MiMICS User Guide

Introduction to MiMICS

Multi-scale Model of Metabolism In Cellular Systems, abbreviated MiMICS, is an extendable computational framework executed in Python and Java to simulate metabolism in 2D and 3D microbial communities. MiMICS couples a genome-scale metabolic network reconstruction (GENRE) with the established platform Hybrid Automata Library (HAL)[1], which contains an agent-based model, and a continuum-scale reaction-diffusion model. Individual agents can represent single-cell bacteria or bacteria populations that exist 2D or 3D world, of which the dimensions can be defined by the user.

A key feature of MiMICS is the user's ability to incorporate multiple -omics integrated GENREs, which can represent unique intracellular metabolic states that differ in predicted parameter values passed to the extracellular models. To generate multiple -omics integrated GENREs, we recommend algorithms such as RiPTIDe[2] or GiMME[3]. Individual agents can decide which metabolic model state to execute based on mechanistic rules input by the user.

At each simulation time step, each MiMICS sub-model is performed to update agent properties and metabolite concentrations. For each agent, the biomass and metabolite concentration from the continuum-scale grid corresponding to the agent's location is converted to a metabolite uptake flux used to constrain the agent's GENRE. Constraint-based flux-balance analysis is used to optimize each agent's GENRE to predict a biomass growth rate, as well as metabolite secretion and uptake fluxes. The biomass growth rate is passed to the ABM to update an agent's biomass. Metabolite secretion and uptake fluxes are passed to the in the continuum-scale reaction-diffusion model to update the metabolite concentrations. In the ABM, bacteria agents can perform behaviors like cell division, motility, and cell shoving mechanics. Simulation outputs such as agent locations, agent intracellular metabolic fluxes and metabolite concentrations can be provided at each simulation time step.

Overview of MiMICS framework structure

Relevant attribute values of each agent are passed between Java and Python MiMICS models using a Py4J Gateway Server, which is first initialized by the user (Figure 1). Using the Py4J Gateway Server, a MiMICS Python file initializes agents and metabolite concentrations in MiMICS. Each agent's metabolic model is optimized in Python using the Python package COBRApy and parallel computing (Figure 1).

Using the Py4J Gateway Server, Python runs and calls upon methods in the *MIMICS* Java class to diffuse metabolites and run agent methods (Figure 1). To perform a set of agent behaviors, the *MIMICS* Java class accesses individual agent methods defined in the *Cell3D* Java agent class. All Java classes (outlined in pink in Figure 1) can be compiled into a MIMICS.jar file to run MiMICS on a High Performance Computing (HPC) system.

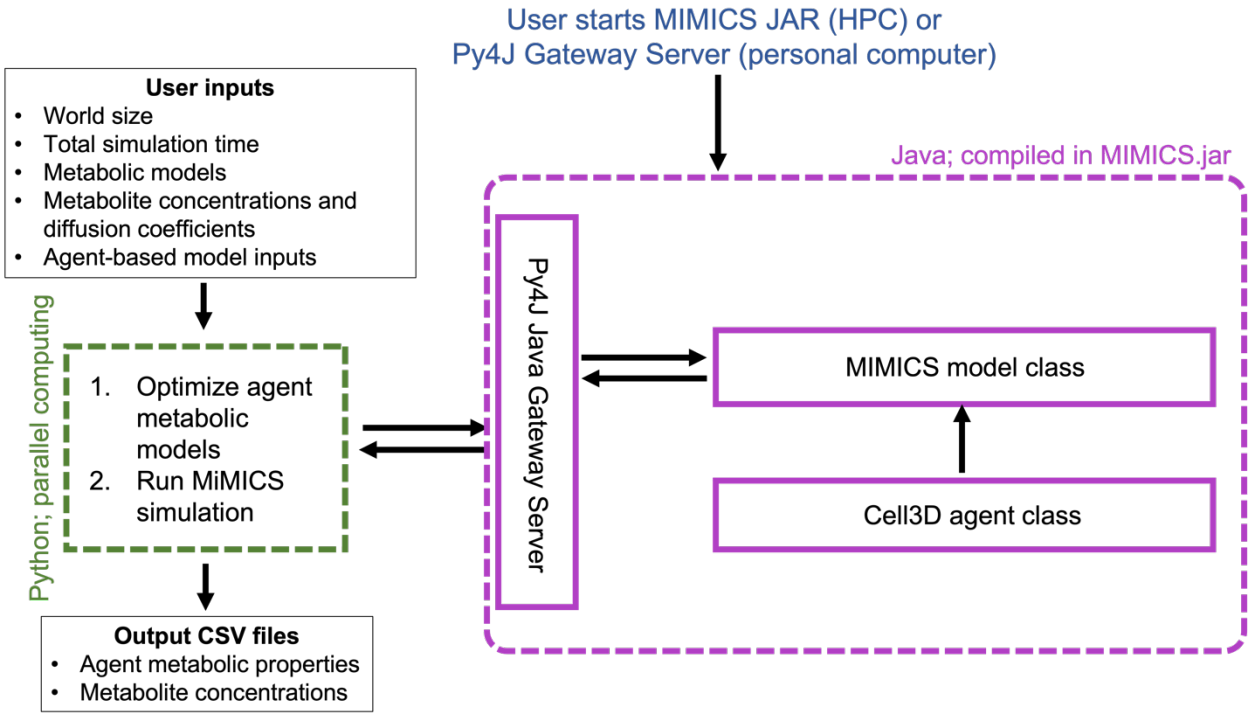

Figure 1. Overview of MiMICS framework structure. MiMICS code based in Python and Java outlined in green and pink, respectively. Simulation inputs are defined and imported in Python. A Py4J Gateway Server, initialized by the user, is required to pass attribute values for each agent between Python and Java models. Python optimizes the agent metabolic models, and calls upon agent-based model and metabolite diffusion Java methods to run the MiMICS simulation. The Java classes outlined in pink are used to compile a MIMICS JAR file that can be used to run MiMICS on a High Performance Computing (HPC) system.

Hybrid Automata Library (HAL) is a published Java library that contains methods to execute the agent-based model and a reaction-diffusion model[1]. MiMICS implements the agent class and metabolite reaction-diffusion methods, the latter solved with partial differential equations (PDEs), in the HAL library. As recommended by HAL, MiMICS uses

IntelliJ IDEA as the IDE to edit and run Java code, and build a compiled MIMICS.jar file. More detailed description of HAL's applications, methods, and tutorials can be found in the HAL publication and in the HAL User Guide.

Each agent's metabolic network model was optimized using the COBRApy Python package. To improve MiMICS runtime, MiMICS uses the Multiprocessing Python package that facilitates parallel computing to asynchronously optimize the agent's metabolic models across multiple central processing units (CPUs). The Py4J python program sets up a Py4J Gateway Server, which is used to pass information between the HAL (Java) and metabolic model (Python) models.

Computational Specifications
Java v.15.0.2
Python v.3.9
Py4J v.0.10.9.1
IntelliJ IDEA v.2020.3.2 Community Edition
HAL v.1.1.0

Overview of MiMICS simulation files

1. *MIMICS.java*: contains the *Cell3D* agent class and *MIMICS* class. Contains methods to simulate the agent-based model and reaction-diffusion model. Contains methods to save MiMICS agent and metabolite concentration outputs. Executed in Java.
2. *MIMICS_gateway.java*: contains methods to start the Py4J Gateway Server, which interfaces MiMICS files written in Java and Python languages. Also contains methods to automatically start the MiMICS Python file, *agents_FBA.py,* if desired. Executed in Java.
3. *agents_FBA.py*: optimizes each agent's metabolic model. Contains functions to initialize the MiMICS agent and metabolite grids, pass agent information between the metabolic model (Python) and HAL (Java) using the Py4J Gateway Server, and run agent-based model and metabolite diffusion methods using the Py4J Gateway Server. Executed in python.

For MiMICS simulations on a High Performance Computing system, a slurm script, was used, named *A_MIMICS_PS.slurm,* to submit simulation jobs.

Installation of MiMICS

Download the MIMICS source code here: https://github.com/tracykuper/mimics.

IntelliJ IDEA is a useful developer environment to edit and run Java and Python code. Download IntelliJ IDEA. Download HAL source code following HAL's instructions, the

download link located in the HAL User Guide, which contains the HAL method library. Set up the HAL library in a new IntelliJ IDEA project according to the instructions in the HAL User Guide. Name the new project *'MIMICS-master'*.

One change is required to make in the HAL library code for MiMICS to run. Within the HAL library, navigate to HAL > GridsAndAgents > AgentsGrid3D. Within the AgentsGrid3D class, change the return method of the GetAgentsRadApprox() method from *void* to *public void*.

Right-click on the *'MIMICS-master'* project to create a new package named *'MIMICS'.* Add the files *MIMICS.java*, *MIMICS_gateway.java*, and *agents_FBA.py* into the '*MIMICS'* package folder.

Configure a Python Interpreter and add Py4J in the IntelliJ IDEA project.
Because MiMICS is written partially in Python, a Python interpreter must be added to the IntelliJ IDEA project. Adding a Python interpreter to an IntelliJ IDEA project is outlined here. Briefly, install the Python Plugin in IntelliJ IDEA. Next, navigate to *'File'* > *'Project Structure'* > *'SDKs'* > Click *'+'* > Select *'Add Python SDK'* > Select *'System Interpreter'* > In the *'Interpreter'* text box, select the folder where the Python virtual environment exists > Click *'OK'* > Click *'Apply'*.

Ensure the necessary Python packages are installed, which include cobra, py4j, numpy, multiprocessing, sys, pandas, math, and time that are all used in MiMICS. To add Python packages, navigate to *'File'* > *'Project Structure'* > *'SDKs'* > Click on the Python SDK > *'Packages'* > Click *'+'* > Install the necessary packages.

Set the Python interpreter in the *'MIMICS-master'* project. Navigate to *'File'* > *'Project Structure'* > *'Modules'* > Right click on you're the project name *'MIMICS-master'* > Click *'+'* > *'Python'* > Set the *'Python Interpreter'* to the Python virtual environment in the *'MIMICS-master'* project folder.

Description of MiMICS Python code, *agents_FBA.py*

The python file, *agents_FBA.py*, optimizes each agent's metabolic model, and executes the agent-based model (ABM) and reaction-diffusion model. Python runs each of these models for a defined number of simulation time steps set by the user. A Py4J Gateway server is used to run the agent-based model and reaction-diffusion model from Python, and pass parameter values of each agent between Python and Java.

