Probabilistic Deep Sequence Models for Bayesian Optimization (A11)

Ethan Cao
etcao@ucsd.edu

Manav Jairam
mjairam@ucsd.edu

Alaa Fadhlallah
afadhlallah@ucsd.edu

Liam Manatt
lmanatt@ucsd.edu

Kyla Park
dap006@ucsd.edu

Anirudh Indraganti
aindraga@ucsd.edu

Yian Ma
yianma.ucsd@gmail.com

Rose Yu
roseyu@ucsd.edu

Abstract

Recent work introduced the Global Epidemic and Mobility (GLEAM) Model that simulates the spread of infectious diseases on a global scale using high-resolution demographic and mobility data and stochastic models of disease transmission. It allows users to simulate disease on a global scale to help develop public health strategies to minimize the impact of epidemics. Recent work has also shown that AI models significantly accelerate traditional epidemic simulations, reducing run-time from weeks to days, which is especially crucial during 'war-time' emergencies. In this paper, we introduce a novel Epidemic Engine Cloud-based API to bridge the GLEAM simulator with AI surrogate models to minimize computational cost and simulation time. The API aims to support real-time, scalable epidemic modeling and simulation querying

Code: https://github.com/Rose-STL-Lab/GLEAM-API

1 Introduction

Epidemics and pandemics pose complex challenges to global health, requiring robust modeling and forecasting tools to mitigate their impact. The COVID-19 pandemic starkly illustrated the critical need for real-time, data-driven epidemic simulation models that enable public health organizations to make informed, rapid-response decisions. Traditional epidemic simulators, such as the GLEAM, play a central role in this effort by simulating disease spread across regions using data on population movement and disease transmission rates. While effective for large-scale scenario testing, these models are limited in their adaptability for real-time applications.

1.1 Literature Review

1.1.1 GLEaM epidemic simulator and GCP services

The GLEAM epidemic simulator was initially proposed in Balcan et al 2010 paper, "Modeling the spatial spread of infectious diseases: The GLobal Epidemic and Mobility computational model." (Balcan et al. 2010) They describe a stochastic model for predicting and simulating the spread of an infectious disease. It has been deployed to predict both influenza and Covid 19, providing invaluable insight into how disease spreads and how countermeasures will affect that spread. This GLEAM simulator uses countless data sources coupled with compartmentalized predictive methods to achieve its impressive accuracy. Unfortunately, this comes at an expensive computational cost. To circumvent this, a Deep Bayesian Active Learning surrogate model has been proposed. This model would require active querying of GLEAM simulation runs. To provide this service, we will levy cloud-hosted APIs to dynamically allocate compute resources. Literature on cloud-integrated APIs highlights the growing need for scalability and reliability when handling high volumes of concurrent API requests. Studies emphasize that cloud environments like Google Cloud include auto-scaling and load balancing. This allows for the computing power of the service to match the demand. It is significantly cheaper to do these tasks on the cloud versus on-premise servers. For epidemic modeling, cloud integration allows simulators to run at scale, handling multiple concurrent API calls from AI models and providing real-time data streams. Another critical advantage of cloud-based API deployment is enhanced security. Many cloud providers offer built-in encryption, authentication, and authorization mechanisms. (Petrillo et al. 2016)

As described in "Modeling the Spatial Spread of Infectious Diseases: The Global Epidemic and Mobility Computational Model" by Balcan et al, the GLEaM model consists of population layer, mobility layer, and disease model. The population layer utilizes high-resolution population data from the "Gridded Population of the World" dataset and "Global Urban-Rural Mapping" project. By designating a subset of the population to a grid of cells at varying resolution levels, the model captures spatial heterogeneity, such as differences in population distribution between urban and rural areas. These data allow the model to account for population dynamics, allowing for more accurate disease spread modeling. The

mobility layer is expected to take parameters, such as worldwide airport network data and commuting network data. These parameters enable the mobility layer to account for both long-range and short-range mobility, which enables the model to accurately simulate disease spread by factoring both international travel and daily commuting patterns. Lastly, the disease model utilizes Susceptible-Latent-Infectious-Recovered (SLIR) compartmental scheme, where it takes infection transmission rate (β) and density of infected population (I / N) to calculate infection rate ($\lambda = \beta \check{} I / N$).

