REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
31-01-2022	Final Report	1-Sep-2020 - 30-Sep-2021

4. TITLE AND SUBTITLE

Final Report: SuperMaaS: A Framework for Modeling as a Service to Support Data- and Model-Driven Analysis and Response to Emerging Crises and Real-Time Events

5a. CONTRACT NUMBER

5b. GRANT NUMBER

W911NF-20-C-0056

5c. PROGRAM ELEMENT NUMBER

6. AUTHORS

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES

Galois, Inc.
421 SW Sixth
Suite 300
Portland, OR 97204 -1622

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)

ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

77289-CS-DRP.1

12. DISTRIBUTION AVAILIBILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Eric Davis
UU	UU	UU	UU		19b. TELEPHONE NUMBER: 503-626-6616

RPPR Final Report
as of 28-Jul-2022

Agency Code: 21XD

Proposal Number: 77289CSDRP **Agreement Number: W911NF-20-C-0056**
INVESTIGATOR(S):

Name: Eric Davis
Email: ewd@galois.com
Phone Number: 5036266616
Principal: Y

Organization: **Galois, Inc.**
Address: 421 SW Sixth, Portland, OR 972041622
Country: USA

DUNS Number: 098009918 EIN:
Report Date: 31-Dec-2021 Date Received: 31-Jan-2022
Final Report for Period Beginning 01-Sep-2020 and Ending 30-Sep-2021
Title: SuperMaaS: A Framework for Modeling as a Service to Support Data- and Model-Driven Analysis and Response to Emerging Crises and Real-Time Events
Begin Performance Period: 01-Sep-2020 **End Performance Period:** 31-Dec-2021
Report Term: 0-Other

Submitted By: Ted Hille Email: ted@galois.com
Phone: (503) 891-6127

Distribution Statement: 1-Approved for public release; distribution is unlimited.

STEM Degrees: **STEM Participants:**

Major Goals: Overarching Goal of the Project:

The goal of the SuperMaaS project is to improve the ability of policy and decision makers to respond to critical events and crises. Prior to this research, the typical workflow relied on large teams of experts, often in separate silos, making decisions in opaque manners that were rarely informed by data or direct modeling of the crises occurring and interventions planned. This method of operation introduces severe limitations, especially when the processes involved require feedback or iterative solutions as new information emerges. The resulting process lacks the agility necessary for proactive responses to real time events and results in knowledge. Further complicating this situation is that such knowledge is typically communicated through the exchange of static PDFs from a variety of domain experts.

Milestones:
- Extension and Development of SuperMaaS Model Registry System [45% complete]
- Extension and Development of SuperMaaS Data and Indicator Registry System [60% complete]
- Extension and Development of SuperMaaS Transformation Registry System [50% complete]
- Extension and Development of SuperMaaS Expert Modeler System [85% complete]
- Extension and Integration of SuperMaaS HMI with CausMos for Visualization in a Consistent Visual Language [60% complete]
- Extension and Integration of Federated Search APIs for SuperMaaS, DataMart, and CausMos [50% complete]
- Ongoing Testing, Evaluation, and Integration of SuperMaaS System

Accomplishments: Over the course of the SuperMaaS program, we were able to accomplish the following:

Development of a system for registering models within the SuperMaaS system. The ultimate goal of this work was to create a low-friction system for expert modelers to submit their models to SuperMaaS. To this end, we were able to create a bare-bones system that captured all the relevant meta-information for a given model along with the model resources. This allowed us to utilize the provided type information to reason about the registered model in relation to other models in SuperMaaS such that we were able to support connecting different models together to enable modeling of more complex scenarios than would be possible with a single model alone. However, despite having constructed an API for HMI integration that fit with our subcontractors at Uncharted's stated requirements, and having frequent meetings to discuss ways to help with integration, our subcontractors never utilized this API, meaning users were required to use lower-level command-line tools to register their models

RPPR Final Report
as of 28-Jul-2022

Development of a system for registering datasets and indicators within SuperMaaS. In addition to registering models, we worked to allow for the registration of datasets required by models as well as indicator data that could be used in conjunction with models. Similar to model registration, we were able to implement the necessary functionality with the core SuperMaaS registry system to support type annotations of data to allow for automated reasoning about the data when combining it with other models / data. Similar to the model registry, we did not get to a point of actual demonstration of the integration with a HMI due to aforementioned issues in working on a final integration with a subcontractor.

Development of a system for registering transforms to allow for the conversion of inputs / outputs between models such that simple incompatibilities between models could be automatically addressed within the SuperMaaS system. In this effort, we were able to develop a robust system for transforms and created a number of general-purpose transforms as examples of how transforms could be written / registered. As with the model and data registration efforts, this work was not ultimately integrated into an HMI for similar reasons.

Development of an expert-modeler system for defining complex graphs of data / model relations in support of allowing expert modelers to answer more complex questions than those addressed via running a single model. In this effort, we were able to create a job pipeline system that supported the composition of data, models, and transforms to allow for automated reasoning about the defined composition, surfacing incompatibilities such that users could address issues in advance of actually running the pipeline. This effort was able to get to the point of defining pipelines via a REST API, with jobs being able to be scheduled/executed within the SuperMaaS system. As in other areas of this program, we were not able to demonstrate a complete integration with a HMI due to issues in the subcontractor closing the final mile by implementing calls into the SuperMaaS system.

Integration with the CauseMos HMI to allow for exploration of model outputs in analyst workflows. We were able to demonstrate the basic integration of datasets generated within SuperMaaS being consumed by the HMI. However, we did not get to the point of supporting round-trip data-management where an analyst could ask questions of domain modelers through the HMI, to the SuperMaaS backend and ultimately back to the modeler. This was primarily a result of not being able to complete the full work on this integration due to the decision to stop funding halfway through the program.

Finally, it should be pointed out that the final increment of program funding was cut which meant that we were unable to take the basic functionality we were able to create in the first half of the program and flesh out features to meet our ultimate goals of usability and flexibility.

Training Opportunities: Nothing to Report

Results Dissemination: Disseminated publicly as open source, distributed during demos and experiments, and made available to domain scientists and practitioners on the program.

Honors and Awards: Nothing to Report

Protocol Activity Status:

Technology Transfer: Nothing to Report

PARTICIPANTS:

Participant Type: PD/PI
Participant: Eric Davis
Person Months Worked: 15.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Other Professional

Participant: Ted Hille
Person Months Worked: 15.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Other Professional
Participant: Sourya Dey
Person Months Worked: 8.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Other Professional
Participant: Sam Cowger
Person Months Worked: 8.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Other Professional
Participant: Alexander Grushin
Person Months Worked: 10.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Other Professional
Participant: Ryan Scott
Person Months Worked: 14.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Other Professional
Participant: P.C. Shyamshankar
Person Months Worked: 8.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Other Professional
Participant: Annie Cherkaev
Person Months Worked: 3.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Other Professional
Participant: Jared Weakly
Person Months Worked: 6.00 **Funding Support:**

Project Contribution:
National Academy Member: N

Participant Type: Other Professional
Participant: Yerim Lee
Person Months Worked: 1.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Other Professional
Participant: Alexander Baskt
Person Months Worked: 6.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Partners

,

I certify that the information in the report is complete and accurate:
Signature: Ted Hille
Signature Date: 1/31/22 4:20PM

galois

SuperMaaS Final Report

Galois: *Alexander Bakst, Sam Cowger, Iavor Diatchki, Ajay Kumar Eeralla, Alexander Grushin, James LaMar, David Lamkins, Yerim Lee, Hari Menon, Ryan Scott, Panchapakesan Shyamshankar, Ted Hille, Eric Davis*[†]
Jataware *Brandon Rose, Justin Garilow*
University of Arizona *Clayton Morrison*
Uncharted *Pascal Proux, Adam Carolli, Nelson Liu*
[†]ewd@galois.com
Jan 30, 2022

1. Introduction

The goal of the SuperMaaS project is to improve the ability of policy and decision makers to respond to critical events and crises. Prior to this research, the typical workflow relied on large teams of experts, often in separate silos, making decisions in opaque manners that were rarely informed by data or direct modeling of the crises occuring and interventions planned. This method of operation introduces severe limitations, especially when the processes involved require feedback or iterative solutions as new information emerges. The resulting process lacks the agility necessary for proactive responses to real time events and results in knowledge. Further complicating this situation is that such knowledge is typically communicated through the exchange of static PDFs from a variety of domain experts.

Knowledge generated in this fashion lacks necessary context and is independent from artifacts generated from both other experts working on the same problem and other forms of analysis, leading to issues of model credibility, and a lack of trust. Furthermore, the fundamentally manual nature of this process and the lack of proofs of correctness and proper implementation can lead to errors, even when the core theory is sound. SuperMaaS aims to provide a framework which advances the state-of-the-art for Modeling as a Service (MaaS), and allows for human-machine teaming during a crisis, improving the capability of experts in various roles to respond as a more cohesive, evidence-driven, team. SuperMaaS has focused on the human-component of this symbiotic process by characterizing human workflows into three categories: (1) Domain modelers, the traditional experts in their fields providing guidance, software, and results to decision makers; (2) Expert modelers, software and simulation experts in the employ of the government who can rapidly acquire basic skills and knowledge about a model; and (3) Analysts, who represent knowledgeable members of the decision making process who, while they cannot operate a simulator, can interpret its results if presented in human explainable fashions.