Python initializes access to Py4J Gateway Server, designated by a variable named *gateway* in the main function (*if __name__ == '__main__':*) of *agents_FBA.py,* using the following code:

```
gateway = JavaGateway(); # INITIALIZE THE PY4J JAVA GATEWAY SERVER
```

To pass attribute values of all agents, and run the agent-based model and reaction diffusion model, Python calls Py4J Gateway methods defined in the *MIMICS_gateway* Java class (described in the *MIMICS_gateway.java* User Guide section).
To execute these Py4J Gateway methods in Python, use the following code structure: gateway.entry_point.*MIMICS_gateway_method*(), where gateway.entry_point is used to access the Py4J Gateway Server entry point, and *MIMICS_gateway_method*() is the name of the desired Py4J method defined in *MIMICS_gateway.java*.

Py4J method to initialize MiMICS ABM and metabolite world

In the *agents_FBA.py* main function, the MiMICS agent-based model world and metabolite PDE grids are initialized by executing the *run_model0()* Py4J method, which is defined in *MIMICS_gateway.java*. In the code shown below, *run_model0()* requires inputs that are imported by the user, described further in the MiMICS User Guide section *MiMICS user inputs*.

```
# INITIALIZE ABM AND METABOLITE PDE GRIDS
gateway.entry_point.run_model0(int(xdim), int(ydim), int(zdim),
int(initial_num_agents), initial_biomass, max_biomass,int(num_met_gas),
int(num_met_carbon), initial_gas_concentrations_java,
initial_carbon_concentrations_java);
print('Agent-based model and reaction-diffusion model initialized')
```

Running MiMICS for multiple simulation time steps

In *agents_FBA.py*, the Python function *run_MIMICS()* is used to iterate solving the agent-based model, the reaction-diffusion model, and each agent's metabolic model for a desired number of simulation time steps.
In the code shown below, *run_MIMICS()* requires inputs that are imported by the user, described further in the MiMICS User Guide section *MiMICS user inputs*.

```
## RUN MIMICS SIMULATION FOR DESIRED SIMULATION TIME STEPS
run_MIMICS(ncpus,int(num_dt),models,media,rxns,int(job_num),
num_met_gas,initial_gas_concentrations_java, D_gas_java, num_gas_step,
num_met_carbon,initial_carbon_concentrations_java,
D_carbon_java,num_carbon_step,metabolite_ids,v_patch, dt_rxn,
dt_growth,initial_biomass,max_biomass,dead_state,output_dir);
print('MiMICS simulation finished')
```

PY4J method to run agent-based model

Within the Python function *run_MIMICS(),* the *run_ABM()* Py4J method is first executed, which runs the agent-based model. In the code shown below,

run_ABM() requires inputs imported by the user, described further in the MiMICS User Guide section *MiMICS user inputs*.

```
gateway.entry_point.run_ABM(initial_biomass,max_biomass,int(dead_state)) ## RUN AGENT-BASED MODEL
```

Py4J methods to pass agent attribute values from HAL (Java) to metabolic models (Python)

Next, within the Python function *run_MIMICS()*, multiple Py4J methods are called which pass arrays of agent attribute values, including agent biomass, index, metabolic state, and the agent's local metabolite concentrations from Java to Python. This agent information passed from Java to Python is used to define and optimize each agent's metabolic models in Python. A brief description of the Py4J methods to pass agent information from Java to Python is provided below:

- *getIndexFromHal():* Py4J method to pass a 1D array of each agent's index value from Java to Python. Indexed according to the agent's index value. Index is an integer agent attribute.
- *getBiomassFromHal():* Py4J method to pass a 1D array of each agent's biomass value from Java to Python. Indexed according to the agent's index value. Biomass is in units of 1 x 10^{14} grams.
- *getMetabolicStateFromHal():* Py4J method to pass a 1D array of each agent's metabolic state value from Java to Python. Indexed according to the agent's index value. Metabolic state is an integer agent attribute.
- *getPatchFromHal_All():* Py4J method to pass a 2D array of each agent's local metabolite concentration values from Java to Python. Rows correspond to each agent, which are indexed according to the agent's index value. Columns correspond to metabolites. Gaseous metabolites are indexed first, followed by carbon metabolites. Metabolite concentrations are in units of mM.

This source code block obtained from the Python function *run_MIMICS()* shows Python executing the respective Py4J methods to obtain arrays containing each agents' biomass, index, local metabolite concentrations, and metabolic state attribute values.

```
## GET VALUES OF AGENT ATTRIBUTES FROM JAVA
biomass_values = list(gateway.entry_point.getBiomassFromHal()) ## GET BIOMASS OF EACH FROM ABM
biomass_values = [x / 1e14 for x in biomass_values] ## CONVERT BIOMASS TO GRAMS
pos_count = len(biomass_values)  ## GET NUMBER OF AGENTS
index_values = list(gateway.entry_point.getIndexFromHal()) ## GET INDEX OF EACH AGENT FROM ABM
```

```
patch_values_from_java =
np.array(gateway.entry_point.getPatchFromHal_All(int(num_met_gas),int(num_met
_carbon))) ## GET METABOLITE CONCENTRATIONS AT EACH AGENT'S LOCATION
metabolic_states = list(gateway.entry_point.getMetabolicStateFromHal()) ##
GET METABOLIC STATE OF EACH AGENT FROM ABM
```

Multiprocessing of agent metabolic model optimization in Python

Next, in the python function *run_MIMICS(),* the Python function *run_GENRE()* is called, which optimizes an agent's metabolic model. Python's multiprocessing, using the python function *pool.starmap_async()*, is used to apply *run_GENRE()* in parallel for each agent, which optimizes each agent's metabolic model in parallel across multiple central processing units (CPUs). For a large number of agents (> 1,000), this multiprocessing method can improve MiMICS computational runtime. The number of CPUs is defined in the variable *ncpus,* described further in the section *MiMICS user inputs*.

Arrays containing each agent's metabolic model state, biomass, index, and metabolite concentrations, contained in the *items* list variable, and are passed as inputs into the multiprocessing *run_GENRE()* function, shown below. Indexing of agent parameter values contained in the multiprocessing input arrays corresponds to the index values assigned to the agents. Only values of agents in an alive metabolic state are passed to the *run_GENRE()* multiprocessing function. Other inputs to *run_GENRE()* are input by the user, described further in section *MiMICS user inputs*.

The outputs from the *run_GENRE()* multiprocessing function, contained in the *results* variable, contains arrays of each agent's index, updated biomass, updated growth rate, updated local metabolite concentrations, and updated metabolic reaction fluxes. Indexing of agent parameter values contained in the *result* arrays corresponds to the index value assigned to the agents. These updated agent parameters are formatted and passed to Java via the Py4J Gateway Server to update the agent-based model and reaction-diffusion metabolite model.

```
## FORMAT AGENT INFORMATION FOR MULTIPROCESSING OF AGENT METABOLIC MODELS
a = np.array((biomass_values,metabolite_concentrations, index_values,
[models]*pos_count,
metabolic_states,[rxns]*pos_count,[metabolite_ids]*pos_count,[v_patch]*pos_co
unt,[dt_rxn]*pos_count,[dt_growth]*pos_count),dtype=object).T
items = list(map(tuple, a))

## PERFORM MULTIPROCESSING OPTIMIZATION OF AGENT METABOLIC MODELS
try:
    results = pool.starmap_async(run_GENRE,
items,callback=accumulateResults).get()
except Exception as e: # RAISE EXCEPTION
    sarray[0]="Multiprocessing FAILED"
    gateway.entry_point.Print_Phrase(sarray[0])
```

```
    sarray[0]=str(e)
    gateway.entry_point.Print_Phrase(sarray[0])
```

Optimization of an agent's metabolic model in Python

Within the *run_GENRE()* Python function, an individual agent's metabolic model state is first assigned, defined by the *model* variable:

```
model = models[metabolic_state] ## ASSIGN METABOLIC MODEL STATE TO AGENT
```

The lower bounds of exchange fluxes of the agent's metabolic model are constrained on the agent's locally available metabolite fluxes. The agent's local metabolite concentrations and metabolic model exchange flux are converted with the following equation:

$$f_M = \frac{c_M * v_{patch}}{dt_{rxn} * b_0} \text{ (Equation 1)}$$

Where f_M is the metabolite flux (mmol/(g *hr), c_M is the metabolite concentration (mM), v_{patch} is the patch volume (L), dt_{rxn} is the metabolite uptake time step, and b_0 is the agent's initial biomass (g).

The agent's metabolic model is optimized for the biomass synthesis reaction with COBRApy's flux-balance analysis. The predicted biomass synthesis rate was used as a growth rate to update the agent's biomass using an exponential growth rate:

$$b = b_0 \exp\left[\mu * dt_{growth}\right] \text{ (Equation 2)}$$

Where b is the agent's updated biomass (g), b_0 is the agent's initial biomass (g), μ is the growth rate informed from the optimized metabolic model (hr^{-1}) and dt_{growth} is the growth time step (hr).

In addition, in the *run_GENRE()* Python function, the predicted exchange metabolite fluxes from the agent's metabolic model are converted to metabolite concentrations (Equation 1) to update the agent's local extracellular metabolite concentrations:

$$c_f = c_0 + \Delta c \text{ (Equation 3)}$$

where c_f is the updated metabolite concentration at the agent's location (mM), c_0 is the initial metabolite concentration at the agent's location (mM), and Δc is the metabolite concentration consumed or secreted by the agent (mM).

Next, in the *run_GENRE()* Python function, the reaction fluxes predicted from the optimized agent's metabolic model are obtained and passed as an output from the *run_GENRE()* function. The list of reaction IDs to save fluxes for are defined by the user, described in the section *MiMICS user inputs.*

The *run_GENRE()* Python function outputs an individual agent's updated biomass, index, growth rate, updated metabolite concentrations local to the agent, and reaction fluxes predicted by the agent's metabolic model.

Py4J methods to pass agent attribute values from metabolic models (Python) to HAL (Java)

Next, within the Python function *run_MIMICS()*, multiple Py4J methods are called to pass arrays of agent attribute values, including agent biomass, growth rate, and the agent's local metabolite concentrations from Python to Java. This agent information passed from Python to Java is used to update agent attribute values in the agent-based model, and update metabolite concentrations in the reaction-diffusion model. A brief description of the Py4J methods to pass agent information from Python to Java is provided below:

- *setGrowthRateFromPython():* Py4J method to pass a 1D array of each agent's growth rate from Python to Java. Indexed according to the agent's index value. Growth rate is in units of hr^{-1}. Used to update agent growth rate attribute in agent-based model. This method assigns a dead metabolic state attribute value to an agent if the agent's growth rate is 0 hr^{-1}.
- *setBiomassFromPython():* Py4J method to pass a 1D array of each agent's biomass from Python to Java. Indexed according to the agent's index. Biomass is in units of 1×10^{14} grams. Used to update an agent's biomass attribute value in agent-based model.
- *setPatchFromPython():* Py4J method to pass a 2D array of each agent's local metabolite concentration from Python to Java. Rows correspond to each agent, which are indexed according to the agent's index. Columns correspond to metabolites. Gas metabolites

are indexed first, followed by carbon metabolites. Metabolite concentrations are in units of mM. Used to update metabolite concentrations in the respective metabolite PDE Grid.

