinBox`	Integer input	`valueChanged`
`QDoubleSpinBox`	Float input	`valueChanged`
`QCheckBox`	Boolean toggle	`toggled`
`QComboBox`	Dropdown selection	`currentIndexChanged`
`QPushButton`	Action button	`clicked`
`QLineEdit`	Text input	`textChanged`

Styling

All widget styles are defined in a single `setStyleSheet()` call at the top of `_build_ui`. The color scheme uses purple tones (`#4d1979` primary, `#e9d5ff` borders, `#faf9ff` background). To change colors or spacing, edit that stylesheet string.