This research has resulted in several key capabilities:

- The ability to register models with abstracted inputs and outputs, their calibration and preparation procedures, and complex execution plans.
- The production of point-lattice emulations of model outputs in the form of data cubes.
- The exposure and integration of SuperMaaS APIs into a larger ecosystem of analyst-focused

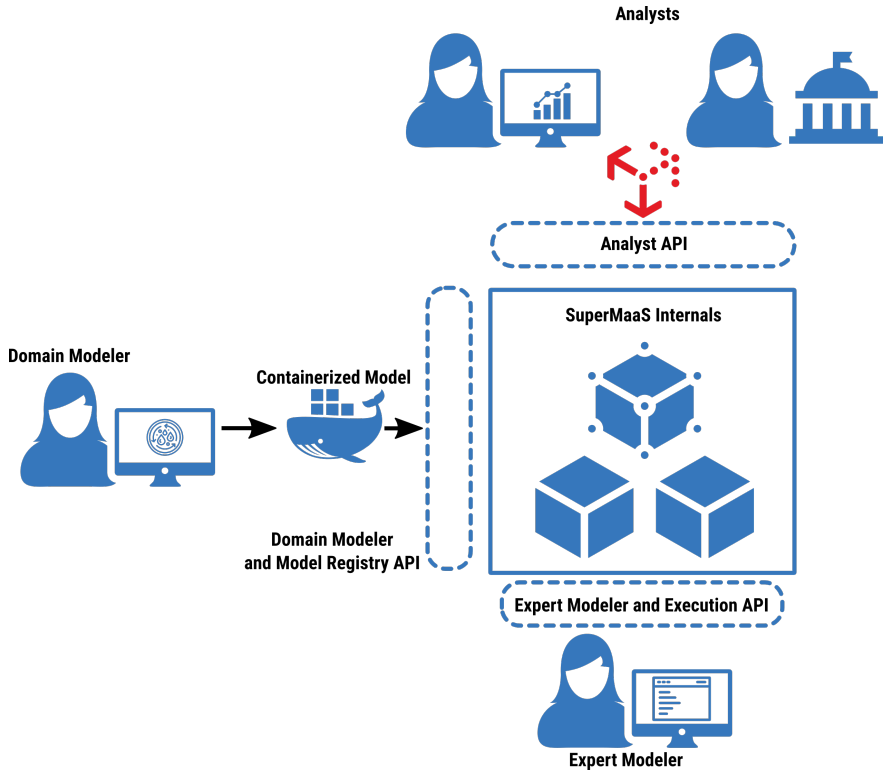


Figure 1: An architecture overview of the SuperMaaS framework.

dashboards, tools, data, and models.

- The generation of new methods for crisis response that improve agility, understandability, usability, and generalizability for models, data, indicators, and model results.

In this report, we discuss the major achievements of the SuperMaaS program, avenues for future work, and the results of experimentation, demonstration, and evaluation.

1.1. SuperMaaS Architecture

Figure 1 shows the notional components of the SuperMaaS architecture as they impact the various user roles. The SuperMaaS framework has been designed to interact with three classes of users in a symbiotic fashion, defining classes of users by their capabilities and general knowledge. We call these users **Domain Modelers**, **Expert Modelers**, and **Analysts**.

We identified three key user-roles involved in the process of creating, customizing and consuming models within the context of the given problem domain. Those roles are:

Domain Modelers. A domain modeler is a subject matter, or domain, expert who has developed an executable model capable of producing predictive results given a set of well structured inputs. For our use case, we assume that the domain modeler is not a software or systems engineer and that their expertise in technical systems might very well be limited to the model they created. The goal of the system is to serve as part of a Human-Machine Team with the domain modeler with the goal of helping the domain modeler to abstract and containerize [**merkel2014docker**]

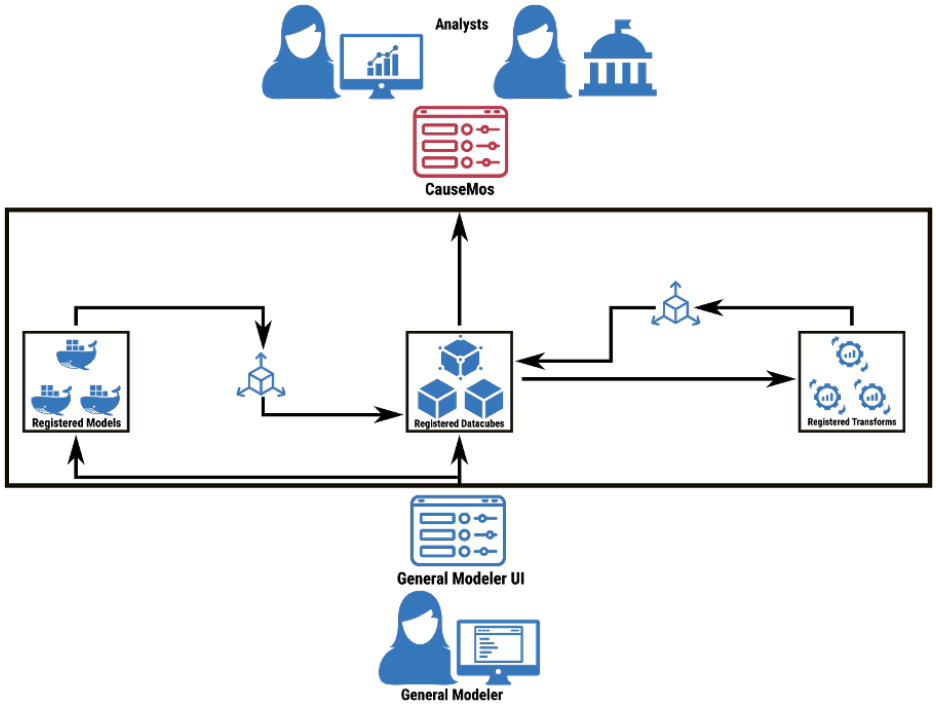


Figure 2: High Level Overview of SuperMaaS Workflows.

their model for use by other user classes.

General or Expert Modelers. A general modeler, or an expert modeler, is someone with general experience executing models, working with software systems, who serves in a support role to an **Analyst**. General modelers may not have any familiarity with a given registered model, but are capable of executing abstracted and registered models as part of the Human-Machine Team within the system to generate new data cubes for an analyst.

Analysts. An analyst is an individual involved directly in policy making, advising, analysis, or other support roles for crisis response. While they are assumed to be experts in their domain, they often lack software or systems expertise and cannot reliably run bespoke models produced by domain modelers without lengthy and specialized training. They participate in SuperMaaS workflows by accessing data cubes produced by general modelers from models registered by domain modelers. The overarching goal of SuperMaaS is to enable the work done by analysts, removing the cognitive load and training required to operate bespoke models produced by domain modelers, by mediating the interface between these user classes.

1.2. Workflows

Figure 2 shows a high-level overview of these workflows. We make the assumption that General Modelers working with SuperMaaS are partnered with one or more Analysts, and tasked with completing modeling tasks related to a few major goals:

- Generating new datacubes as the result of an experiment with a model.

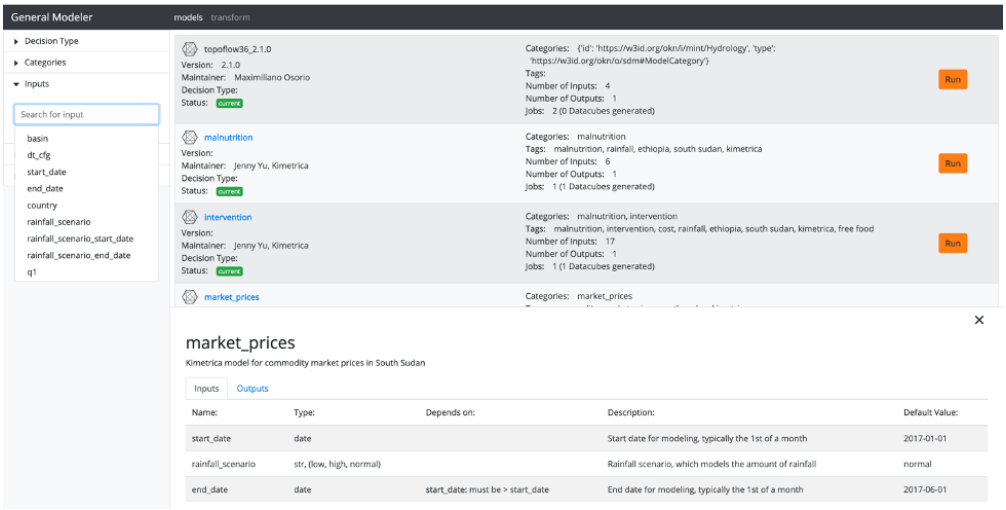


Figure 3: SuperMaaS Model Registry Example

- Transforming datacubes into new datacubes using our datacube transform system.
- Identifying datacubes that either already exist in SuperMaaS, or resulted from a General Modeler experiment or transform.

1.2.1. Generating New Datacubes as the Result of an Experiment

The principal role of the General Modeler is ensuring that Analysts they are working with have access to datacubes which allow them to visualize, understand, and develop predictions, understanding, and policy outcomes. While SuperMaaS works to abstract models provided by Domain Modelers into more usable, proceduralized, variants with fewer, more understandable inputs, and formats their outputs into the generalized schema of a datacube, model operation is still a task that requires generalized technical expertise.

Initial versions of SuperMaaS focused on pre-generated datacubes to aid in rapid testing of both SuperMaaS and CauseMos, such pre-generated datacubes only cover a small amount of the possible model spaces of interest, and will not exist for new models registered with SuperMaaS. The primary way that General Modelers assist Analysts, is therefore in the generation of new data cubes. Figure 3 shows the SuperMaaS model registry interface, which displays all models in the registry to General Modelers; allows them to search and facet the registry based on their needs; contains information on the model's purpose, inputs, and maintainer; and finally a link to the model execution page.