Of note, arrays passed from Python to Java must be formatted and passed in a 1D or 2D *jvm* array, which can easily be read in Java.

The below code demonstrates Python passing arrays of updated agent attribute values to Java, including arrays containing each agent's updated growth rate, updated biomass, and updated metabolite concentrations where each agent is located.

```
## SEND AGENT'S UPDATED GROWTH RATE, BIOMASS, AND METABOLITE CONCENTRATIONS
TO JAVA
gateway.entry_point.setGrowthRateFromPython(growth_rates_total,
index_values_for_java)
gateway.entry_point.setBiomassFromPython(new_biomass_total,
index_values_for_java)
gateway.entry_point.setPatchFromPython(int(num_met_gas),
int(num_met_carbon),patch_value_for_java,index_values_for_java)
```

Diffuse metabolites

Next, within the Python function *run_MIMICS()*, Python calls upon the Py4J method *Diffuse_Metabolites()* to perform metabolite diffusion. In the code shown below, *Diffuse_Metabolites()* requires inputs imported by the user, described further in section *MiMICS user inputs*.

```
## RUN METABOLITE DIFFUSION
gateway.entry_point.Diffuse_Metabolites(int(num_met_gas),initial_gas_concentr
ations, D_gas,int(num_gas_step),int(num_met_carbon),
initial_carbon_concentrations, D_carbon, int(num_carbon_step))
```

Save MiMICS simulation outputs

Lastly, within the Python function *run_MIMICS()*, Python calls upon the Py4J methods to save simulation outputs at the current simulation time step. The Py4J method *Save_cell_info()* saves the attribute values of each agent. The Py4J method *Save_met_info()* saves the metabolite concentrations at each grid location in the PDE Grid. Further explanation of the output files is described further in section *Description of MIMICS Java class, MIMICS.java.*

In this code shown below, *job_num* is the variable that contains an integer of the simulation job input by the user, described further in section *MiMICS user inputs*, and *t* is the simulation time step.

Note in this example code, *Save_met_info()* was only run for desired simulation time points, defined by the list variable *times_save*. If the metabolite PDE Grid is very large, we recommend only saving metabolite concentrations for a small set of desired time points to avoid very large datasets.

```
# SAVE MODEL OUTPUTS: AGENT ATTRIBUTE VALUES AND METABOLITE CONCENTRATION
INFORMATION
if t == 0:
    gateway.Save_cell_info(int(t), int(job_num),output_dir)
if  t >0:
    gateway.Save_cell_info(int(t), int(job_num),output_dir)
times_save=[0,25,50,91,92,93,94,95,96,97,98,99] # DEFINE TIMES TO SAVE
METABOLITE CONCENTRATIONS
if  t in times_save:
    gateway.Save_met_info(int(t), int(job_num),output_dir)
```

In addition, each agent's predicted metabolic reaction flux values are saved as a CSV file directly from Python, named '*rxn_fluxes#.csv*', where # is the simulation job number.

```
# SAVE MODEL OUTPUTS: AGENT METABOLIC REACTION FLUXES
df_rxns = pd.DataFrame()
df_rxns['time'] = np.ones(len(updated_index))*t
df_rxns['job_num'] = np.ones(len(updated_index))*job_num
df_rxns['cell index'] = updated_index
for r,flux in zip(rxns,rxn_flux_all):
    df_rxns[r] = flux

times_save=[0,25,50,91,92,93,94,95,96,97,98,99]
rxns_flux_output_filename = output_dir+'rxns_flux' + str(job_num)+ '.csv'
if t == 0:
    df_rxns.to_csv(rxns_flux_output_filename)
if t in times_save:
    df_rxns.to_csv(rxns_flux_output_filename,mode = 'a',header = False)
```

Description of MIMICS Gateway Java class, *MIMICS_gateway.java*

MiMICS implements a Py4J Gateway Server to facilitate passing attribute values for each agent between the metabolic model Python environment and HAL Java environment. In addition, Python calls upon Java methods to run the agent-based model and reaction-diffusion model. The *MIMICS_gateway* class, defined in the file *MIMICS_gateway.java*, initializes the Py4J Gateway Server. The *MIMICS_gateway* class contains the Java Py4J methods to pass attribute values for each agent. In addition, the *MIMICS_gateway* class accesses Java methods (defined in the *MIMICS* Java class) to initialize and run the agent-based model and reaction-diffusion model (Figure 1).

Note, in the *main()* method of *MIMICS_gateway.java,* a Py4J Gateway Server is initialized in Java. The *main()* method of *MIMICS_gateway.java* can also automatically run the

Python file *agents_FBA.py,* further described in section *Running MiMICS on a personal computer.*

MIMICS_gateway Java class methods to initialize and run the agent-based model and reaction-diffusion model are defined below:

- *run_model0*(): Py4J method to initialize a *MIMICS* Java class model object. Initializes metabolite PDE Grids for gaseous and carbon metabolites. Sets periodic boundary conditions. Executes MIMICS Java class methods, *Initialize_Random()* and *Initialize_Metabolites(),* to initialize agents and initialize metabolite PDE Grid concentrations, respectively.
- *run_ABM():* Py4J method to run the agent-based model. Executes the *MIMICS* Java class method *StepCells()* to run the agent-based model.
- *Diffuse_Metabolites():* Py4J method to diffuse metabolites. Executes the *MIMICS* Java class method *Gas_Diffuse()* and *Carbon_Diffuse()* to perform diffusion of gaseous and carbon metabolite grids, respectively. Metabolite diffusion is simulated using HAL's alternating direction implicit (ADI) method PDE solver.

MIMICS_gateway Java class methods to pass attribute values of each agent between HAL (Java) and metabolic models (Python) through the Py4J Gateway Server are defined below:

- *getIndexFromHal():* Py4J method to pass a 1D array of each agent's index from Java to Python. Indexed according to the agent's index value. Index is an integer agent attribute.
- *getBiomassFromHal():* Py4J method to pass a 1D array of each agent's biomass value from Java to Python. Indexed according to the agent's index value. Biomass is in units of 1×10^{14} grams.
- *getMetabolicStateFromHal():* Py4J method to pass a 1D array of each agent's metabolic state value from Java to Python. Indexed according to the agent's index value. Metabolic state is an integer agent attribute.
- *getPatchFromHal_All():* Py4J method to pass a 2D array of each agent's local metabolite concentration from Java to Python. Rows correspond to each agent, which are indexed according to the agent's index value. Columns correspond to metabolites. Gas metabolites are indexed first, followed by carbon metabolites. Metabolite concentrations are in units of mM.
- *setGrowthRateFromPython():* Py4J method to pass a 1D array of each agent's growth rate from Python to Java. Indexed according to the agent's index value. Growth rate is in units of hr^{-1}. Used to update agent growth rate attribute in agent-based model. This method assigns a dead metabolic state attribute value to an agent if the agent's growth rate is 0 hr^{-1}.

- *setBiomassFromPython():* Py4J method to pass a 1D array of each agent's biomass from Python to Java. Indexed according to the agent's index value. Biomass is in units of 1 x 10^{14} grams. Used to update agent biomass attribute value in agent-based model.
- *setPatchFromPython():* Py4J method to pass a 2D array of each agent's local metabolite concentration from Python to Java. Rows correspond to each agent, which are indexed according to the agent's index value. Columns correspond to metabolites. Gas metabolites are indexed first, followed by carbon metabolites. Metabolite concentrations are in units of mM. Used to update metabolite concentrations in the respective metabolite PDE Grid.

MIMICS_gateway Java class methods, that save simulation outputs at each time step are defined below:

- *Save_met_info():* Py4J method to execute the *MIMICS* Java class method, *Save_Met_Info(),* which saves metabolite grid concentrations.
- *Save_cell_info():* Py4J method to execute the *MIMICS* Java class method, *Save_Cell_Info(),* which saves the values of agent attributes for each agent.

Description of MIMICS Java class, *MIMICS.java*

The *MIMICS* Java class, defined in the file *MIMICS.java*, defines methods for extracellular metabolite initialization and diffusion, agent initialization and execution of agent class methods, and saving simulation outputs. The *MIMICS* Java class methods are described here. Necessary inputs for these methods are described further in section *MiMICS user inputs.*

Initialize and run metabolite reaction-diffusion model

The metabolite reaction-diffusion model methods used in the *MIMICS* Java class were constructed using the HAL reaction-diffusion metabolite model library. Detailed information about these methods can be found in the HAL User Guide or in HAL tutorial files[1].

- *Initialize_Metabolites(): MIMICS* method to initialize metabolite concentrations in the respective metabolite PDE Grid.
- *Gas_Diffuse(): MIMICS* method to diffuse gaseous metabolites concentrations in the respective metabolite PDE Grid. Assumes constant metabolite concentrations in the aqueous phase outside of the biofilm region.
- *Carbon_Diffuse(): MIMICS* method to diffuse carbon metabolites concentrations in the respective metabolite PDE Grid. Assumes constant metabolite concentrations in the aqueous phase outside of the biofilm region.

Initialize and run agent-based model

The agent-based model methods used in the *MIMICS* Java class were constructed using the HAL agent-based model library. Detailed information about these methods can be found in the HAL User Guide or in HAL tutorial files[1].

- *Initialize_Random(): MIMICS* method to initialize a random distribution of agents in the agent-based model world. Agents are initialized with random x and y coordinates at the z = 0 μm surface. Initializes the values of attributes for the agents.
- *StepCells(): MIMICS* method to perform agent class methods for each agent. Agent class methods are described further in section *Description of MIMICS Java class, MIMICS.java: MiMICS agent class methods.*

Saving MiMICS simulation outputs

The *MIMICS* Java class contains methods to save simulation outputs of agent attribute values and metabolite concentrations at each time point, described in this section.