We will be mainly using three Google Cloud Product APIs to run GLEaM simulator with Deep Bayesian Active Learning surrogate model – Cloud Run, Compute Engine, and Cloud Storage Bucket. The Cloud Run service powers the main API, offering high scalability and flexibility to handle the GLEaM model's data-intensive traffic. Since the GLEaM model is containerized in a Docker image, Cloud Run is an ideal service, allowing consistent deployment specifically designed for containers. Based on user permissions and data requests, Compute Engine runs the GLEaM model by launching a VM instance with a custom Docker image. With custom configurations, Compute Engine allows users to customize VM configurations and use custom images, which would enable us to easily deploy several different epidemic simulator models. The global availability of Compute Engine would also enable the GLEaM model to be accessed worldwide with minimal latency, making it suitable for global epidemic modeling. Additionally, seamless integration with other Google Cloud services allows Compute Engine to store output data in Cloud Storage, ensuring data is readily available for the main API, enabling on-demand access for users.

1.1.2 Model Training and Active Learning

A major paper we based our work on was Deep Bayesian Active Learning for Accelerating Stochastic Simulation. This paper contains information about speeding up stochastic simulations in active learning frameworks specifically within the domain of our topic. This is all based on an Interactive Neural Process (INP) for training deep surrogate models. This framework can be broken down into two parts: Spatio-Temporal Neural Process (STNP) and a reward acquisition function called Latent Information Gain (LIG). The STNP can be broken down into two parts: spatial and temporal aspects. The spatial aspect assumes we do not treat locations on the map as independent, we capture the relationships between locations via graphs and other methods.The temporal aspect delves into capturing relationships at different times. Multiple time variables are used to represent this relationship instead of a single one. Each time variable is sampled based on past history and allows the model to build far better connected relationships between all of the various variables. The reward acquisition function (RAF) aspect of this framework creates an interesting twist to the active learning framework. There are two main choices of RAF discussed are Expected Information Gain (EIG) and LIG as mentioned above. Due to various limiting factors, the only way EIG is feasible is through normality assumptions of the conditional distribution of predicting future states. As such, LIG is proposed to be used to gain further flexibility. Instead of calculating the value in the observational space, it is done on the latent (theoretical) space. This gives reductions in computational cost, making it a more feasible option as a scorer. As well, in Disentangled Multi-Fidelity Deep Bayesian Active Learning Wu

et al. (2023), Wu et al. discuss how multi-fidelity data can be leveraged in order to reduce the computational cost and time of training a Deep Bayesian Active Learning Model. The multi-fidelity data is obtained by running a variety of simulations each with their own level of data quality. Then, using a novel model architecture that separates the local and global hidden representations for each fidelity level, the model is then trained on this multi fidelity data. The novel model architecture prevents errors from trickling up to the highest-fidelity level and therefore maintains accuracy at all fidelity level data. Furthermore, an acquisition function and batch learning are utilized to speed up the data generation and training processes.

In our approach, we are doing a methods based topic. Therefore, we will discuss what kind of data our approach can be used for. In this explicit format, working with epidemic data is the best. This is due to the fact that it was designed to specifically work with this kind of data. But as a further framework, this can be used in many different domains. Anything that relies on stochastic simulations while utilizing temporal and spatial data would be great. Some other examples would be environmental and weather forecasting, seismic activity prediction, and urban traffic flow prediction and management. If there is any application that makes use of stochastic simulations along with prediction based on time and location data, this framework would be a very good choice.

2 Methodology

2.1 API Design

2.1.1 API Architecture

The API follows a RESTful architecture, adhering to the principles of Representational State Transfer (REST). This design ensures stateless communication, making it easier for clients to interact with the API using HTTP requests and responses. To standardize and document the API, we use the OpenAPI Specification (OAS). The OpenAPI Specification defines the API structure, including detailed information about each endpoint, the request and response formats, parameter requirements, and available operations. The OpenAPI Specification is written in YAML format, which is consistent with the format used for API requests and responses.

The Endpoints:

1. `/ (Root)`: This endpoint is responsible for updating the user's number of runs in the system. It tracks the usage of the API by individual users.
2. `/cost_estimation`: This endpoint accepts a single set of parameters. The cost of the instance is estimated and compared to the user's monthly limit. If the estimated cost is below the threshold.
3. `/gleam_simulation`: This endpoint takes in the user parameters to generate stress testing and data. The data is then dumped into a bucket for later retrieval by the user.

4. `/download_folder`: This endpoint retrieves the data from the data bucket and returns it to the user.
5. `/gleam_ml`: This endpoint runs the ML model and produces the predictions. These predictions are then returned to the user.