Once a model has been identified, the General Modeler navigates to the Model Execution and Experimental Design page associated with a given model, like the one for Topoflow shown in Figure 4. This page is populated using metadata provided during model registry, and attempts to build a usable, understandable, interface that adequately documents the use cases, limitations, requirements, and default values for the model. SuperMaaS exposes model inputs using our initial draft of a design of Experiments interface, generating, displaying, and helping the General Modeler understand the experimental runs that will be generated on the basis of the selected input ranges,

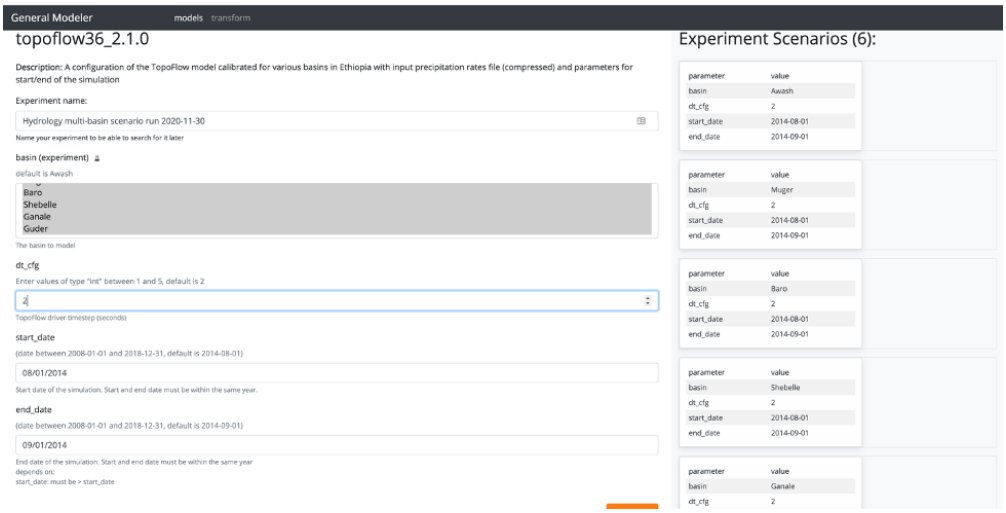


Figure 4: SuperMaaS Model Execution and Experimental Design Page

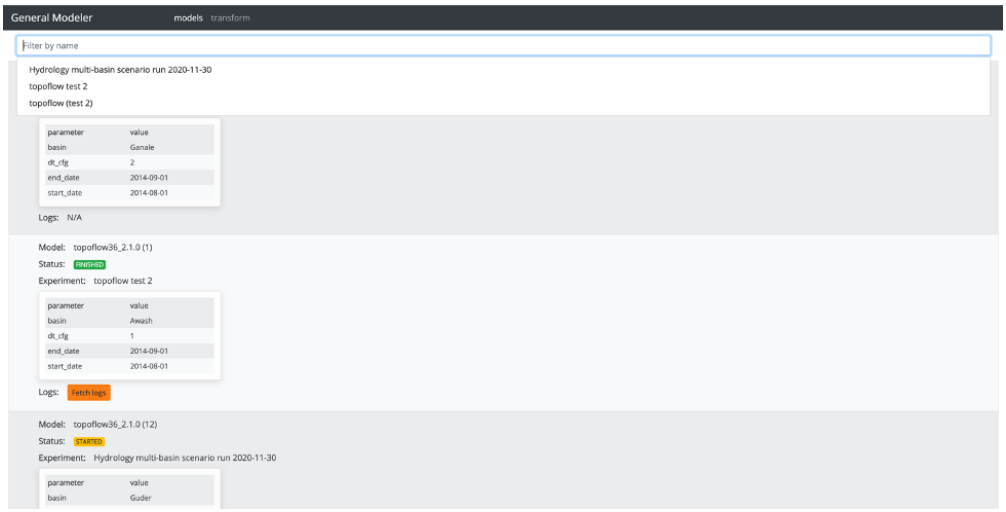


Figure 5: SuperMaaS Job Interface

and finally resulting in a set of jobs for the SuperMaaS execution engine to run.

In Figure 5, we show the results of several running experiments in SuperMaaS. General Modelers have access to this interface in order to query running jobs, fetch logs for successful or failed jobs, and access the datacubes that result from a job, or set of jobs.

2. Models Supported

During the course of the SuperMaaS project, we have registered, tested, verified, and deployed a range of supported models, including:

- **Accessibility Model**
- **Topoflow**
- **Flee South Sudan, 2013 to 2015**

- **MaxHop**
- **Google Trends**
- **Cycles Results Model**
- **Conflict Model**
- **DSSAT For Kenya Maize**
- **LPJmL crop yield forecasts**
- **Malnutrition Model**
- **Population Model**
- **Water and Sanitation Model WASH**
- **Population Model**
- **Cycles Results Model**
- **TWIST**
- **Baseline Conflict Model**
- **Kimetrica Crop model**
- **Cycles Results Kenya**

Registration of these models led to the development of reusable workflows around model registration discussed in section 3 involving SuperMaaS pack files to ensure reproducable model registration, and ensure registered models continue to be compatible with new model registration concepts and methods.

3. Model Packaging/Versioning

SuperMaaS' approach to model packaging aims to satisfy a number of high-level objectives:

Backwards compatibility: models, once registered should continue to be usable with future versions of SuperMaaS to the extent possible.

Asynchronicity: models should be registerable independently of any running instance of SuperMaaS, and should be storable until such an instance becomes available.

Reproducibility: the same models registered multiple times should produce identical registered artifacts.

3.1. SuperMaaS Pack Files

A SuperMaaS Pack consists of three files, with the same base name:

1. A pack definition (.spdef): A JSON file which describes the pack. This file should be small.
2. A pack archive (.sparc): An archive file (probably .tar.gz) that contains the contents of the pack. This file is expected to be large.
3. A pack signature (.spsig): A signature that verifies the authenticity that the archive corresponds to the definition.

A complete pack consists of three files uploaded to a recognized storage repository, with the same base name and extensions as above. For example, a complete registered version of DSSAT could be;

```
s3://supermaas-model-stdlib/dssat.spdef
s3://supermaas-model-stdlib/dssat.sparc
s3://supermaas-model-stdlib/dssat.spsig
```

In this case the URI of this particular DSSAT model is `s3://supermaas-model-stdlib/dssat`; this is the URI that should be specified as the node URI in pipelines that would like to use this model.

3.1.1. Pack Definition

The pack definition is a JSON file that describes the pack. When a user-submitted pipeline references a pack, the pack.spdef file is the first to be fetched and validated, and the contents of this file determine how the rest of the pack is interpreted.

At minimum, the spdef file should contain the following fields:

- **version:** The version of the pack definition. This is necessary to support backward- compatibility with SuperMaaS, the idea being that if a model is registered and provides a valid pack definition for the current version of SuperMaaS, future versions should be able to ingest that pack even if new features have been added. This is the same mechanism employed by docker-compose files (https://docs.docker.com/compose/compose- file/compose-versioning/).
- **kind:** The kind of registerable entity represented by the pack. Examples include model, transform, cube, closure, etc.
- **hash:** The content hash of the pack archive. This is necessary to ensure that the archive file (below) is the one corresponding to this definition, as well as allowing SuperMaaS to avoid duplicate fetches of large archive files.

The remaining fields of the pack definition are determined by the values for the above two; for example,

- **kind:** model will require the type of the model, tags, etc..
- **kind:** cube will require the type of the cube, tags, etc.

3.1.2. Pack Archive

The pack archive is a tar.gz (most likely) file that contains the actual contents of the pack. The layout of the archive is determined by the (version, kind) tuple from the pack definition. For example,

kind: model will require the docker image (from docker save), and optionally ground- truth cubes of the appropriate type for verification. kind: cube will require the files of the cube itself.

3.1.3. Pack Signature

The pack signature is a digital signature that verifies the authenticity of the pack definition. This will allow SuperMaaS to reject importing packs that have not been signed by a trusted party,

which provides some security against the already hard problem of arbitrary code execution.

3.1.4. Pack Workflow

The workflow for working with packs have two parts: the pack creation workflow, and the pack consumption workflow.

With the above context, we can now define what it means for a model to be ”registered” – A model is registered if it has a pack definition uploaded to a place where a running deployment of SuperMaaS can access it. The definition may have alongside it a pack archive (not the case for dummy/stub/phantom models), as well as an optional signature.

SuperMaaS will not care how the pack was created – it could be created manually by a developer, by a registration UI, a SuperMaaS client, etc. As long as the pack files are valid, it can be ingested into a SuperMaaS deployment.

There are two main ways that packs can be consumed by SuperMaaS:

- **Explicit ingest:** someone (usually a SuperMaaS admin) asks the system to directly ingest a pack, and make it available for pipeline construction. The main use-case for this is preseeding a SuperMaaS deployment.
- **Implicit ingest:** someone (usually a general modeler) submits a pipeline execution request, which references a registered model through its URI. During pipeline preprocessing, this URI is used to fetch the pack definition file from the remote storage, and validate the rest of the pipeline (since its the definition that contains the type information, etc.). After this point, the entity registered at that URI is available in SuperMaaS for search, and construction of subsequent pipelines.

3.1.5. Implementation in SuperMaaS

When a pack is ingested into SuperMaaS, it is given a canonicalized URI: `local://<random-ID>`, and a mapping is stored from its original URI to this ID. This local URI can be used in pipelines analogously to the original URI; if the original URI is used, then pack definition is fetched and if the archive hash has not changed, the local URI is up-to-date, and is substituted. If not, the pack is re-ingested.