- *Save_Cell_Info(): MIMICS* method to save each agent's x, y, z coordinates (units: patch location), index, metabolic state, biomass (units: grams), growth rate (units: hr^{-1}), and metabolite concentration at the agent's location. The simulation time step is listed in the *'time'* column. The simulation job number is listed in the *'job_num'* column. The output filename from this method is '*agent properties#.csv'*, where the # is the simulation job number. Rows correspond to each agent and columns correspond the agent attribute. If *patch_scale* was not set to 1 in the user inputs (described in section *MiMICS user inputs*), x,y,z coordinates values will need to be multiplied by the *patch_scale* value. Additional parameter values to be saved should be added by the user in this Java method.
- *Save_Met_Info(): MIMICS* method to save information about the extracellular metabolite concentrations in each of the metabolite grid locations. Metabolite concentrations are in units of mM. The simulation time step is listed in the *'time'* column. The simulation job number is listed in the *'job_num'* column. The output filename from this method is *'met_grid#.csv'*, where the # is the simulation job number. Rows correspond to each x,y,z coordinate value in the metabolite PDE grid and columns correspond to each metabolite (i.e. oxygen, glucose). Additional metabolite concentrations to save should be added by the user in this Java method.

MiMICS agent class methods

The *MIMICS.java* file contains the agent class methods. The agent class is named *Cell3D*. The agent methods used were constructed using the HAL agent-based model library. Detailed information about these agent methods can be found in the HAL User Guide or in HAL tutorial files[1].

The *Cell3D* class was defined as a *SphericalAgent3D* class in the HAL agent library. In addition to HAL's *SphericalAgent3D* class attributes, a brief description of the *Cell3D* class attributes that were defined in MiMICS are described below:

- *mass:* agent biomass; units: grams
- *forceSum:* sum of forces acting on agent
- *angle:* agent directional angle; units: degrees
- *index:* agent index; integer
- *growth_rate:* agent growth rate; units: hr^{-1}
- *metabolic_state:* agent metabolic state assignment; integer
- *t_switch_0:* agent's recorded time to switch to metabolic state 0; units: minutes
- *t_switch_1:* agent's recorded time to switch to metabolic state 1; units: minutes

Note, the *Cell3D* attributes *metabolic_state*, *t_switch_0*, and *t_switch_1* was used for agents to simulate different metabolic model states. The attribute *metabolic_state* is an integer representing the agent's metabolic state that can be assigned in the agent-based model. For example, an agent's *metabolic_state* = 1 corresponds to metabolic state 1. The attributes *t_switch_0* and *t_switch_1* recorded the time an agent was exposed to a new metabolite environment to switch to metabolic state 0 or metabolic state 1, respectively. Further demonstration of the regulation of an agent's metabolic state attribute value is shown in the section *MiMICS example of P. aeruginosa biofilm metabolic states.*

A brief description of the *Cell3D* class methods is described below:

- *Init(): Cell3D* method that initializes values of agent attributes such as agent biomass, index, growth rate, metabolic state, directional angle, size, and recorded time to switch to a new metabolic state.
- *ForceCalc(), CalcMove(), and MoveDiv(): Cell3D* method that runs HAL's agent mechanical calculations.
- *Biomass_Divide(): Cell3D* method that performs biomass division when an agent's biomass exceeds a maximum biomass threshold. Calls the *Init() Cell3D* method to initialize the attribute values of a daughter agent.

Printing statements from MiMICS

To print statements from MiMICS when simulating on a personal computer, you can print statements directly from Python and Java as normal.

When running on a High Performance Computer, MiMICS does not easily allow for printing statements from Python. If needed, the Py4J method *Print_Phrase()* can be used to pass statements from Python to Java to print from Java.

An example Python code is shown below using the Py4J method *Print_Phrase()*. The *sarray* variable defines a Java-language string in Python, that is passed to Java using the Py4J method *Print_Phrase()* to printed from Java.

```
sarray = gateway.new_array(gateway.jvm.java.lang.String,2) ## STRING FOR
PRINTING FROM JAVA
sarray[0]="ERROR" ## ERROR STATEMENT
gateway.entry_point.Print_Phrase(sarray[0])  ## PRINT ERROR STATEMENT USING
PY4J GATEWAY
```

Running MiMICS on personal computer

For populations of less than 10,000 simulated agents, MiMICS may efficiently run on a personal computer. When simulating on a personal computer, it is recommended to run the uncompiled MiMICS source code using IntelliJ IDEA.

To run the uncompiled MiMICS source code on a personal computer, open *agents_FBA.py, MIMICS_gateway.java*, and *MIMICS.java* in IntelliJ IDEA.

Note, the Py4J Gateway Server (i.e. start the *MiMICS_gateway.java* file) must be running BEFORE running the MiMICS Python file, *agents_FBA.py*. This starting operation ensures the Py4J Gateway Server is open and ready to receive Py4J commands from Python. Two options are available for you to 1) manually start both the MiMICS Java and Python files or 2) only start the MiMICS Java file, which automatically runs the MiMICS Python file. Both options are described below.

Option 1: User manually starts Py4J Gateway Server in Java and manually starts MiMICS Python file

Although perhaps more tedious, manually starting both Java and Python MiMICS files may be advantageous to easily print statements from both Java and Python, especially when troubleshooting errors.

Locate the *main()* method in *MIMICS_gateway.java.* Ensure the *main()* method has commented out the sections shown below.

```
// MIMICS GATEWAY MAIN METHOD
    public static void main(String[] args) throws IOException,
InterruptedException {
        GatewayServer gatewayServer = new GatewayServer(new
MIMICS_gateway()); // INITIALIZE JAVA PY4J GATEWAY SERVER
        gatewayServer.start(); // START JAVA GATEWAY SERVER FOR PYTHON ACCESS
        System.out.println("PY4J Gateway Server Started");
//
//        // USE  CODE BELOW FOR MIMICS GATEWAY SERVER TO AUTOMATICALLY RUN
MIMICS PYTHON FILE (agents_FBA.py)

//        String command = "python agents_FBA.py"; // PYTHON FILE TO OPTIMIZE
EACH AGENT'S GENRE
//        Process p = Runtime.getRuntime().exec(command); // CALL PYTHON FILE
//        p.waitFor();
//
//        // PRINT STATEMENTS CALLED FROM THE PYTHON FILE
//        try (BufferedReader br = new BufferedReader(new
InputStreamReader(p.getInputStream()))) {
//            String line;
//            while ((line = br.readLine()) != null) {
//                System.out.println(line);
//            }
//        }
//        gatewayServer.shutdown(); //SHUT DOWN JAVA GATEWAY SERVER WHEN
MIMICS IS FINISHED
    }
```

Next, start the *MIMICS_gateway.java* file, which will open the Py4J Gateway Sever in Java. ALWAYS ensure that *MIMICS_gateway.java* is running first so that a Py4J Gateway Server can be accessed by Python. Once the *MIMICS_gateway.java* file is running and Py4J Gateway Sever is established, start the *agents_FBA.py* file to connect Python to the Py4J Gateway Server.

Once the number of MiMICS simulation time steps is complete, *agents_FBA.py* will automatically shut down. *MIMICS_gateway.java* and the Py4J Gateway Sever will continue running until shut down manually by the user.

Option 2: User manually starts the Py4J Gateway Server in Java, which automatically runs the MiMICS Python file

Locate the *main()* method in *MIMICS_gateway.java.* Ensure the *main()* method has the uncommented, active code shown below:

```
// MIMICS GATEWAY MAIN METHOD
    public static void main(String[] args) throws IOException,
InterruptedException {
        GatewayServer gatewayServer = new GatewayServer(new
MIMICS_gateway()); // INITIALIZE JAVA PY4J GATEWAY SERVER
        gatewayServer.start(); // START JAVA GATEWAY SERVER FOR PYTHON ACCESS
        System.out.println("PY4J Gateway Server Started");
//
        // USE  CODE BELOW FOR MIMICS GATEWAY SERVER TO AUTOMATICALLY RUN
```

```
MIMICS PYTHON FILE (agents_FBA.py)

        String command = "python agents_FBA.py"; // PYTHON FILE TO OPTIMIZE
EACH AGENT'S GENRE
        Process p = Runtime.getRuntime().exec(command); // CALL PYTHON FILE
        p.waitFor();

        // PRINT STATEMENTS CALLED FROM THE PYTHON FILE
        try (BufferedReader br = new BufferedReader(new
InputStreamReader(p.getInputStream()))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        }
        gatewayServer.shutdown(); //SHUT DOWN JAVA GATEWAY SERVER WHEN MIMICS
IS FINISHED
    }
```

Next, start *MIMICS_gateway.java* file, which will open the Py4J Gateway Sever in Java. The *main()* method in *MIMICS_gateway.java* will automatically run *agents_FBA.py*, which will connect Python to the Py4J Gateway Server and run MiMICS.

Once the number of MiMICS simulation time steps is complete, both *MIMICS_gateway.java* and *agents_FBA.py* will automatically shut down.

Running MiMICS on a High Performance Computer

Running MiMICS on a High Performance Computing (HPC) system can be advantageous to reduce computational runtime. HPC systems can provide multiple computing nodes to simultaneously run multiple MiMICS simulations for replicate simulations or perturbations to input parameter values. In addition, HPC systems can provide large CPU resources for parallel processing of agent metabolic model optimization.

Running MiMICS on a HPC system requires MiMICS Java files be compiled as a JAR file in IntelliJ IDEA.
To compile the MIMICS.jar file in IntelliJ IDEA:

1. Select *'File'* > select *'Project Structure'*
2. Select *'Artifacts'* from the side bar menu > Click *'+'* > select *'JAR'* > select *'From Modules with Dependencies'*
3. In the *'Create Jar From Modules'* window > select *'Main Class'* > select *'Project'* > select the *'MIMICS_gateway'* class > Click *'OK'* to exit.
4. Select *'Apply'* in *'Project Structure'*. Click '*OK'* to exit.

5. In the main toolbar select '*Build'* > select *'Build Artifacts'* > in the *'Action'* menu, select *'Build'*. The MIMICS.jar file will be compiled and saved in the artifacts folder of the project.

More information to compile a .jar file with IntelliJ IDEA can be found here.

A slurm job script can be used to execute the *MIMICS.jar* file in the HPC system. In addition, the slurm script can define the number of desired nodes, number of tasks, number of tasks per node, number of CPUS per tasks, total computational time for one simulation, a result output file, computing partition, and memory for one simulation.

Below is an example of an *A_MIMICS_PS.slurm* job script:

```
#!/bin/bash
#SBATCH --ntasks=2
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=35
#SBATCH --time=0-00:50:00
#SBATCH --output=result%a
#SBATCH --partition=parallel
#SBATCH --mail-type=BEGIN,END

module load anaconda
module load java

export NUM_PROCS=$SLURM_CPUS_PER_TASK
export NUM_ARRAY=$SLURM_ARRAY_TASK_ID

java -jar MIMICS.jar
```

Because a Py4J Gateway Server is set up for each MiMICS simulation, only one MiMICS simulation can be performed on one computing node (i.e. multiple MIMICS simulations **can not** be performed on one computing node). Therefore, the *ntasks-per-node* slurm parameter, which defines the number of MiMICS simulations per computing node, **must be set to one** to avoid an error.