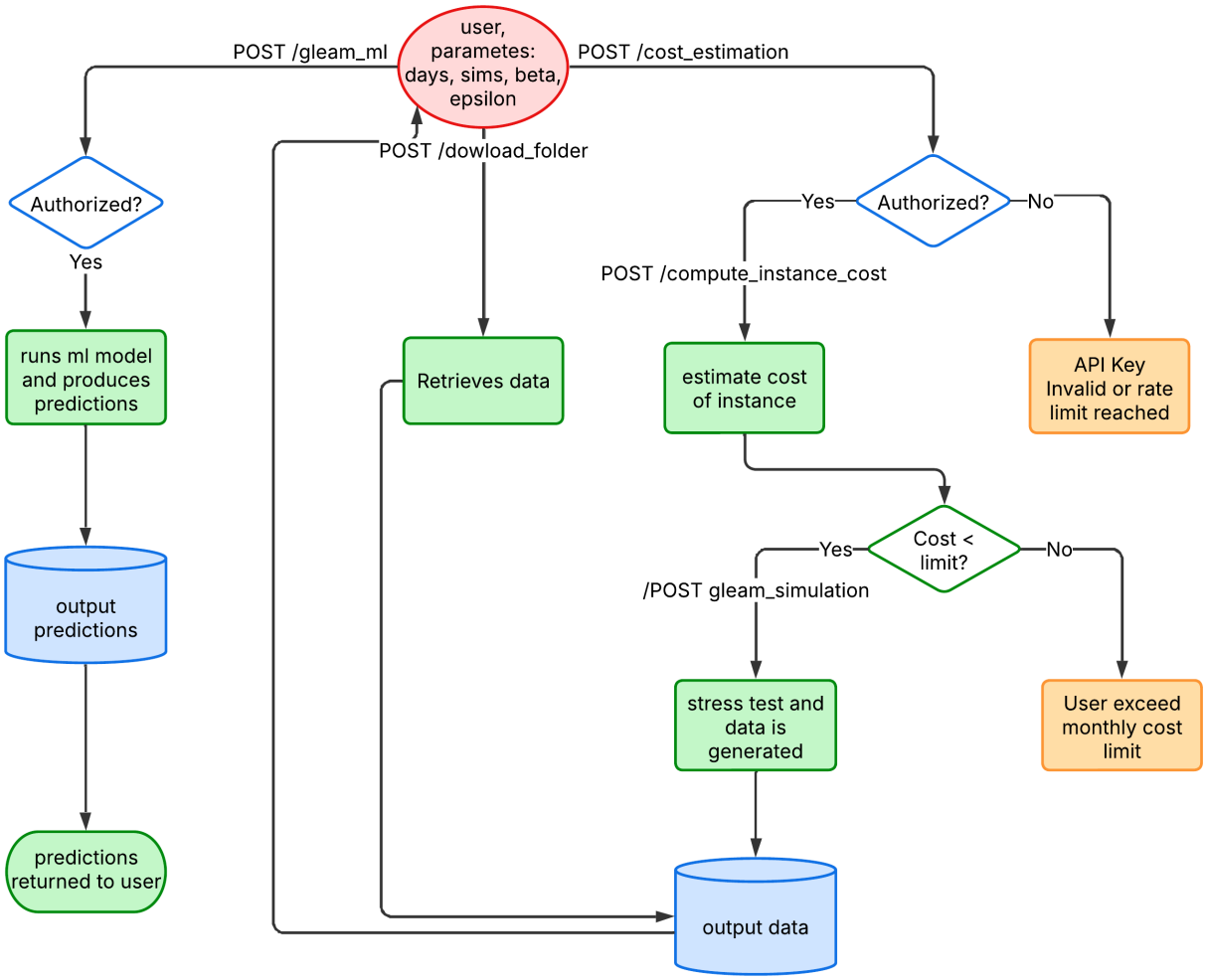


Figure 1: API architecture

2.1.2 Security and Deployment

The API key management is handled through Firestore, where valid API keys are stored along with associated user information (such as usage limits and roles). The API employs rate limiting such that users are restricted to a maximum of 1000 calls per minute to prevent abuse and ddos attacks. The API is deployed using Google Cloud Run and continuous deployment is set up from a GitHub repository. The deployment process uses a Docker image built from the GitHub repository, which guarantees consistency across environments. Stress testing was conducted to assess the robustness of our API under different traffic condition using `polinux/stress Docker Image` to simulate various load scenarios by generating

concurrent requests to the API, stressing CPU, memory, and I/O resources. The cost of the instance is based on the number of GPUs, CPUs, and RAM used over a given number of hours. It retrieves up-to-date pricing from the Google Cloud Billing API, updating default prices if a match is found. Finally, it computes the total cost by multiplying the resource quantities by their hourly rates. The cost of the instance in addition to the user's monthly cost must be below the user's monthly threshold.