If a domain modeler wants to make their model available to general modelers in a specific SuperMaaS instance but do not want to make it generally available, they can push their pack directly to the deployment, whereupon it will still be assigned a canonicalized URI and can be used as such.

If the same model is uploaded to multiple different repositories (with the same hash), SuperMaaS will store a mapping from this new URI to the canonical URI of the existing model, and avoid refetching the archive unless the contents have changed.

The explicit ingest workflow referenced above can be done in a shallow manner – fetching only the definition, and making a notation in the database that the archive itself has not been fetched.

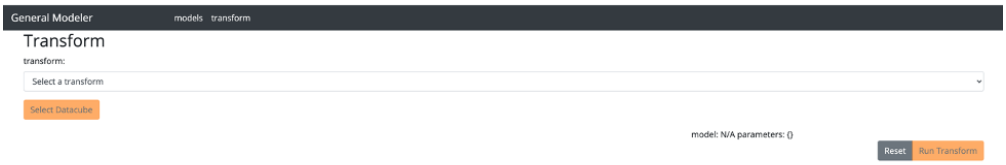


Figure 6: SuperMaaS Pipeline Scheduling Tool

This allows a deployment of SuperMaaS to contain the metadata for a large number of models/transforms/cubes/etc to show a rich searchable working set, without needing to actually fetch hundreds of gigabytes of data. If the models/transforms/cubes are actually used in a pipeline context, they can be lazily fetched, and the database can be updated to reflect that. This also allows us to support the use-case where we want to register a cube whose model is not executable (but contains metadata).

4. Pipeline Execution Engine

The SuperMaaS job execution engine allows for the execution of "plans" or "pipelines" consisting of directed acyclic graphs rooted in either parameters or data cubes, and terminating in export nodes that finalize postprocessing on model outputs for consumption by analysts, or other models. Users interact with this new system by designing plans for their experiments like that shown in the next figure:

Here, the main computational steps happen in the Model and Postprocessing transform nodes, which are part of the same experimental setup, denoted with the dashed outline. The experiment is parameterized over a configuration with n different scenarios (denoted with the gray vertices). The idea is that the nodes within the dashed outline will be run n different times, each with a different configuration. These configurations are used to construct the final datacube at the end.

An experiment is a SuperMaaS abstraction that translates into copies of a subsection of the plan, as shown here:

Note that the experimental configuration has been copied across the n different scenarios. Each scenario has a copy of `paramA` and `paramB`, but they all are instantiated with distinct values corresponding to the particular configuration they originated from. After each scenario has completed, the datacube that it produces is fed into an Aggregation transform, which (in this case) concatenates all of the datacubes into a single datacube.

A further layer of translation reveals how each sub-plan turns into individual Docker invocations, as shown here:

Each dashed box with a Docker icon in the top-left corner indicates an invocation of a single Docker container. Since the Model, Postprocessing transform, and Aggregation transform each have their own Docker image, they each require individual Docker calls. Moreover, the Docker invocations for the Model and Postprocessing transforms each have to occur n times, correspond-

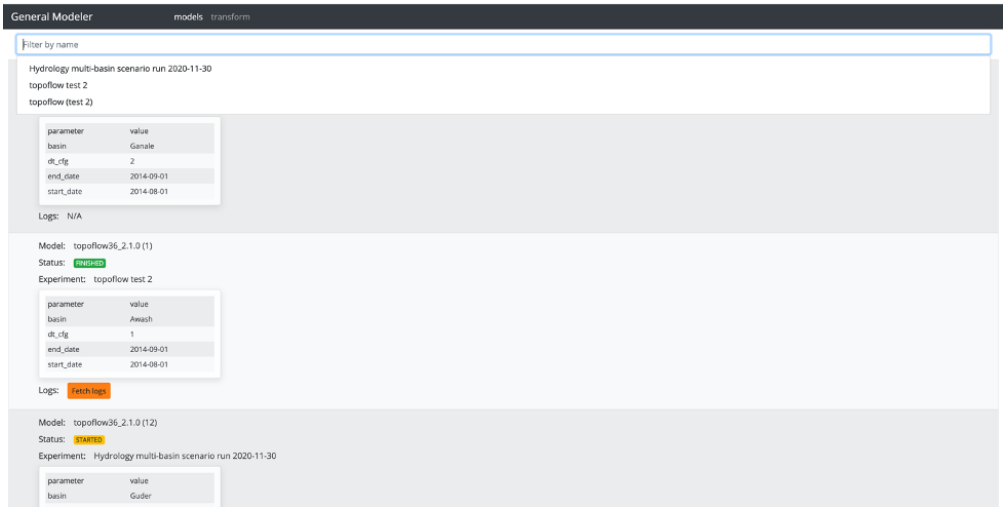


Figure 7: Example of an experiment in the SuperMaaS Pipeline

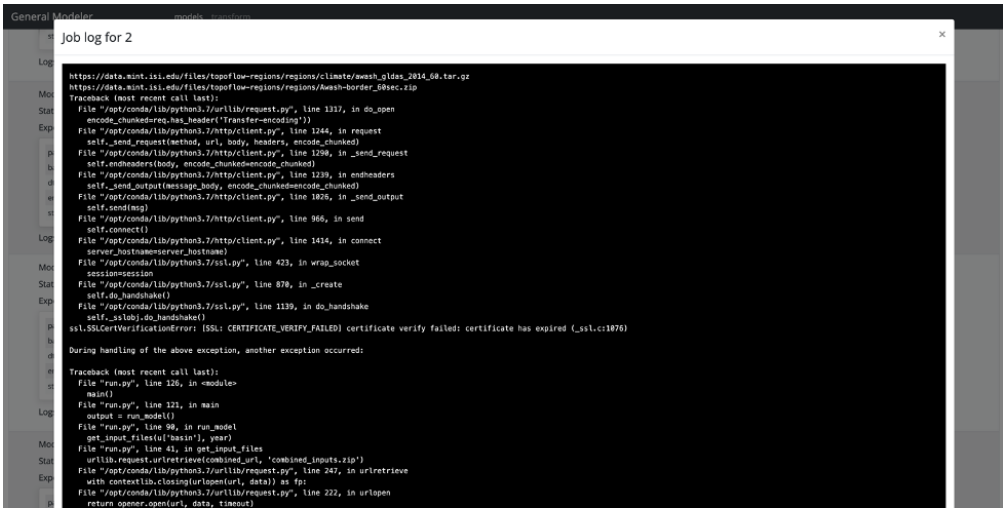


Figure 8: Docker outputs resulting from pipeline execution for debugging

ing to the n different configurations in the experiment. In total, there are $2n + 1$ total Docker invocations.

Note that this suggests a strategy for parallelization. Each of the n Docker invocations for the Model node above the first black horizontal line can be run in parallel, as there are no data dependencies. Similarly, each of the n Docker invocations for the Postprocessing transform node can be run in parallel.

SuperMaaS as designed and implemented satisfies these capabilities requirements, and can run all of the individual Docker calls previously noted. We have exposed APIs into these capabilities to allow implementation of a UI driving these capabilities. While CauseMos is the UI we have designed for, this API remains generic enough for other UIs to be utilized as well.

4.1. Type System

An execution plan composes several nodes, each with different numbers of parameters and outputs of varying types. In order to ensure that the overall composition of nodes is well formed, we have developed a type system for execution plans. The type system is designed to be a guardrail for modelers to help prevent certain classes of subtle mistakes that could lead to confusing behavior when a plan is executed. If there is a type mismatch, SuperMaaS can use the type information to produce informative error messages, which can drastically shorten feedback loops.

Each node in a plan is a function type with some number of parameters and outputs, each with their own individual types. When there is an edge between two nodes in a plan, this represents the output of one node becoming the parameter to another node. The type system enforces that the type of the output must match the type of parameter.

Parameter and output types range from scalar types (integers, floating-point numbers, strings, etc.) to datacube types. Nodes in an execution plan can be classified by the types of their parameters and outputs. Models, for instance, typically take only scalar parameters and produce datacube outputs, while transforms usually take at least one datacube as a parameter. Export nodes do not have any output types at all—instead, they are only executed for their side effects.

It is worth emphasizing that the type system is only one tool that modelers have for checking the validity of their nodes. The type system catches errors at plan-registration time—that is, before the plan is ever executed. As a result, there are certain classes of mistakes that the type system currently cannot detect. For instance, if a model requires that none of the values in a datacube parameter should be equal to NaN, then that can only be checked when running the node itself. There is a larger question of whether the type system should be extended to be able to enforce more sophisticated properties, but in general, there will always be limitations to what it is able to check.

4.2. Pipeline and Workflow Extensions for SuperMaaS

The SuperMaaS task executor and multi-node pipline infrastructure supports asynchronous execution, allowing the pipeline system to execute workloads in parallel on the basis of inter-job dependencies in the workflow. The pipeline and workflow engines also allow for pipelines to serve

as the primary entry point for BYOD (bring your own data) capabilities from DataMart, or if the General Modeler has a custom data set.

Pipelines in SuperMaaS have been implemented as directed acyclic graphs (DAGs) of nodes where each node performs some computation using either a model, or a transformer. Edges in the DAG represent depedencies, so nodes in the pipeline must be executed in order, according to dependence satisfaction. Edges indicate when data needs to be forwarded to a given node from other nodes in the DAG, as input. SuperMaaS' scheduler works asynchronously over the DAG, invoking execution of a model or transform when all of its dependencies have been satisfied by new model executions, or the corresponding datacube already exists within the system.