The slurm script also defines the number of CPUs (NUM_PROCS) and the simulation job number (NUM_ARRAY), which are passed to python to define *ncpus* and *job_num* in Python, respectively. Other desired parameters to export from slurm to Python, such as in cases where multiple parameter values are desired to be tested, can be added by the user.

Run this command in the Slurm workload manager to execute the slurm job script and run multiple MiMICS simulations:

```
sbatch --array=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14 A_MIMICS_PS
```

The '--array' is used to define the job number index array. The job index number will be used as the SLURM_ARRAY_TASK_ID and used to define the simulation job number. The slurm script name is A_MIMICS_PS. In this example slurm command, 15 simulations were performed.

MiMICS user inputs

Inputs into the MiMICS source code allows for users to extend the MiMICS framework to simulate metabolic processes in multi-cell communities of interest. As MiMICS incorporates HAL Java methods, we recommend users read the HAL manual to understand the HAL's extendable agent and metabolic grid methods available to users. This section outlines the essential files and parameters the user will input into MiMICS. **All essential areas for user input occur in the main function (*if __name__ == '__main__':*) of the Python file, *'agents_FBA.py',* underneath the commented heading 'USER INPUT'.**

Input the number of central processing units

Multiple central processing units (CPUs) can be used for parallel processing of agent metabolic models.

When MiMICS is run on a personal computer, in the main function in *'agents_FBA.py'*, the user must input the number of central processing units (CPUs) in the integer variable *ncpus*.

When MiMICS is run on a High Performance Computing system, the number of central processing units (CPUs) can be defined in a slurm script, and passed from the slurm script to Python, shown below. More description to import *ncpus* from slurm can be found in section *Running MiMICS on a High Performance Computer.*

```
## SET OR IMPORT NUMBER OF MULTIPROCESSING CORES, AND INITIALIZE
MULTIPROCESSING WORKERS
ncpus = 2
#ncpus=int(os.getenv('NUM_PROCS')) # COMMENT OUT WHEN RUNNING ON HPC SYSTEM.
JOB NUMBER VALUE DEFINED IN MIMICS HPC SLURM JOB FILE.
```

Input the simulation job number

Multiple MiMICS simulations can be run in parallel, where each MiMICS is defined by the simulation job number, *job_num*. Multiple MiMICS simulations are helpful when running multiple MiMICS simulation replicates or perturbations to parameter values. The simulation job number that is used to name MiMICS output files.

When running MiMICS on a personal computer, define the simulation number contained in the *job_num* integer variable. In the source code shown below, the simulation job number was set to 0.

```
## SET OR IMPORT THE SIMULATION JOB NUMBER
job_num = 0 # DEFINE JOB NUMBER HERE WHEN RUNNING MIMICS ON PERSONAL COMPUTER
#job_num=int(os.getenv('NUM_ARRAY')) # COMMENT OUT WHEN RUNNING ON HPC
SYSTEM. JOB NUMBER VALUE DEFINED IN MIMICS HPC SLURM JOB FILE.
```

When MiMICS is run on a High Performance Computing system, *job_num* can be defined a slurm script file, and passed from the slurm script to Python, shown below. More description to import *job_num* from slurm can be found in section *Running MiMICS on a High Performance Computer.*

Input agent-based model parameter values

Essential agent-based model (ABM) parameter values need to be contained in an Excel file. Input the filename of the Excel (XLSX) file as a string in the *abm_parameters_filename* string variable.

```
## DEFINE FILENAME (XLSX) OF ABM PARAMETER VALUES
abm_parameters_filename = 'insert directory + XLSX filename of ABM parameter
values'
```

The ABM parameter file follow the tabulated format shown in Table 1, with equivalent headings and parameter variable names. The ABM parameter file must also contain all values of parameters in the units shown in Table 1.

Table 1. Essential agent-based model parameter inputs into MiMICS.

Parameter name	Parameter description	Parameter value	Unit
xdim	x-dimension length	100	micrometers
ydim	y-dimension length	100	micrometers
zdim	z-dimension height	10	micrometers
patch_scale	length of one patch	1	micrometers
v_patch	volume of a patch unoccupied by an agent	1.00E-16	L
total_sim_time	total simulation time	5.00E-01	hours
time_step	simulation time step	5.00E+00	minutes
dt_growth	biomass growth time scale	5.00E+00	minutes
dt_rxn	metabolite uptake time scale	0.05	seconds
dt_diffuse_carbon	carbon diffusion time scale	0.01	seconds
dt_diffuse_gas	gas diffusion time scale	0.0025	seconds
initial_biomass	initial biomass of an agent	1.00E-12	grams
max_biomass	maximum biomass of an agent	2.00E-12	grams
initial_num_agents	initial number of agents	2	agents
dead_state	metabolic assignment of a dead cell state	3	

The *xdim*, *ydim*, *zdim*, and *patch_scale* parameters are used to define the x,y,z grid dimensions of the ABM world and metabolite PDE grids.

The *total_sim_time* and *time_step* parameters are used to calculate the total number of simulation time steps.

The *v_patch* and *dt_rxn* parameters are used to convert metabolite concentrations and metabolite fluxes (Equation 1). The *dt_growth* parameter is used to calculate the updated biomass of an agent (Equation 2).

The *dt_diffuse_carbon* and *dt_diffuse_gas* parameters are used to calculate the scaled metabolite diffusion coefficients used in HAL's metabolite PDE solver. The HAL platform recommends the scaled diffusion coefficient be less than 1. Therefore, at micron patch length scales where the scaled carbon or gas metabolites can be unstable, we recommend adjusting the time scales separately for carbon and gas metabolites to achieve stable diffusion coefficients. To adjust for these differences in time scales, *dt_rxn*, *dt_diffuse_carbon*, and *dt_diffuse_gas* parameters are used to calculate the number of diffusion time steps for carbon and gas metabolites per metabolite reaction time step. We recommend defining time scales for *dt_rxn*, *dt_diffuse_carbon*, and *dt_diffuse_gas* that allow for stable scaled metabolite diffusion coefficients, but a low number of diffusion time steps to limit computational time.

The *initial_biomass* and *max_biomass* parameters are used to define the range of biomass for initialized agents. The *max_biomass* parameter is used to define the maximum biomass an agent can contain. If an agent exceeds the *max_biomass,* a new daughter agent is placed in the simulation, with the mother agent biomass divided between the mother and daughter agents. The *initial_num_agent* integer parameter initializes the number of agents in the ABM. The *dead_state* parameter represents the integer value to the metabolic state agent attribute indicating the agent is in a dead metabolic state.

Input metabolite media conditions

In the main function in *'agents_FBA.py'*, the user must input the Excel filename (XLSX) containing the information used to define the metabolite conditions to simulate in MiMICS. Define the filename as a string in the *media_filename* string variable.

```
## DEFINE FILENAME (XLSX) OF NUTRIENT MEDIA
media_filename = 'insert directory + XLSX filename of metabolite conditions'
```

The metabolite condition Excel sheet must follow the tabulated format and parameter units shown in Table 2. For each metabolite in the media recipe, define the metabolite name, corresponding ID of the exchange metabolite in the metabolic model, and metabolite concentration (mM units). The metabolite ID is used to constrain the metabolite exchange bounds in the metabolic model. Thus, the metabolite ID should correspond to the exchange metabolite reaction that is in the metabolic model.

Metabolites desired to have simulated partial differential equation (PDE) Grids in HAL must be defined in the columns *'Gas PDE index'* or '*Carbon PDE index'*, respective if the metabolite is a gas or carbon metabolite, and have a diffusion coefficient (unit: cm^2/s) listed in the *'Diffusion coefficient (cm^2/s)'* column. Within the *'Gas PDE index'* and '*Carbon PDE index'* columns, list the desired index order of the metabolite, which will assign the metabolites index in an array of metabolite PDE Grids. For example, in Table 2, only glucose and lactate are carbon metabolites with PDE Grids. In addition, according to the '*Carbon PDE index'*, glucose will be at the zero-index position and lactate will be at the first index position in the array of carbon metabolite PDE Grids.

All metabolites are used to constrain the lower bounds of the metabolite exchange reaction for each agent's metabolic model (Equation 1). For metabolites without PDE Grids, metabolite concentrations are assumed constant, and the concentration listed in *'Metabolite concentration (mM)'* is converted to a flux to constrain each agent's metabolic model (Equation 1). For metabolites with PDE Grids, metabolite concentrations are obtained from the respective metabolite PDE Grid at the location of an agent, converted to a flux (Equation 1) and used to constrain the agent's metabolic model.

Table 2. Organization of metabolite inputs into MiMICS.

Metabolite name	Metabolite ID	Metabolite concentration (mM)	Gas PDE index	Carbon PDE index	Diffusion coefficient (cm^2/s)
Water	EX_cpd00001_e	1000	NA	NA	NA
Glucose	EX_cpd00027_e	3.2	NA	0	0.0000017
Lactate	EX_cpd00159_e	9	NA	1	0.000001
NH4+	EX_cpd00013_e	2.3	NA	NA	NA
SO4	EX_cpd00048_e	0.27	NA	NA	NA
Na+	EX_cpd00971_e	66.6	NA	NA	NA
HPO4	EX_cpd00009_e	2.5	NA	NA	NA
K+	EX_cpd00205_e	15.8	NA	NA	NA
Cl-	EX_cpd00099_e	79.1	NA	NA	NA
Ca2+	EX_cpd00063_e	1.7	NA	NA	NA
Mg2+	EX_cpd00254_e	0.6	NA	NA	NA
Fe(iii)	EX_cpd00021_e	0.0036	NA	NA	NA
H+	EX_cpd00067_e	2.5	NA	NA	NA
O2	EX_cpd00007_e	0.25	0	NA	0.000011
CO2	EX_cpd00011_e	0.0132	1	NA	0.00001

The desired metabolite PDE grids to save should be added in the *MIMICS* Java class methods, *Save_Met_Info()* and *Save_Cell_Info(),* which are described further in the section *Description of MIMICS Java class, MIMICS.java*

Input metabolic model files

Next, the user must input the metabolic model filenames as a string into the string variables '*model0', 'model1',* ….etc. The metabolic model files should be an XML format and are imported using COBRApy. Adjust the number of metabolic model string variables ('*model0', 'model1',* '*model2', 'model3',*….etc.) according to the number of metabolic model states you desire to simulate in MiMICS.