2.2 Global Epidemic and Mobility Model (GLEAM)

The surrogate model we have developed requires simulation data from the Global Epidemic and Mobility Model (GLEAM) [1]. GLEAM is a spatially structured, stochastic epidemic simulation that integrates real-world population and mobility data to model the spread of infectious diseases at a global scale. The model accounts for heterogeneous mixing patterns, mobility networks, and stochastic disease transmission, making it a powerful tool for epidemic forecasting. The core dynamics of GLEAM are governed by a large number of parameters inputted to it such as virus attributes, population attributes, etc. It outputs data based on these factors, which is then used by our model to train on. Our surrogate model aims to learn the transmission dynamics of epidemics based on input parameters mentioned above. The GLEAM model serves as a high-fidelity source of epidemic simulation data, which we use to train a spatio-temporal neural process model.

2.3 Surrogate Model

To approximate pathogen spread based on the parameters efficiently, we use an STNP model that selects data to train on based on a Bayesian Active learning framework.This framework chooses data based on latent information gain score, which allows the computation of the score to be done in the lower dimensional latent space, speeding up the data selection process without loss of information. The model architecture also ensures that time is being processed on multiple variables, ensuring a better understanding of the spread of a pathogen over various time increments. We also ensure that we track the locations of the pathogen's spread via an adjacency matrix that represents each location against each other. This way, we ensure to capture the dependency relationship between locations properly and replicate the true nature of pathogen spread in our model.

2.3.1 Data

In our setup, we train a STNP model based on data formatted in a specific format. Our training data has a shape of (16636, 58, 28, 10), each representing:

- 58: California counties
- 28: Time increments (28 days a month because of February)
- 10: Input features

We also loaded graph data represented as an adjacency matrix for California counties. With this, we were able to train the model as mentioned above.

2.4 Bayesian Active Learning

In this framework, we create a more computational effective strategy for training the STNP model. Usually, massive amounts of simulation data are required for training surrogate models in order to have them generalize well using observed data.

Instead, using the GLEAM simulator alongside the Bayesian Active Learning (BAL) framework, allows us to reduce the computational cost. In the BAL framework, we choose an initial data point(s) to train. Once the training is finished, we then select a new data point to train based on a given acquisition function (AF). In our instance, we used an acquisition function called latent information gain (LIG) in order to evaluate the next data point. This measures how acquiring a label will reduce the uncertainty of the model's latent variables. The higher the score, the better the data point quality.

In our project, using EIG, which does a similar calculation but in the observational space, proved to be computationally expensive due to having no closed-form solution and only being viable under constrained scenarios. LIG is able to bypass these constraints and be extremely effective. This overall process repeats for the amount of queries that are user defined.

2.4.1 Scoring Experiment

Scoring allows us to evaluate and select data points for training the surrogate model in the Bayesian Active Learning (BAL) framework. We leveraged different settings to find the scoring that prioritizes data points that maximizes the model's learning efficiency while minimizing computational costs.

In this experiment, we evaluated two scoring methods with two different batch sizes 1 and 3:

- Old Default Scoring Method
- New Scoring with $\lambda = 0.5$

New scoring method uses a modified acquisition function incorporating a scaling parameter of $\lambda = 0.5$ for distance reward, which adjusts the weight given to uncertainty-based selection.

2.4.2 Batch Size Experiment

To explore a way to further optimize training efficiency with the BAL framework, we analyzed how different batch size affects the model performance. We used greedy batch selection strategy to ensure that each batch contains a diverse and informative set of data points, reducing redundancy and improving training efficiency.

We tested three different batch sizes:

- Batch size = 1
- Batch size = 3
- Batch size = 5

2.5 Model Predictions

Once trained, we were able to make predictions on any kind of dataset inputted to the model. The data must conform to the shape expectations as listed above. First, we serialize the trained model into an object. Then, we take the inputted data and split it into batches of size 128, a parameter in the model configurations, to process data. Each set of 128 is then inputted into the model's forward function and given an output. These outputs are then all concatenated together and returned.