Additionally, the pipeline and workflow execution system support a rich set of guardrails and model behaviors, naming all outputs according to metadata, leveraging model metadata for input restrictions, and allowing pipelines to produce multiple datacube outputs, copying the output of a model to multiple nodes in the DAG, allowing one-to-many pipeline construction.

5. Transforms

SuperMaaS implements a set of standard transforms as part of it's core interface, meant to ease the process of model input/output impedance matching by providing standard transformations on representations, units, etc, and allowing for compositional transforms that understand algebra on unit dimensions to ensure correctness. In this section, we describe these capabilities as currently developed.

5.1. Expanding SuperMaaS's Sphere of Influence

SuperMaaS was, as the name suggests, built to be a robust modeling service. With the framework for execution plans in place, as described above, it can act as a robust data pre- and post-processing service as well. A "transform" is morally a function from datacube(s) to datacube(s), and is a mechanism to retrieve the result of a model's execution, manipulate it in some user-specified way, and make the result available for analysis (by an Analyst) or further transformation (in the rest of a pipeline specified by a General Modeler). We authored several transforms as proofs-of-concept, which help to illustrate the substantial potential power of a transform.

As with models, the data manipulation functionality at the core of a transform may be authored in any language - the only restriction is that language's ability to interact with the SuperMaaS service via HTTP requests. We chose Python as an exemplar language for the transforms we authored, having already written a set of modeling "utilities" in Python that speed the process of interacting with the REST API that SuperMaaS exposes.

The process of registering a transform is almost identical to that of registering a model, and doesn't bear rehashing. We were able to reuse almost all of the metadata format and schema we developed, originally only for models, for transforms.

As described above, transforms form an integral part of almost any SuperMaaS execution plan. Often, a General Modeler may need/want to include a slew of serial transformations on data to

get it into the form they desire. It would, then, behoove a transform author to ensure that their functionality is relatively atomic - in a thriving SuperMaaS ecosystem, this would increase the chances that a General Modeler could find an ”off-the-shelf” transform, or set thereof, that suits their needs, rather than needing to go request or write one themselves.

(The power of transforms in SuperMaaS stems not from the power of each of the following programs - transforming from one set of units to another isn't an unsolved problem - it is the ecosystem in which those basic blocks of functionality can exist, underpinned by our execution pipeline infrastructure and able to be composed, manipulated, and parameterized at will by a General Modeler.)

5.2. Examples

Three transforms we authored during the course of the program stand out as exemplars of balancing the desired atomicity of a transform with the potential breadth of functionality. Each has the general flavor of translating between a number of syntactically different, but semantically similar, representations of data - a classic data processing task - hence the idea of a ”matrix” of functions.

Unit matrix: Given a specification of input units and output units, assuming dimensional agreement between the two, this transform will convert datacube data from one to the other. This transform, based largely on the `pint` library in Python, can comprehend all SI base units (e.g. kg, m, s, etc.), many hundreds of named compositions of those units (e.g. Newtons, horsepower, etc.), and any valid algebraic combination of any base or composed units, making the potential input and output units of this transform theoretically limitless. It also includes facilities for limited natural language specification of such units (e.g. ”kilograms per meter squared”, ”pound ft”, etc.).

Spatial matrix: Converts datacube data from and to decimal latitude-longitude pairs, sexagesimal (i.e. degrees-minutes-seconds) latitude-longitude pairs, and administrative region names. Converting from the latter format to either of the former two requires GADM encoding/decoding data, which is readily available online and can be prepackaged into the transform without user intervention.

Temporal matrix: Given data in one representation of a temporal point, as well as a specification of that format and of the desired format, we can convert the data to the desired format. These specifications leverage widely-used and well-understood time format codes underpinned by ISO 8601. Additionally, this transform can, in many cases, infer the format of temporal data it is given, allowing increased flexibility in cases of e.g. temporal data of differing formats in the same datacube/dataset or occasional corrupted data values.

```
[
    "K_alpha_Cu_d_220",
    "K_alpha_Mo_d_220",
    "K_alpha_W_d_220",
    "RKM",
    "UK_force_ton",
    "UK_hundredweight",
    "UK_ton",
    "US_force_ton",
    "US_hundredweight",
    "US_international_ampere",
    "US_international_ohm",
    "US_international_volt",
    "US_therm",
    "US_ton",
    "abampere",
    "abcoulomb",
    "aberdeen",
    "abfarad",
    "abhenry",
    "abohm",
    "degree",
    "degree_Celsius",
    "degree_Fahrenheit",
    "degree_Rankine",
    "degree_Reaumur",
    "delta_degree_Celsius",
    "delta_degree_Fahrenheit",
    "delta_degree_Reaumur",
    "denier",
    "didot",
    "dirac_constant",
    "dram",
    "dry_barrel",
    "dry_gallon",
    "dry_pint",
    "dry_quart",
    "dtex",
    "dyne",
    "electrical_horsepower",
    "electron_g_factor",
    "electron_mass",
    "hectare",
    "henry",
    "hertz",
    "hogshead",
    "horsepower",
    "hour",
    "hundredweight",
    "impedance_of_free_space",
    "imperial_barrel",
    "imperial_bushel",
    "imperial_cup",
    "imperial_fluid_drachm",
    "imperial_fluid_ounce",
    "imperial_fluid_scruple",
    "imperial_gallon",
    "imperial_gill",
    "imperial_minim",
    "imperial_peck",
    "imperial_pint",
    "imperial_quart",
    "inch",
```

Figure 9: Example units transforms supported by the current version of SuperMaaS

6. Initial Galois UI

Galois worked on an initial UI to support the Domain Modeler and General Modeler workflows for listing and running models within the SuperMaaS system.

6.1. Model Listing

In order to support General Modelers' ability to find registered models within the SuperMaaS system, Galois developed a simple data-grid interface that listed all the models currently registered

within the system, support simple filtering based on a few initial criteria:

- A classification of the nature of the decision being informed by the model
- A modeler-defined categorization of the model
- The name of input(s) exposed by the model

Once the user filtered the set of models down to the few they were interested in, the user could then select a row in the results table to see relevant details related to the model:

- A modeler-provided description of the model
- A table of exposed inputs, detailing
  - The name of the input
  - The type of input (e.g. ”date”, ”float”, ”boolean”, etc.)
  - A modeler-supplied description of the parameters’ purpose
  - The default value for the parameter (if applicable)
- A table of available outputs, detailing
  - The name of the output
  - The type of output (e.g. ”date”, ”float”, ”boolean”, etc.)
  - A modeler-supplied description of the output’s purpose

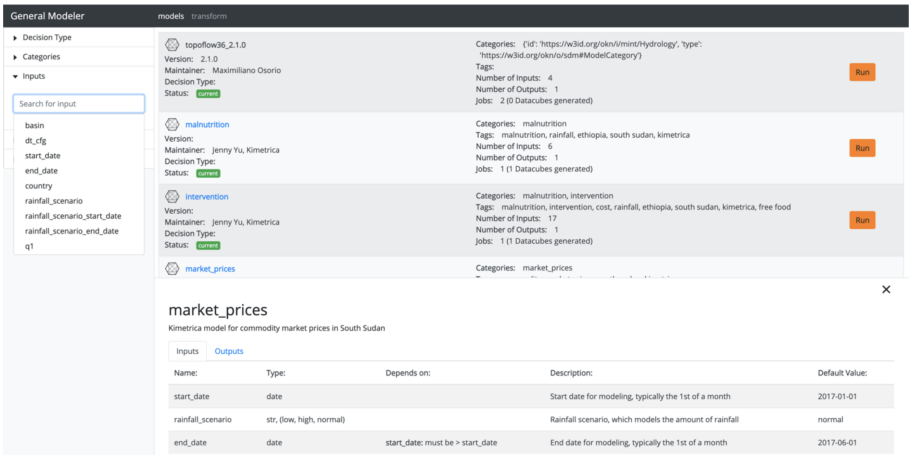
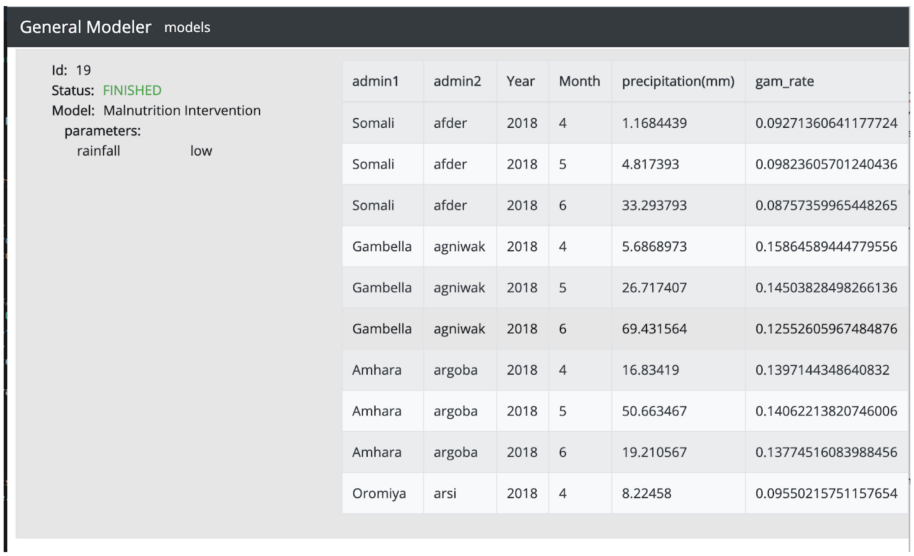


Figure 10: General Modelers Interface: Model Discovery

In addition to being able to search for models, the UI allowed for the perusal of generated datasets, where the user could select a given dataset and see a brief summary of the data and a sample listing of values.



admin1	admin2	Year	Month	precipitation(mm)	gam_rate
Somali	afder	2018	4	1.1684439	0.09271360641177724
Somali	afder	2018	5	4.817393	0.09823605701240436
Somali	afder	2018	6	33.293793	0.08757359965448265
Gambella	agniwak	2018	4	5.6868973	0.15864589444779556
Gambella	agniwak	2018	5	26.717407	0.14503828498266136
Gambella	agniwak	2018	6	69.431564	0.12552605967484876
Amhara	argoba	2018	4	16.83419	0.1397144348640832
Amhara	argoba	2018	5	50.663467	0.14062213820746006
Amhara	argoba	2018	6	19.210567	0.13774516083988456
Oromiya	arsi	2018	4	8.22458	0.09550215751157654

Figure 11: General Modeler's Interface - Datacube Exploration

6.2. Model Execution

The principal role of the General Modeler is ensuring that Analysts they are working with have access to datacubes which allow them to visualize, understand, and develop predictions, understanding, and policy outcomes. While SuperMaaS worked to abstract models provided by Domain Modelers into more usable, proceduralized, variants with fewer, more understandable inputs, and formats their outputs into the generalized schema of a datacube, model operation is still a task that requires generalized technical expertise.