Next, the user must define the desired order of metabolic models in the list variable *metabolic_models_files*. In the case where only metabolic model is desired to simulate in MiMICS, only list the one metabolic model file name in the *metabolic_models_files* list variable. In the case where multiple metabolic models are input into MiMICS, list the metabolic models in the desired order in the *metabolic_models_files* list variable. For an agent to simulate one of the metabolic model states, the agent should have an attribute assigned in the agent-based model that is an integer corresponding to the index of the desired metabolic model in the *metabolic_models_files* list. An example code demonstrating this concept is shown in section *MiMICS example of P. aeruginosa biofilm metabolic states.*

```
## DEFINE FILENAMES (XML) FOR METABOLIC MODELS
model0 = "insert directory + filename of metabolic model 0"
model1 = "insert directory + filename of metabolic model 1"
#model2 = "..."
#model3 = "..."
#model4 = "..."

## DEFINE THE ORDER OF METABOLIC MODELS THAT AGENTS CAN ACCESS VIA THE
AGENT'S INTEGER ATTRIBUTE
metabolic_models_files = [model0,model1]
```

Input list of intracellular reactions to output fluxes for in MiMICS

At each time step, MiMICS outputs the flux values of metabolic reactions from each agent's metabolic model. Reactions to save flux values for need to be defined in an CSV file. Input the file the filename (CSV) containing the desired reaction IDs in the string variable *rxn_ids_filename*.

```
## DEFINE FILENAME (CSV) OF REACTION IDS TO SAVE REACTION FLUX OF EACH AGENT
rxn_ids_filename = 'insert directory + filename of reaction IDs'
```

The reaction IDs need to correspond to reaction IDs in the metabolic model. The CSV file should be formatted with the reaction IDs name listed in one column (example shown below).

RXN ID 1
RXN ID 2
RXN ID 3

We recommend an input of less than 1000 reactions and only IDs of reactions with expected non-zero flux to avoid outputs of large simulation datasets.

Input directories of simulation output files

Input the desired directories for MiMICS simulation outputs files.

```
## DEFINE DIRECTORY FOR SIMULATION OUTPUT FILES
output_dir = 'insert directory for simulation output files'
```

Suggested areas in MiMICS code to customize

Although we have listed the necessary user inputs to run the MiMICS source code, other areas of the MiMICS source code may be altered to fit the user's needs. We recommend reading HAL's manual to review HAL's built-in agent-based model and metabolite PDE Grid methods. Particularly, HAL can simulate in 2D and 3D, which can be adjusted in the MiMICS source code. In addition, HAL offers different types of agent classes and methods to simulate agent behaviors. In particular, the *friction*, *radius*, and *force_scaler SphericalAgent3D()* agent class attributes can be modified in the MiMICS source code, which are used in HAL's agent mechanical force calculations. Custom agent behaviors can be added to the *Cell3D* agent class in *'MIMICS.java'.*

MiMICS can be used to assign the agent's metabolic state attribute in the agent-based model, which can be passed to Python to simulate a unique metabolic model state for an agent. Rules and attributes for agents to regulate a metabolic model state in the agent-based model should be added by the user accordingly. An example of this metabolic model regulation is in the section *MiMICS example of P. aeruginosa biofilm metabolic states.*

To pass different parameter values between Java and Python (i.e. agent attributes, reaction fluxes) that are not currently included in MiMICS, add the corresponding Py4J methods to the *'MIMICS_gateway.java'* file. For example, Py4J methods to pass a *'species'* agent attribute could be used to simulate different microbial species metabolic models.

Metabolic model optimization can be adjusted according to the user's needs, such as functions offered in COBRApy (e.g. different metabolic model solver methods, gene knock-outs, or different optimization targets). Multiple metabolic model states, which can be inputs into MiMICS, can be generated by integration of -omics data, measured at bulk or single-cell levels, into metabolic models with developed algorithms such as RiPTIDe[2] or GiMME[3].

Run MiMICS simulation

Once all inputs and desired code modifications are accounted for, run the simulation following the sections: *Running MiMICS on a High Performance Computer* or *Running MiMICS on a personal computer.*

Description of MiMICS simulation outputs

MiMICS outputs the attribute values of each agent, agent reaction fluxes, and metabolite concentrations at each simulation time point. Detailed description of the functions that save MiMICS simulation outputs are in the section *Description of MIMICS Java class, MIMICS.java.*

1. *Metabolite concentrations.* The metabolite concentrations at each x,y,z coordinate in the metabolite grid are output in a CSV file named *'met_grid#.csv'*, where # defines the simulation job number. The row corresponds to a unique x,y,z location in the metabolite grid. Columns correspond to the x,y,z coordinates and the metabolite concentrations. The method *Save_Met_Info()* in *'MiMICS.java'* saves this metabolite output file. Desired metabolite concentrations should be added by the user in the *Save_Met_Info()* MIMICS class method. The simulation time step is listed in the *'time'* column. The simulation job number is listed in the *'job_num'* column.
2. *Agent attribute values.* The attribute values of each agent are output in a CSV file named *'agent_properties#.csv',* where # defines the simulation job number.

Rows correspond to each agent and columns correspond an agent's attribute. The method *Save_Cell_Info()* in *'MiMICS.java'* saves this output file. The file contains each agent's x,y,z coordinates, index, biomass, growth rate, metabolic state, and metabolite concentrations at the agent's location. The simulation time step is listed in the *'time'* column. The simulation job number is listed in the *'job_num'* column. Other desired values of agent attributes should be added by the user in the *Save_Cell_Info()* MIMICS class method.

3. *Agent metabolic reaction fluxes.* Each agent's metabolic model predicted intracellular and exchange reaction flux values are saved as a CSV file named *'rxns_fluxs#.csv',* where # contains an integer of the simulation job number. Each row corresponds to each agent and the columns contain each agent index and flux value of a metabolic reaction. Reaction IDs are listed in the first row. Reaction fluxes are only saved for agents that are in an active or alive metabolic state (i.e. agents without the dead metabolic state assignment). The simulation time step is listed in the *'time'* column. The simulation job number is listed in the *'job_num'* column. This file is saved directly from Python in the *run_MIMICS()* Python function.

MiMICS example of *P. aeruginosa* biofilm metabolic states

MiMICS has a feature where agents can run a unique metabolic model state by regulating their value of a metabolic model state attribute in the agent-based model. To demonstrate this feature, an example MiMICS simulation of metabolic state regulation in a *Pseudomonas aeruginosa* biofilm is shown in this section. Simulation files for this example can be found on GitHub: https://github.com/tracykuper/mimics.

For this example simulation, the Java classes are named '*MIMICS_gateway_PA'* and *'MIMICS_PA'.* The agent class is named *'Cell3D_PA'.* The python file is named *'agents_FBA_PA.py'.* Simulations were performed across 35 CPUs on the University of Virginia Rivanna High Performance Computing System using the slurm script *A_MIMICS_PS*.

Table 3 shows the agent-based model parameter inputs. Agents were represented as 2 μm *P. aeruginosa* single-cells residing in a 230 μm x 230 μm x 40 μm world. Five agents were randomly initialized at t = 0hrs. The simulation was run for five-minute time steps for a total growth period of ten hours. A growth phase lag time was added.

Table 3. Agent-based model inputs for the *P. aeruginosa* biofilm simulation

Parameter name	Parameter description	Parameter value	Unit
xdim	x-dimension length	230	micrometers
ydim	y-dimension length	230	micrometers
zdim	z-dimension height	40	micrometers
patch_scale	length of one patch	2	micrometers
v_patch	volume of a patch unoccupied by an agent	1.00E-16	L
total_sim_time	total simulation time	1.00E+01	hours
time_step	simulation time step	5.00E+00	minutes
dt_growth	biomass growth time scale	5.00E+00	minutes
dt_rxn	metabolite uptake time scale	0.05	seconds
dt_diffuse_carbon	carbon diffusion time scale	0.01	seconds
dt_diffuse_gas	gas diffusion time scale	0.0025	seconds
initial_biomass	initial biomass of an agent	1.00E-12	grams
max_biomass	maximum biomass of an agent	2.00E-12	grams
initial_num_agents	initial number of agents	5	agents
dead_state	metabolic assignment of a dead cell state	4	
lag_phase	total time of growth lag phase	1.75	hours

Four *P. aeruginosa* metabolic model states were input into MiMICS: aerobic state, denitrification –nitric oxide (NO) secretion state, denitrification +NO secretion state, and oxidative stress state. In the *metabolic_models_files* list variable, the metabolic model states were ordered according to metabolic state agent attribute key (Table 4). For example, agents with a value of 2 for their metabolic state attribute, which will be assigned in the ABM, will use the denitrification +NO secretion metabolic model to simulate metabolism. Agents with a value of 4 for their metabolic state attribute will be considered dead and will not use a metabolic model.

```
## DEFINE FILENAMES (XML) FOR METABOLIC MODELS
model0 = "aerobic_state.xml"
model1 = "denitrification_state.xml"
model2 = "denitrification_NO_state.xml"
model3 = "oxidative_stress_state.xml"

## DEFINE THE ORDER OF METABOLIC MODELS THAT AGENTS CAN ACCESS VIA THE
AGENT'S INTEGER ATTRIBUTE
metabolic_models_files = [model0,model1,model2,model3]
```

Table 4. Key to map metabolic state agent attribute to metabolic model state

Metabolic state #	Metabolic Model State	Oxygen	Nitric oxide	Stochastic parameter, R_n
0	Aerobic	$[O_2] \geq [O_2]_t$	$[NO] < [NO]_t$	No relation
1	Denitrification - NO secretion	$[O_2] < [O_2]_t$	$[NO] < [NO]_t$	$R_n \geq 0.94$
2	Denitrification +NO secretion	$[O_2] < [O_2]_t$	$[NO] < [NO]_t$	$R_n < 0.06$
3	Oxidative stress	No relation	$[NO] \geq [NO]_t$	No relation
4	Dead cell (no metabolic model)	No relation	No relation	No relation

Agent attributes to record the time an agent was exposed to a new metabolite condition corresponding to a metabolic state were added for each active metabolic state attribute value.

```
double t_switch_0; // AGENT TRACKING TIME TO SWITCH TO METABOLIC STATE 0
double t_switch_1; // AGENT TRACKING TIME TO SWITCH TO METABOLIC STATE 1
double t_switch_2; // AGENT TRACKING TIME TO SWITCH TO METABOLIC STATE 2
double t_switch_3; // AGENT TRACKING TIME TO SWITCH TO METABOLIC STATE 3
```