Our prediction output will have a shape of (number of observations, 29, 24), each representing:

- 9: Time increments (28 days a month because of February but including input in the begnning)
- 24: Compartments

The 24 compartments each represents total number of people of :

Table 1: Compartment Labels for Prevalence

Compartment	**Label**
1	Latent Prevalence
2	Infectious Symptomatic Prevalence
3	Infectious Asymptomatic Prevalence
4	Hospitalized Prevalence
5	ICU Prevalence
6	Removed Asymptomatic Prevalence
7	Removed Symptomatic Prevalence
8	Home Asymptomatic Prevalence
9	Home Mild Prevalence
10	Home Severe Prevalence
11	Removed Hospitalized Prevalence
12	Deaths Hospitalized Prevalence

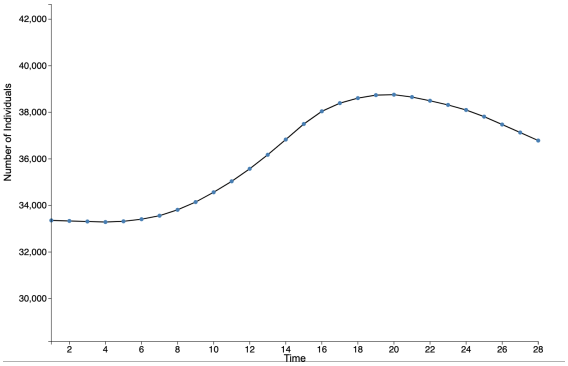
Table 2: Compartment Labels for Incidence

Compartment	Label
13	Latent Incidence
14	Infectious Symptomatic Incidence
15	Infectious Asymptomatic Incidence
16	Hospitalized Incidence
17	ICU Incidence
18	Removed Asymptomatic Incidence
19	Removed Symptomatic Incidence
20	Home Asymptomatic Incidence
21	Home Mild Incidence
22	Home Severe Incidence
23	Removed Hospitalized Incidence
24	Deaths Hospitalized Incidence

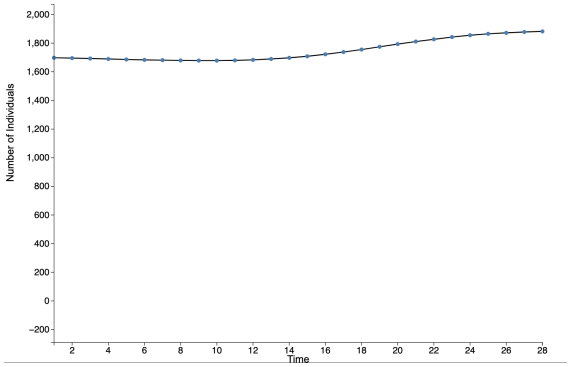
3 Results

3.1 Prediction Output

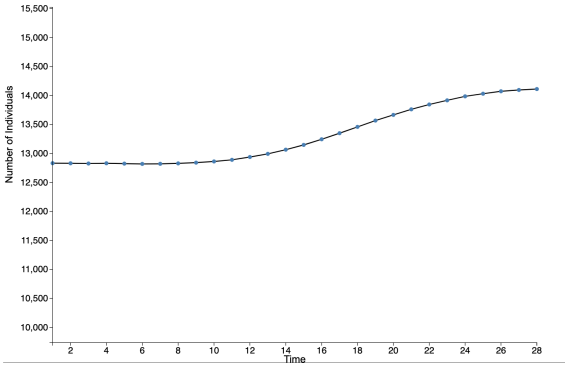
Figure 2 shows example outputs - temporal changes in the number of individuals across compartments 2 (Infectious Symptomatic Prevalence), 4(Hospitalized Prevalence), 19(Removed Symptomatic Incidence), and 23 (Removed Hospitalized Incidence).



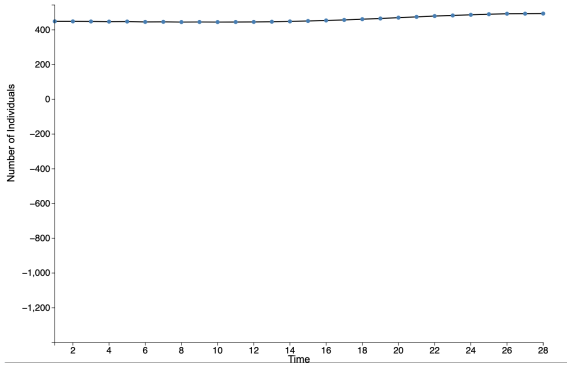
(a) Compartment 2



(b) Compartment 4



(c) Compartment 19



(d) Compartment 23

Figure 2: Prediction Outputs for different compartments

We evaluated the performance of our model using the log-scaled Mean Absolute Error (MAE) loss on the test dataset. The figure 3 presents a comparison of the log-scaled test MAE between our Leam-US model and the offline performance of a baseline model.