While the December Experiment contained a number of pre-generated datacubes to aid in rapid testing of both SuperMaaS and CauseMos, such pre-generated datacubes only cover a small amount of the possible model spaces of interest, and did not exist for new models registered with SuperMaaS. The primary way that General Modelers could assist Analysts was in the generation of new datacubes. Figure 3 shows the SuperMaaS model registry interface, which displayed all models in the registry to General Modelers; allowing them to search and facet the registry based on their needs; contained information on the model's purpose, inputs, and maintainer; and finally linked to the model execution page.

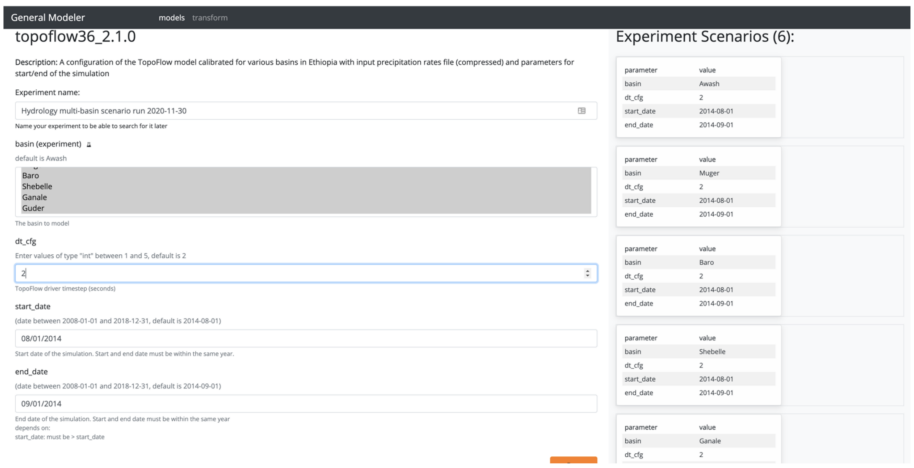


Figure 12: General Modeler's Interface - Model Execution

Once a model had been identified, the General Modeler would navigate to the Model Execution and Experimental Design page associated with a given model, like the one for Topoflow shown in Figure 3. This page was populated using metadata provided during model registry, and attempted to build a usable, understandable, interface that adequately documents the use cases, limitations, requirements, and default values for the model.

SuperMaaS exposed model inputs using our initial draft of a design of Experiments interface, generating, displaying, and helping the General Modeler understand the experimental runs that will be generated on the basis of the selected input ranges, and finally resulting in a set of jobs for the SuperMaaS execution engine to run.

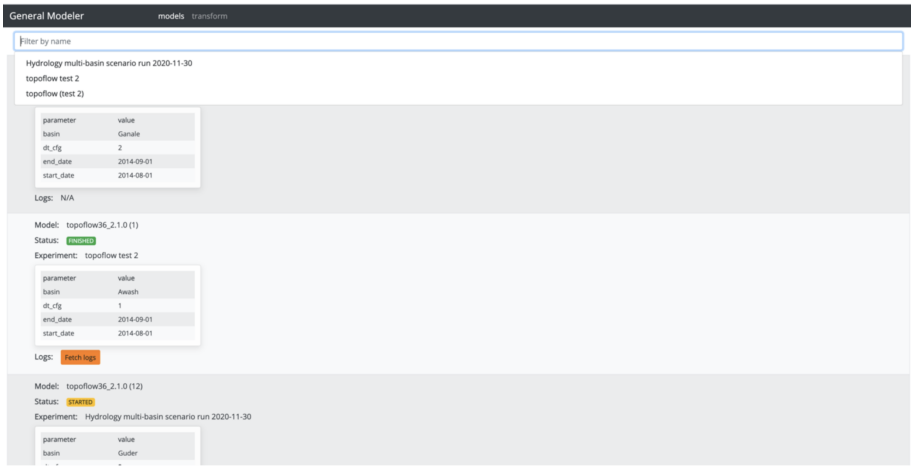


Figure 13: SuperMaaS Job Interface

In Figure 4, an example page showing the results of several running experiments in SuperMaaS can be seen. General Modelers had access to this interface in order to query running jobs, fetch logs for successful or failed jobs, and access the datacubes that result from a job, or set of jobs.

It should be noted that this initial UI approach was dropped in favor of leveraging the Dojo system developed by Jataware for the registration of models and using Causemos for the model/-datacube search/listing as well as the model execution.

7. Model Emulation

We frame the model emulation problem as follows. Given a model M with some numeric input parameters $p_1, p_2, ..., p_m$, we wish to construct a computationally-inexpensive model M', such that $M'(p_1, p_2, ..., p_m) \approx M(p_1, p_2, ..., p_m)$; we call M' an *emulator*. A model emulator can be useful when the original model has a long running time and/or requires significant computational resources, but results must be obtained rapidly, with limited computing power, e.g., when formulating a response to a humanitarian crisis, while in the field. Another potential benefit of emulation is to provide insight into the operation of a "black-box" model, provided that the emulator is sufficiently interpretable; this is the case with our approach, since we represent the emulator as piecewise linear function, consisting of a set of simple linear equations. A related benefit is that emulation can sometimes help with model debugging, as we show in section Section 7.5. Finally, emulation can be used to assess the complexity of a model: a model that can be emulated with high accuracy and low computational overhead is, presumably, simpler than one that is difficult to emulate.

With these goals in mind, we extended the functionality of SuperMaaS to automatically construct an emulator, given a registered model, and to evaluate this emulator by comparing its output (for different parameter values) with that of the original model. Thus far, the SuperMaaS version of the emulator has only been fully applied to the MaxHop model, which outputs the probability of encountering locusts for different geographic locations; the input parameters are the increase in temperature and the increase in precipitation. While there might not be a strong need to emulate MaxHop in practice, because it is already fairly efficient in terms of computation, it provides an interesting test case and a useful illustrative example. However, we began to apply a portion of the algorithm to the DSSAT model, which is much more computationally expensive, and also applied a standalone version of the algorithm to two compartmental epidemiological models that were not registered in SuperMaaS, namely, the classical SIR model and the double-pandemic SEIRP model.

In the following, we provide some background on linear emulation, and then outline the components of our software. Subsequently, we present two emulation approaches: a *local* approach, which may provide a high degree of accuracy near some set of parameter values $p_1, p_2, ..., p_m$, but with potentially low accuracy in regions of the parameter space that are far from these values, and a *global* approach, where there is an attempt to provide high accuracy for all parameter values. We demonstrate the effectiveness of the global approach on the MaxHop model, and later, describe the application of the approach to the DSSAT, SIR and SEIRP models.

7.1. Linear Emulation Background

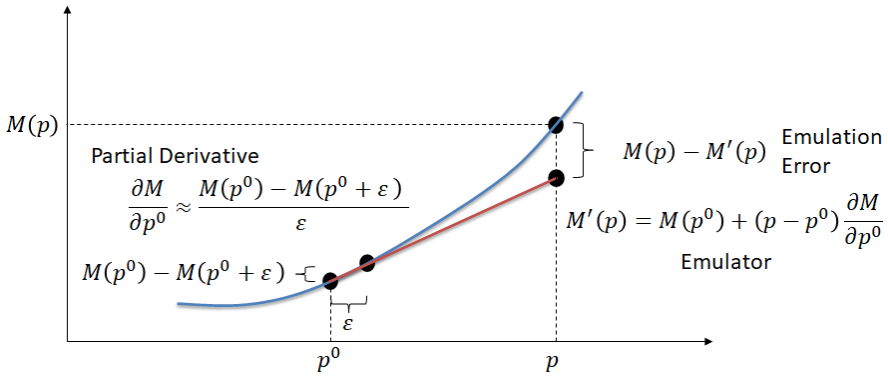


Figure 14: An illustration of linear emulation.

Suppose, for simplicity, that the original model M takes in just a single parameter p. We can perform the *sensitivity analysis* of the model M with respect to p by evaluating the model with some *baseline* value of p (which we call p^0), then evaluating it again on some *perturbed* value $p^0 + \epsilon$, and comparing the output of the model for the baseline and the perturbed case. As we show in Figure 14, if we divide the output difference $M(p^0 + \epsilon) - M(p)$ by ϵ, then we are approximating the derivative of M with respect to p near p^0.