In the agent-based model (ABM), agents were assigned the value of their metabolic state attribute based on biologically-releveant rules defined in the *Change_metabolic_state() Cell3D_PA* method. *P. aeruginosa* can alter its intracellular metabolism in low oxygen and toxic nitric oxide (NO) biofilm microenvironments. Accordingly, the *Change_metabolic_state()* method provides a set of mechanistic rules to represent how *P. aeruginosa* cells shift their metabolic state in response to their local extracellular oxygen and nitric oxide concentrations. Both stochastic mechanisms and metabolite (i.e. oxygen and nitric oxide) sensing mechanisms were used for agents to regulate their metabolic state attribute (Table 4). This metabolic state attribute value was used to assign a corresponding metabolic model state to the agent in Python (Table 4). The *Change_metabolic_state() Cell3D_PA* method was added for each agent to run in the *StepCells() MIMICS_PA* method.

```
// METHOD TO ASSIGN AGENT METABOLIC STATE ATTRIBUTE BASED ON MECHANISTIC
RULES
public void Change_metabolic_state() {
    // GET METABOLITE CONCENTRATIONS AT THE AGENT'S LOCATION
    double o2_here = G.gas_metabolites.get(0).Get(this.Isq());
    double NO_here = G.gas_metabolites.get(1).Get(this.Isq());

    // DEFINE METABOLITE CONCENTRATION THRESHOLDS TO ASSIGN AN AGENT
METABOLIC STATE ATTRIBUTE
    double o2_lim = 0.21; // OXYGEN THRESHOLD, UNITS: MILLIMOLAR
```

```
    double no_lim = 1; // NITRIC OXIDE, UNITS: MICROMOLAR

    // SWITCH TO AEROBIC STATE IF NO IS LOW AND O2 IS HIGH.
    if (NO_here*1e3<no_lim & o2_here>=o2_lim){
        this.t_switch_0 = this.t_switch_0 + 5; // TRACK TIME THE AGENT
EXPERIENCES THIS EXTRACELLULAR METABOLITE CONDITION
        if (this.t_switch_0 >= 10) {
            this.metabolic_state = 0;
            this.t_switch_1 =0; this.t_switch_2 =0; this.t_switch_3 =0;  //
SET RECORDED TIME TO ZERO FOR OTHER METABOLIC STATES
        }
    }

    // SWITCH TO DENITRIFICATION STATE +/- NO SECRETION IF NO IS LOW AND O2
IS LOW
    if (o2_here<o2_lim & NO_here*1e3<no_lim){
        this.t_switch_1 = this.t_switch_1 + 5; // TRACK TIME THE AGENT
EXPERIENCES THIS EXTRACELLULAR METABOLITE CONDITION
        if (this.t_switch_1 >= 10) {
            double Rn= Math.random(); // STOCHASTIC METABOLIC STATE PARAMETER
            // STOCHASTIC ASSIGNMENT OF +/- NITRIC OXIDE SECRETION
DENITRIFICATION STATE
            if (Rn >= 0.06) { this.metabolic_state = 1;} // DENITRIFICATION
STATE WITHOUT NO SECRETION
            if (Rn < 0.06) { this.metabolic_state = 2;} // DENITRIFICATION
STATE WITH NO SECRETION
            this.t_switch_0 =0; this.t_switch_3 =0; // SET RECORDED TIME TO
ZERO FOR OTHER METABOLIC STATES
        }
    }

    // SWITCH TO OXIDATIVE STRESS STATE IF NO IS HIGH
    if (NO_here*1e3>=no_lim ){
        this.t_switch_3 = this.t_switch_3 + 5; // TRACK TIME THE AGENT
EXPERIENCES THIS EXTRACELLULAR METABOLITE CONDITION
        if (this.t_switch_3 >= 10) {
            this.metabolic_state = 3;
            this.t_switch_0 =0; this.t_switch_1 =0; this.t_switch_2 =0; //
SET RECORDED TIME TO ZERO FOR OTHER METABOLIC STATES
        }
    }
}
```

The *Cell3D_PA* method *Pili_Move()* was added to simulate surface motility.

Metabolite inputs are shown in Table 5. Synthetic Cystic Fibrosis Sputum Medium (SCFM) was used as the media recipe. PDE3D grids were set for oxygen, nitrate, nitric oxide (NO), and glucose. Oxygen, nitrate, and glucose concentrations were used to constrain the agent's metabolic model lower bound of the respective exchange reaction. For agents that did not run the denitrification +NO secretion metabolic model, the NO concentration was used to constrain the agent's metabolic model NO exchange upper bound, simulating cytotoxic NO transport into the cell. Oxygen, nitrate, NO, and glucose

were updated to save in the *Save_Met_Info()* and *Save_Cell_Info() MIMICS_PA* class methods.

Table 5. Metabolite inputs to *P. aeruginosa* biofilm simulation

Metabolite name	Metabolite ID	Metabolite concentration (mM)	Gas PDE index	Carbon PDE index	Diffusion coefficient (cm^2/s)
Water	EX_cpd00001_e	1000	NA	NA	NA
Glucose	EX_cpd00027_e	3.2	NA	0	0.000001675
Lactate	EX_cpd00159_e	9	NA	NA	NA
Alanine	EX_cpd00035_e	1.8	NA	NA	NA
Arginine	EX_cpd00051_e	0.3	NA	NA	NA
Aspartate	EX_cpd00041_e	0.8	NA	NA	NA
Cysteine	EX_cpd00084_e	0.2	NA	NA	NA
Glutamic acid - glutamate	EX_cpd00023_e	1.5	NA	NA	NA
Glycine	EX_cpd00033_e	1.2	NA	NA	NA
histidine	EX_cpd00119_e	0.5	NA	NA	NA
isoleucine	EX_cpd00322_e	1.1	NA	NA	NA
leucine	EX_cpd00107_e	1.6	NA	NA	NA
lysine	EX_cpd00039_e	2.1	NA	NA	NA
methionine	EX_cpd00060_e	0.6	NA	NA	NA
phenylalanine	EX_cpd00066_e	0.5	NA	NA	NA
proline	EX_cpd00129_e	1.7	NA	NA	NA
ornthinine	EX_cpd00064_e	0.7	NA	NA	NA
serine	EX_cpd00054_e	1.4	NA	NA	NA
threonine	EX_cpd00161_e	1	NA	NA	NA
tryptophan	EX_cpd00065_e	0.01	NA	NA	NA
tyrosine	EX_cpd00069_e	0.8	NA	NA	NA
valine	EX_cpd00156_e	1.1	NA	NA	NA
NH4+	EX_cpd00013_e	2.3	NA	NA	NA
SO4	EX_cpd00048_e	0.27	NA	NA	NA
Na+	EX_cpd00971_e	66.6	NA	NA	NA
HPO4	EX_cpd00009_e	2.5	NA	NA	NA
K+	EX_cpd00205_e	15.8	NA	NA	NA
Cl-	EX_cpd00099_e	79.1	NA	NA	NA
Ca2+	EX_cpd00063_e	1.7	NA	NA	NA
Mg2+	EX_cpd00254_e	0.6	NA	NA	NA
Fe(iii)	EX_cpd00021_e	0.0036	NA	NA	NA

H+	EX_cpd00067_e	2.5	NA	NA	NA
O2	EX_cpd00007_e	0.25	0	NA	0.0000114
CO2	EX_cpd00011_e	0.0132	NA	NA	NA
N2	EX_cpd00528_e	0.48	NA	NA	NA
Nitrate	EX_cpd00209_e	0.35	2	NA	0.0000114
Nitric Oxide	EX_cpd00418_e	0	1	NA	0.0000126

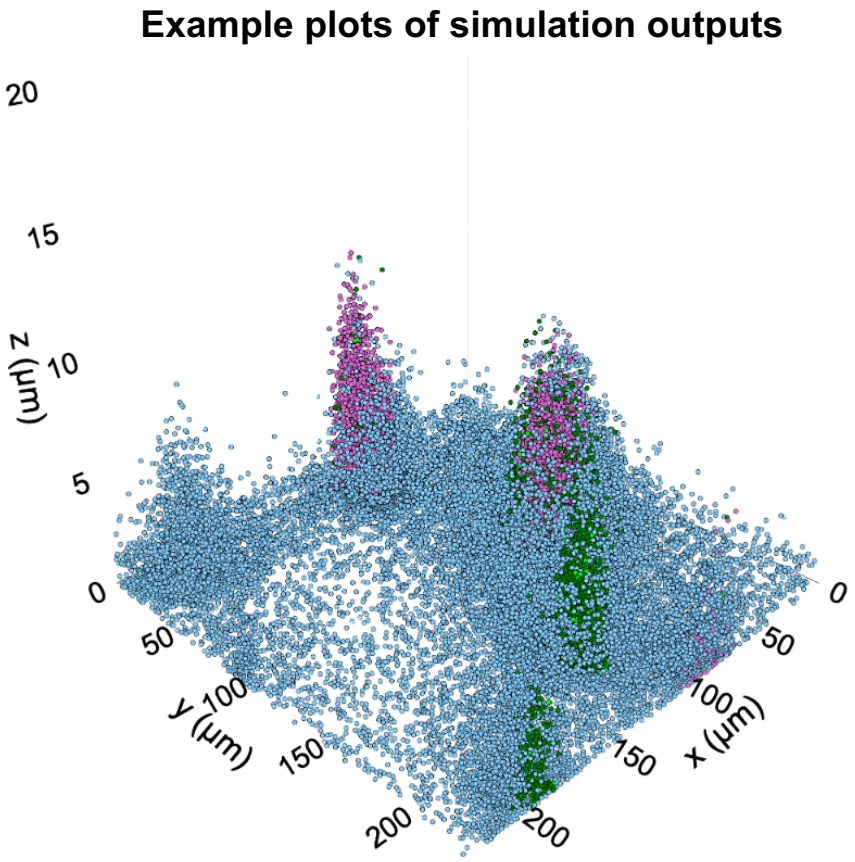

Figure 2. Representative 3D visualization of *P. aeruginosa* biofilm MiMICS output. Agents are colored to their metabolic state attribute (i.e. blue = aerobic, dark green = denitrification -NO, light green = denitrification +NO, pink = oxidative stress).