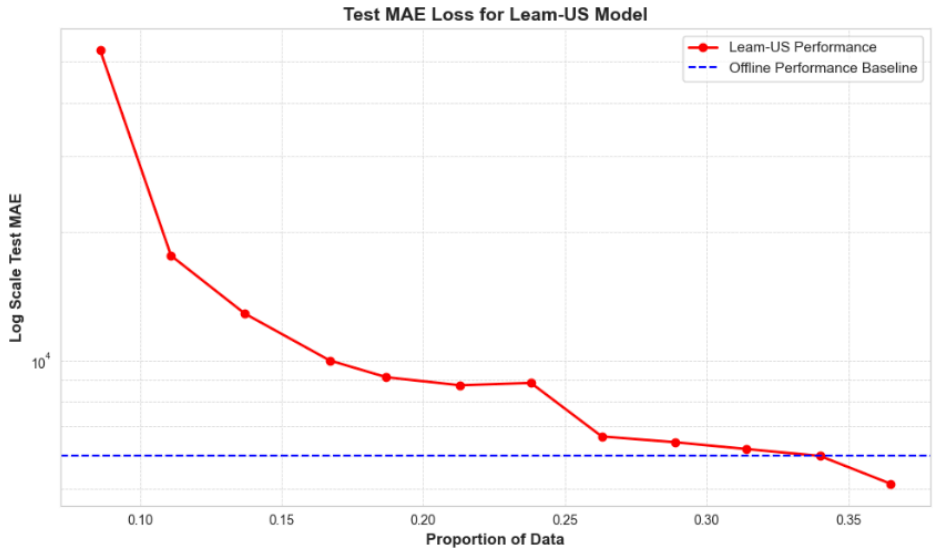


Figure 3: Test MAE of Leam-US and Offline

Leam - US roughly converges with the offline model at 0.36. This indicates that it is able to achieve the same test MAE by training on approximately 36% of the data.

3.2 Baysian Active Learning Experiments

3.2.1 Scoring Experiment

Figures 4 and 5 illustrate the results of our scoring experiment, where we evaluated the performance of the SEIR model (simple and naive simulator) using our active learning framework with batch sizes of 1 and 3, respectively, over 8 iterations. The figures display both the score images and the Mean Absolute Error (MAE) images obtained over iterations. This gives a relative understanding of how the framework works and its general behavior. This was done because the SEIR model is much easier to visualize in comparison to the LEAM-US simulator, which is much higher dimensional.