Now, given some new parameter value p, we can perform *linear emulation* by multiplying the input difference $p - p^0$ by the derivative, in order to obtain the expected change in the model output, relative to its baseline output $M(p^0)$. By adding this change to the baseline output, we obtain the emulated output $M'(p)$. If the model accepts multiple input parameters, then we compute the expected change for each parameter, sum the changes, and add the sum to $M(p^0)$.

Finally, we can compute the *emulation error* as the difference $M(p) - M'(p)$. If M is a linear model, then we would expect the error to be zero, since linear emulation is used; otherwise, the error will likely be non-zero. By considering the magnitude of the error, we are, in a sense, performing *nonlinearity testing*, i.e., capturing the degree of nonlinearity in the model.

We note that in Figure 14, the model has a single output, represented by the vertical axis. In actuality, the above operations are repeated for every value of every numeric dependent output variable that is produced by a model (we assume that the model produces tabular output, where each value corresponds to a row, and each output variable corresponds to a column).

7.2. Software Components

The above emulation primitives are implemented by the following three compute nodes:

- `sensitivity-analysis`: Performs sensitivity analysis on a model M. As input, it takes $m + 1$ datacubes, the first datacube produced by the model for some baseline set of numeric parameters (of type `float`), and each of the other datacubes produced by the model with some parameter perturbed from the baseline. As output, it produces a single datacube,

which captures the sensitivity of the model to the perturbation in each of the parameters; it also contains the baseline set of parameters.

- `linear-emulation`: Uses the datacube that was output by the *sensitivity-analysis* node to approximate the output of the original model, given some set of parameters. Together, the sensitivity analysis datacube (a.k.a. the *linear emulation datacube*) and the linear emulation node implement the *emulator* M'. The linear emulation node can be passed the id of the linear emulation datacube (as the `sensitivity-analysis-cube-id` scalar parameter), along with values of `float` model parameters, and it will produce a datacube with the same schema as the datacubes produced by the original model M.
- `nonlinearity-testing`: Compares the output of the original model M and above emulator M' on some set of parameters, to compute the emulation error.

The compute nodes, along with the MaxHop model itself, are combined into two pipelines: `MaxHop-build-emulator-pipeline.json` and `MaxHop-apply-emulator-pipeline.json`. The former pipeline is used to build the emulator (by this, we mean generating the linear emulation datacube). The latter pipeline evaluates the emulator on some new set of parameters. The pipelines can be reconfigured to perform emulation on other models; in the future, it should be possible to automatically generate the pipeline files automatically, from the metadata of the model in question.

Finally, we provide a command-line client program called `emulation.py`; it interacts with a running instance of SuperMaaS to execute the pipelines, in order to build and evaluate emulators.

7.3. Local Emulation

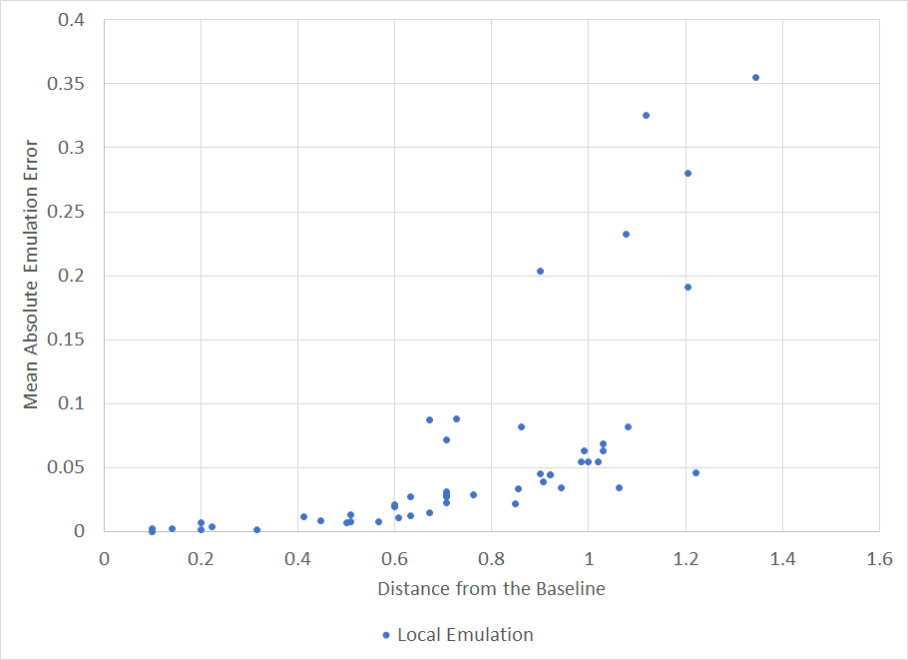


Figure 15: Local emulation errors, as a function of distance to the baseline.

In an initial experiment, we applied the approach of Section 7.1 to the MaxHop model, which was registered in SuperMaaS with two parameters exposed: `meanTempIncrease` and `annualPrecipIncre` We used baseline values of 0 for both of these parameters, and performed sensitivity analysis by perturbing each parameter independently by 0.1. We then used the resulting linear emulation datacube to perform emulation 50 times, using randomly-generated values for `meanTempIncrease` and `annualPrecipIncrease` each time, and computing the emulation error by comparing the output of the emulator with the output of the MaxHop model for the same parameter values. In Figure 15, we plot the absolute emulation errors, averaged over the all of the dependent output variable values (i.e., over the locust probabilities for different geographic regions), as a function of the Euclidean distance of a given set of parameter values (which can be viewed as a point in the parameter space) to the baseline values. Not surprisingly, errors tend to rise considerably as the distance to the baseline increases, illustrating that a local linear emulator may be insufficient at accurately covering the entire parameter space.

7.4. Global Emulation

To provide more effective emulation across the parameter space, defined by the ranges $p_1^{min} \leq p_1 \leq p_1^{max}, ..., p_m^{min} \leq p_m \leq p_m^{max}$, we can generate multiple linear emulation datacubes, each with a different baseline. But what baselines should be chosen? One approach is to evenly sample the parameter space with a grid of points, and to use each of these points as a baseline for a linear emulation datacube; however, it is possible that some regions of the parameter space may need to be sampled more or less densely than others (if they have more or less nonlinearity, respectively). To enable this sort of adaptive sampling, we use a "divide and conquer" algorithm, which first attempts to build a single linear emulation datacube for the entire parameter space. As the baseline, it uses the "lower" corner $(p_1^{min}, ..., p_m^{min})$ of the parameter space (e.g., `meanTempIncrease` $=$ `annualPrecipIncrease` $= -1$); when performing sensitivity analysis, it perturbs a given parameter p_i by $\epsilon = p_i^{min} + 0.5(p_i^{max} - p_i^{min})$ (i.e., to the midpoint between the maximum and minimum value, e.g., `meanTempIncrease` $=$ `annualPrecipIncrease` $= 0$; thus, potentially, ϵ may be quite large). The resulting emulator is tested only once, on the "upper" corner $(p_1^{max}, ..., p_m^{max})$ of the parameter space (e.g., `meanTempIncrease` $=$ `annualPrecipIncrease` $= 1$); the hypothesis is that if we test on the point that is as far away as possible from the baseline, then we are more likely to uncover a large emulation error. Of course, this is by no means guaranteed, but performing just a single test has the advantage of reducing the number of times that the model must be executed.

If the resulting emulation error for $(p_1^{max}, ..., p_m^{max})$ does not exceed some given threshold, then nothing more is done, and the single linear emulation datacube is returned. Otherwise, the parameter space is partitioned into two equal regions, on some parameter; if parameter p_i is chosen for this purpose, then one region will have the range $p_i^{min} \leq p_i \leq p_i^{split}$, and the other will have the range $p_i^{split} \leq p_i \leq p_i^{max}$, where $p_1^{split} = p_i^{min} + 0.5(p_i^{max} - p_i^{min})$, with ranges for other parameters being identical for both regions. Then, the algorithm is recursively applied to both regions, and thus keeps subdividing the space until all regions have a sufficiently low emulation error. The datacubes with sufficiently low emulation errors (for sufficiently small regions of the parameter space) are returned, to later be used by the linear emulation node.

(As an additional technical detail, to choose the parameter p_i on which to partition the space, the algorithm attempts to partition on every parameter, and performs the emulation test for the "lower" of the two regions each time; the partition that results in the lowest emulation error is ultimately chosen.)

To our knowledge, our global emulation algorithm is somewhat novel; while a related "divide and conquer" piecewise linear approximation technique has been published in "L. Grama and C. Rusu, Hilbert Transform by Divide-and-Conquer Piecewise Linear Approximation, ECCTD 2011 20th European Conference on Circuit Theory and Design, pp. 398-401, 2011", it has been designed specifically for the case where there is only a single parameter, and cannot be easily extended to the multi-parameter case.