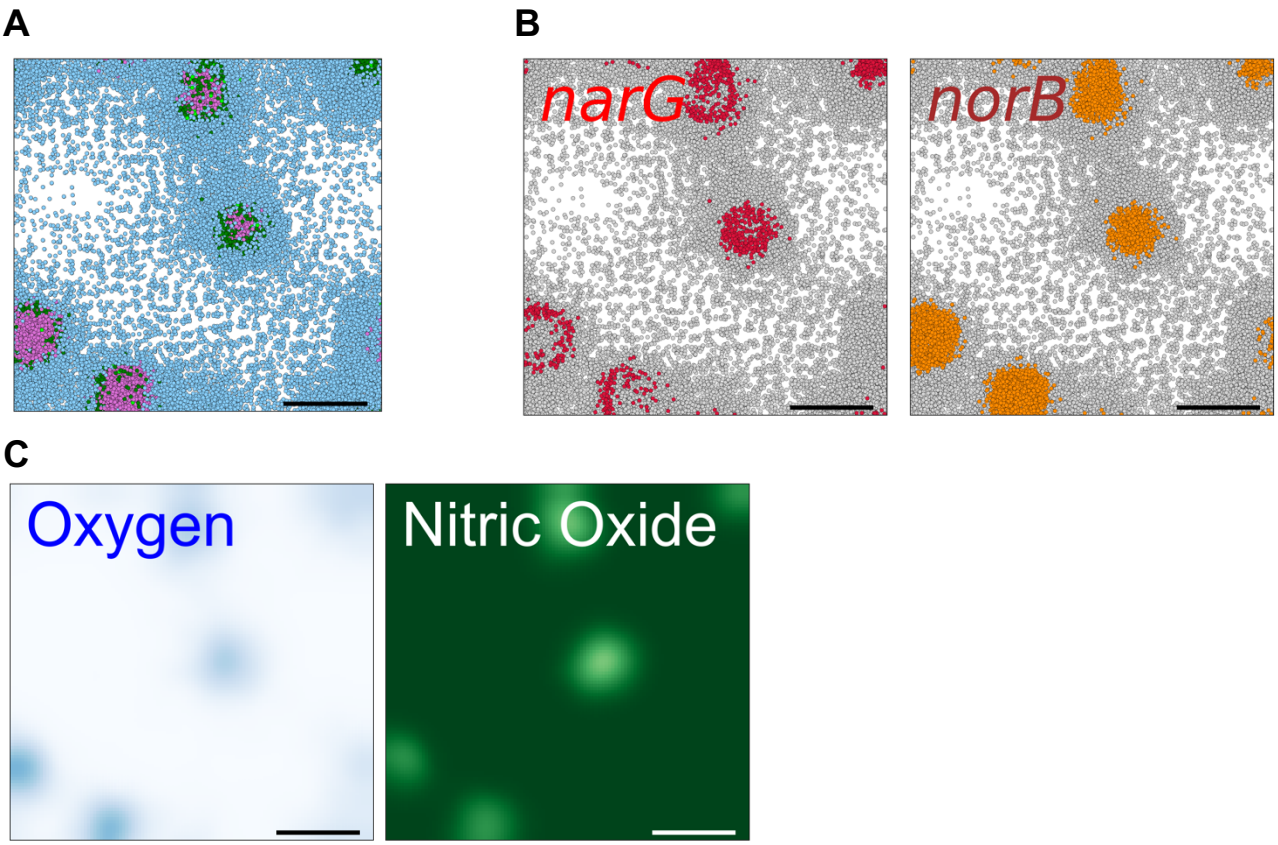

Figure 3. Representative 2D visualization of *P. aeruginosa* biofilm MiMICS output. (A) Agents are colored to their metabolic state attribute (i.e. blue = aerobic, dark green = denitrification -NO, light green = denitrification +NO, pink = oxidative stress). (B) Agents colored according to active flux through an intracellular reaction encoded by a gene. (C) Oxygen and nitric oxide concentrations in biofilm. Scale bar represents 50 μm.

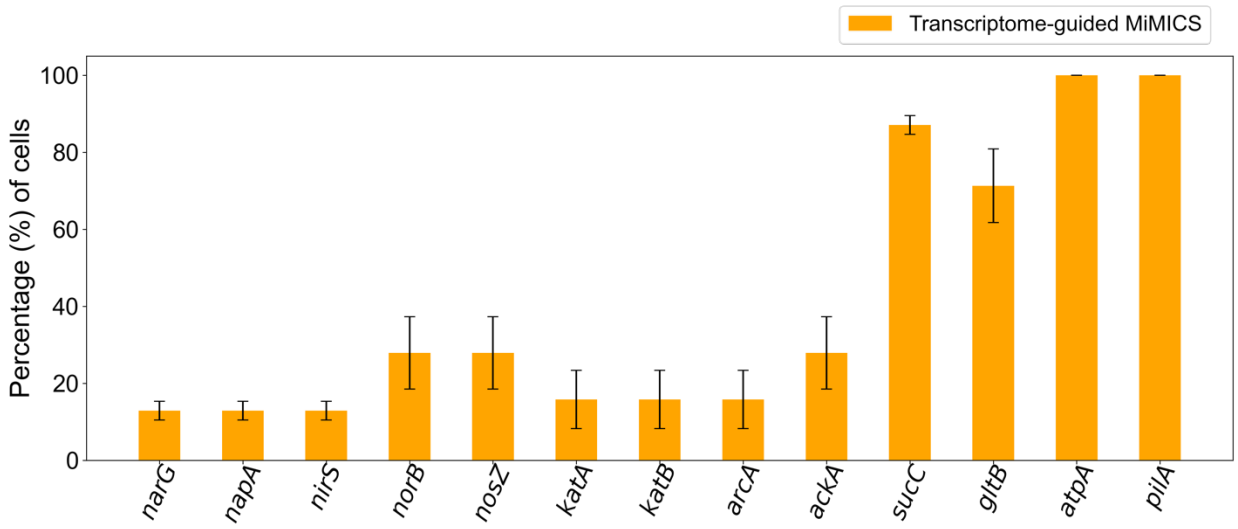

Figure 4. Percentage of total *P. aeruginosa* agents with active flux through an intracellular reaction encoded by a gene. Average and standard deviation reported from 6 replicate MiMICS simulations at the final simulation time point.

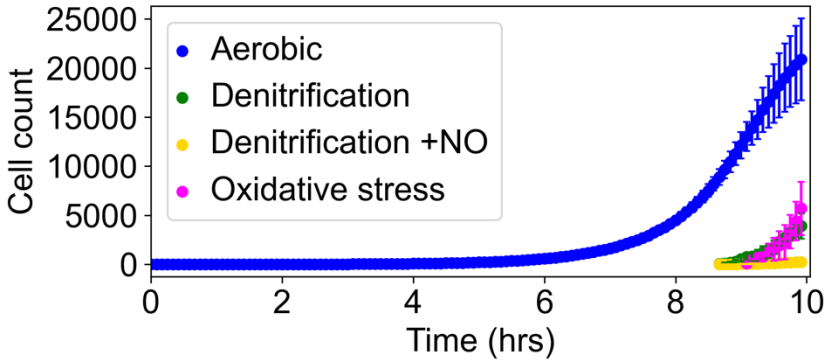

Figure 5. Dynamics of the number of *P. aeruginosa* agents classified by metabolic state attribute over time. Average and standard deviation reported from 6 replicate MiMICS simulations.

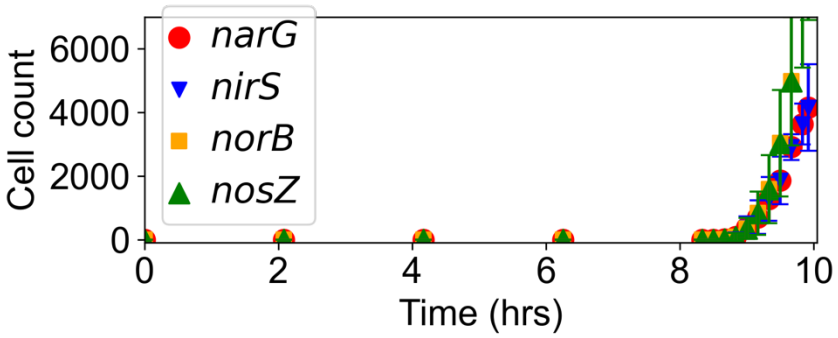

Figure 6. Dynamics of the number of *P. aeruginosa* agents with active flux through a reaction encoded by a gene. Average and standard deviation reported from 6 replicate MiMICS simulations.

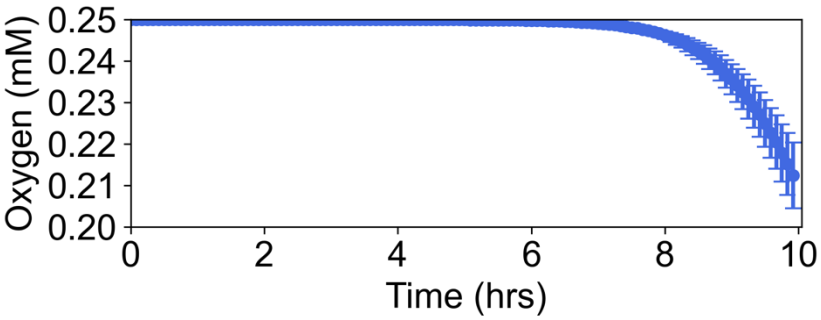

Figure 7. Dynamics of oxygen concentration in the biofilm. Average and standard deviation reported from 6 replicate MiMICS simulations.

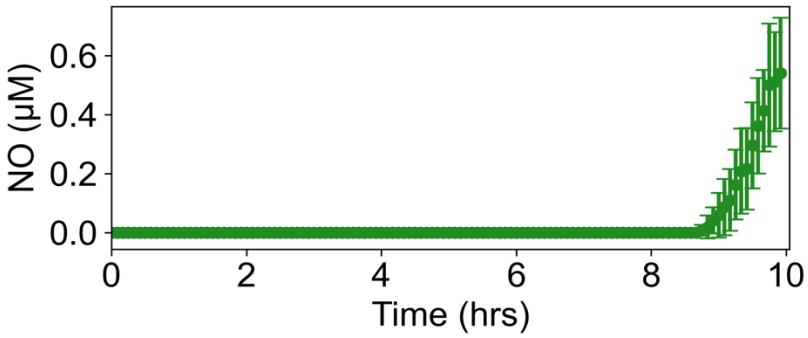

Figure 8. Dynamics of nitric oxide concentration in the biofilm. Average and standard deviation reported from 6 replicate MiMICS simulations.

REFERENCES

1. Bravo, R. R. *et al.* Hybrid Automata Library: A flexible platform for hybrid modeling with real-time visualization. *PLoS Comput Biol* **16**, e1007635 (2020).
2. Jenior, M. L., Moutinho, T. J., Dougherty, B. V. & Papin, J. A. Transcriptome-guided parsimonious flux analysis improves predictions with metabolic networks in complex environments. *PLoS Comput Biol* **16**, e1007099 (2020).
3. Becker, S. A. & Palsson, B. O. Context-Specific Metabolic Networks Are Consistent with Experiments. *PLoS Comput Biol* **4**, e1000082 (2008).