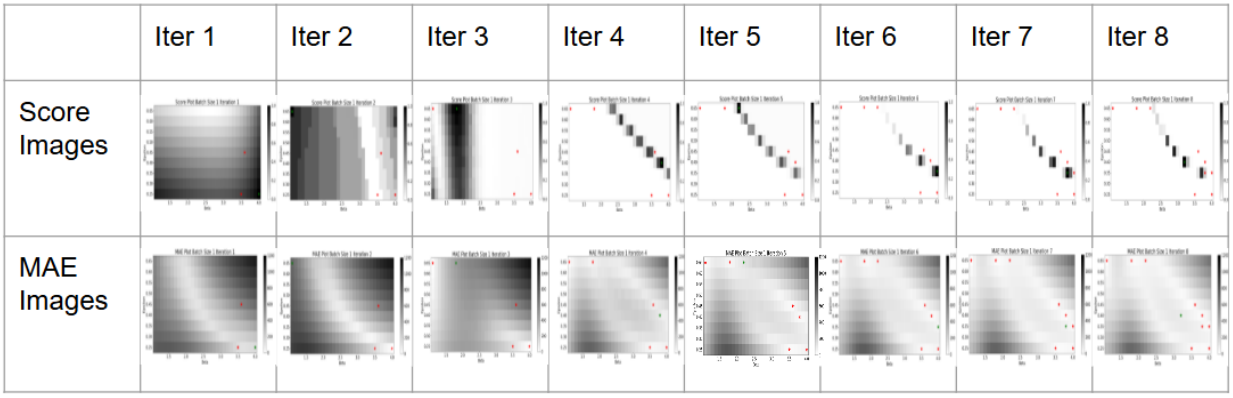


Figure 4: Score and MAE Images of Batch 1

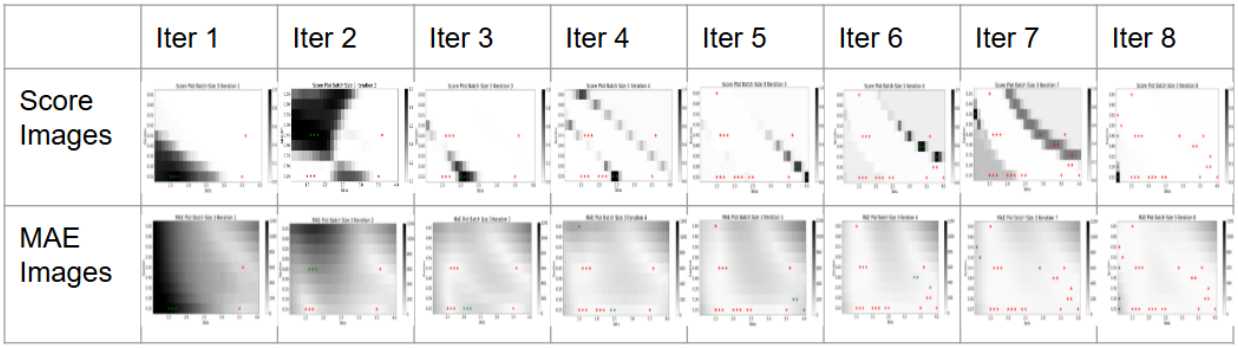


Figure 5: Score and MAE Images of Batch 3

3.2.2 Batch Size Experiment

Figure 6 shows the test MAE losses across different batch sizes of 1, 3, and 5. This shows batch active learning with a greedy approach to point selection. The general goal is to train on more data even if the data is generally of lower quality, to gain performance improvements. This algorithm scales with the batch size. It operates as such: we choose the best data point in accordance to LIG, traverse the dataset again until we find the highest LIG score for the joint dataset until the desired batch size is achieved. In this experiment, we realized there was a benefit to selecting more data points, but it has a limit to its effectiveness.

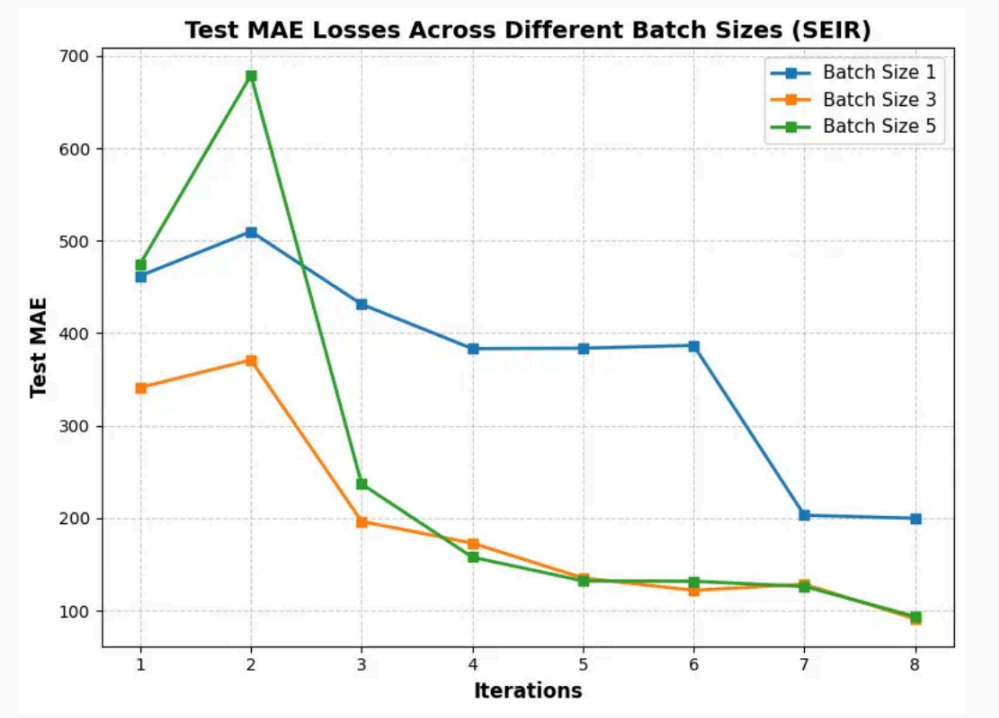


Figure 6: Test MAE Across Batch Sizes

4 Discussion

4.1 Leam-US and Bayesian Active Learning

A clear trend was shown in the performance of the Leam-US model as we increased the proportion of data used for training. The log-scaled MAE loss of the Leam-US model exhibited a convergence to the offline model baseline as training went on. The purpose of this was to demonstrate that we were able to achieve similar performance to the offline model on sixty-four percent less data through the active learning framework. Hence, we are able to make similar predictions with Leam-US with less computational cost.

4.2 Cloud Capabilities at Scale

Actionable insights from epidemic modeling requires vast amounts of data. In this vein, we have implemented a scalable workflow, where data can be dumped to a GCP storage bucket, as a store for Big Query analysis down stream. This allows advanced users to collect simulation data in a cloud environment for large-scale analysis. Depending on user spend allowance, researchers would be able to run hundreds of simulations spanning equal number of virtual machines, and pool all the data in a central, cloud hosted, location.

5 Conclusion

The developed API efficiently integrates AI surrogate models with the GLEAM simulator, enabling real-time epidemic forecasting and adaptive disease modeling. By leveraging Bayesian Active Learning, the model significantly reduces computational costs while maintaining high accuracy. Stress testing confirmed the system's scalability and robustness, making it suitable for deployment in high-demand, real-time scenarios.

6 Appendix

6.1 Project Proposal Introduction

Epidemic modeling is crucial for disease forecasting and prevention. Epidemic simulators are used to characterize risks, forecast impact on healthcare, and supplement public-health policy making. However, current epidemic simulations are computationally expensive and time consuming when running simulations of high granularity. This hinders public health policy making, especially during war-time emergencies, such as COVID-19.

Recent advancements AI surrogate models have shown promise in accelerating traditional epidemic simulations. AI surrogate models mimic the behavior of traditional numerical simulations at a much higher inference, which reduces run time from weeks to days. However, existing AI models are trained on pre-simulated data, which is costly to store and static, making them incompatible with the dynamic nature of disease spread.

Our solution is to create an API that integrates AI surrogate models with the epidemic simulator. This API will allow seamless real-time interaction and data exchange. By bridging AI techniques with established simulation models, we hope to decrease run-time, reduce computational overhead, and enhance public health policy-making.

6.2 Project Proposal and Output

In Quarter 2, our project builds on these findings by addressing the deficiencies of our Quarter 1 project. We will introduce two novel aspects: we will replace the SEIR placeholder Docker image with the Global Epidemic and Mobility (GLEAM) model and substitute the neural process model with the Temporal Spatial network model. We will be granted access to the GLEAM simulator by Northeastern University. The primary output will be a fully functioning API that integrates both the GLEAM simulator and the Temporal Spatial network model, enabling dynamic and efficient epidemic simulation.

The primary output of this project will be a fully functional, secure, and scalable API that integrates the GLEAM epidemic simulator with a Temporal Spatial network model. The API will:

1. Enable real-time epidemic simulations with significantly reduced computational overhead.
2. Provide researchers and policymakers with an accessible tool for exploring dynamic disease spread scenarios.

The project will culminate in the release of the API along with a detailed technical report and documentation, ensuring its usability, reproducibility, and long-term impact.

7 Contributions

- Ethan Cao: Helped create the database, ML endpoint, stress testing, and cost estimation.
- Alaa Fadhlallah: helped with cost estimation and stress testing, data visualization, website
- Manav Jairam: Helped with model training, cloud computing, generating predictions
- Liam Manatt: Designed the simulator creation logic, implemented end points for VM creation and data retrieval
- Kyla Park: Helped working on test image and experimenting docker/custom image, data visualization, website.
- Anirudh Indraganti: Helped with replicating papers, model training, generating predictions, and implementing the API endpoint that exposes a trained version of the LEAM-US simulator for prediction.

References

Balcan, D., B. Gonçalves, H. Hu, J. J. Ramasco, V. Colizza, and A. Vespignani. 2010. "Modeling the spatial spread of infectious diseases: The global epidemic and mobility computational model." *Journal of Computational Science* 1 (3): 132–145. [Link]

Petrillo, F., P. Merle, N. Moha, and Y.-G. Guéhéneuc. 2016. "ARE REST APIs for cloud computing well-designed? An exploratory study." In *Lecture Notes in Computer Science*.: 157–170. [Link]

Wu, D., R. Niu, M. Chinazzi, Y. Ma, and R. Yu. 2023. "Disentangled multi-fidelity deep Bayesian active learning." In *International Conference on Machine Learning*. PMLR