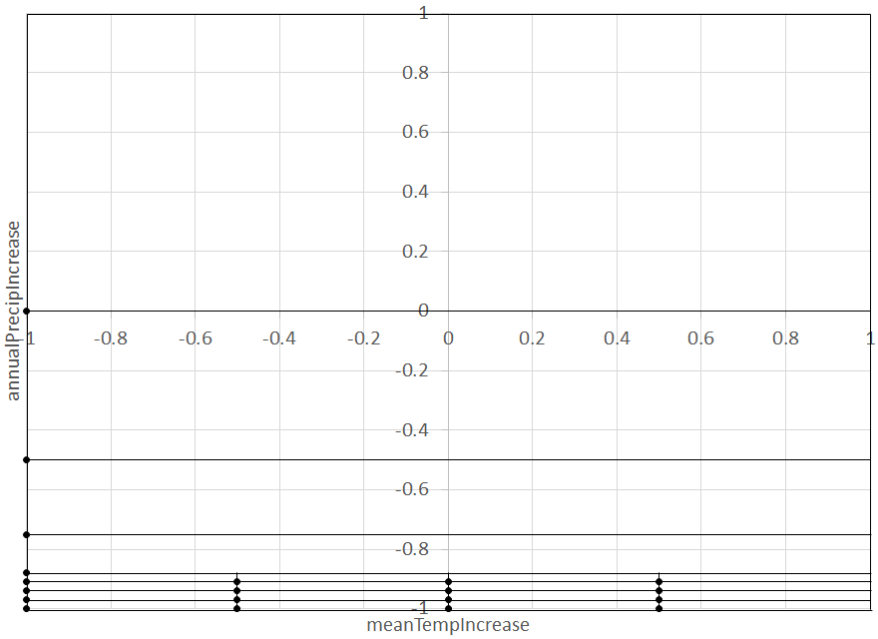


Figure 16: A partitioning of the MaxHop parameter space into regions, where each region is approximated with a single linear equation. Black lines denote region boundaries, while black dots denote baseline parameter values for the different regions.

We applied the algorithm to the MaxHop model, with an error threshold of 0.04 (for deciding whether or not to partition a given region); the algorithm required in the order of 20 minutes of computation time. Figure 16 illustrates the resulting partitioning of the parameter space. Interestingly, for low precipitation increase values, a denser sampling / finer partitioning is performed by the algorithm, in order to maintain a low emulation error; this suggests that for such values, the model exhibits more nonlinearity in its behavior.

We then reran our 50 trial experiment with the resulting global emulator, which is defined as the set of local emulators, one for each region. During each trial, given a set of input parameter values, the `linear-emulation` node automatically selects a linear emulation datacube that covers the region of the parameter space that contains these values. If we add the resulting mean

absolute error values (for the same distances to the original baseline of `meanTempIncrease` = `annualPrecipIncrease` = 0) to our plot, we observe, in Figure 17, that emulation errors are now much lower, thereby demonstrating the potential effectiveness of the global emulation algorithm (at least, for some models).

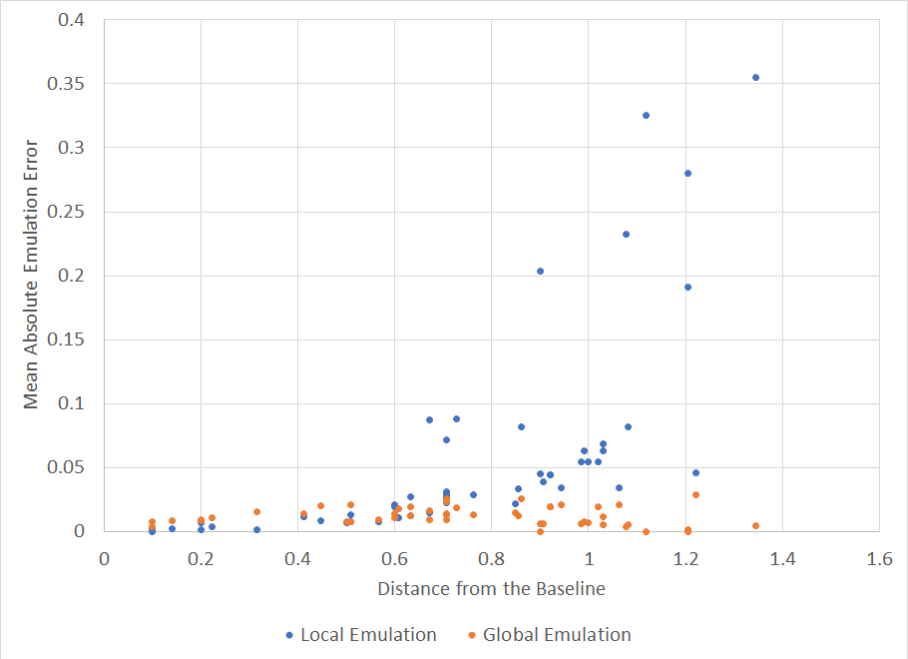


Figure 17: Local emulation errors, as a function of distance to the baseline.

7.5. Application to the DSSAT Model

Presently, our model emulation procedure assumes that a change in a varied parameter will not affect the exact set or the order of output variable values (rows in the datacube) that are output by a model. Some models can break this assumption; for example, for the DSSAT and Cycles models, changing a `float` parameter can somewhat alter the exact subset of geographic locations that are included in the datacube. Nonetheless, we did make an initial attempt to gauge the sensitivity of DSSAT to its parameters. In the process, we have discovered a bug in the model configuration: specifically, for the Maize crop, the model's outputs were not affected by the rainfall parameter. We reported the bug to the DSSAT team at the University of Florida. In the future, we hope to adjust our implementation of the algorithm to handle situations where the number of output variables is affected by changes in input parameters.

7.6. Application to Epidemiological Models

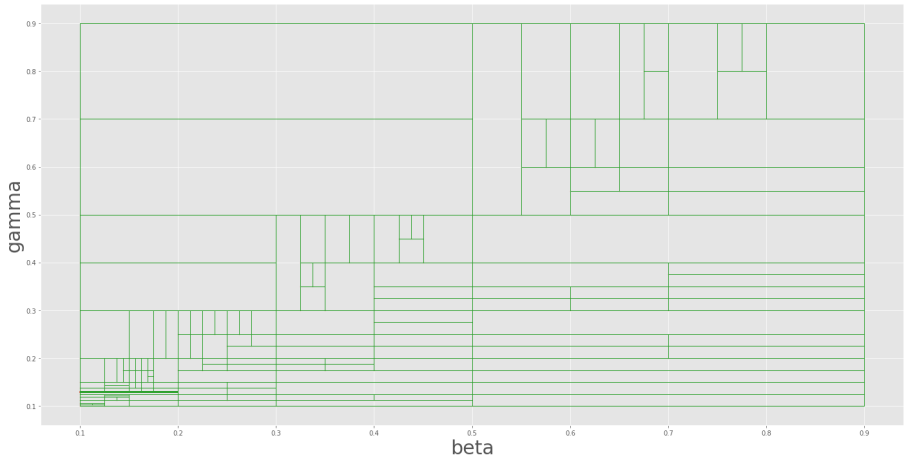


Figure 18: A partitioning of the SIR parameter space into regions.

More recently, we applied the global model emulation algorithm to compartmental COVID-19 models, specifically, the SIR and the SEIRP model. While these models are computationally-inexpensive, we wished to gauge their complexity, in terms of nonlinearity. In Figure 18, we show a partitioning of the parameter space that was produced by the algorithm for the SIR model, where two parameters were varied: `beta` (the infection rate) and `gamma` (the recovery rate); the initial proportions of susceptible (S), infected (I) and recovered (R) individuals were fixed at 0.99, 0.01 and 0, respectively. For simplicity, the output of the model was constrained to a single, scalar value, namely, the number of recovered individuals after 200 time steps (though the algorithm can work with vector outputs, as we have previously shown for the MaxHop model). An error threshold of 0.01 was used, and recursion was limited to a depth of 15 (a region was not partitioned beyond this limit, even if the test error exceeded 0.01). For each region, a simple linear equation is produced, tying the output to parameter values, e.g., $\texttt{output} = 9.209563200470313 * \texttt{beta} - 6.080993989792093 * \texttt{gamma} + 0.11967053217550297$, for the region bounded by $\texttt{beta} \in [0.1, 0.125]$ and $\texttt{gamma} \in [0.1, 0.103125]$. By examining this equation, we can easily see that the proportion of recovered individuals grows with the infection rate `beta`, but decreases with the recovery rate `gamma`; the approach thus helps to provide *explainability* for black-box models. We evaluated the accuracy of emulation by randomly generating `beta` and `gamma` values, and computing the absolute emulation error each time; this was done 20 times for each region. The expected absolute emulation error (computed as the average of emulation errors, with each one weighted by the relative size of the region within which the input parameters were generated) was 0.01. All observed absolute errors were below 0.1; given that the number of recovered individuals can range in $[0, 1]$, we can say that emulation error was always under 10%; thus, emulation is fairly effective for the SIR model.

We have also applied the emulation approach (with the same error threshold and recursion limit) to the SEIRP model, described in the paper "T. W. Ng, G. Turinici and A. Danchin, A double epidemic model for the SARS propagation, BMC Infectious Diseases 3, 2003". For this model, we used individual counts, rather than proportions, with initial conditions fixed as $Ip0 = 500,000$,

$S0 = 6,300,000$, $E0 = 100$, $I0 = 50$, $R0 = 0$ and $Rp0 = 0$. The model has five, rather than two varied parameters, and emulation was much more difficult, with tens of minutes of computation required, relative to just a few seconds for the SIR model. The median error was 163, but the expected error was $192,635$, and the maximum observed error was $7,463,367$; in other words, throughout most of the parameter space, emulation error is low, but in a few areas, it can be quite high; this suggests that the nonlinearity/complexity of the SEIRP model is much higher than it is for the SIR model.

8. Model Skill Assessment

Model skill assessment has become a central concern during the COVID-19 pandemic. Over the course of the pandemic, a large number of models, of varying levels of credibility, were released by domain scientists, mathematicians, and modelers. Many of the early models that were advanced, such as the cubic model, or IHME's logarithmic model, proved to have low predictive power, and had low skill when it came to predicting new cases, deaths, hospitalizations, and other important metrics. Many times this results from the application of "curve fitting" resulting in models that predicted future behavior on the basis of rough mathematical fittings of prior data, but with little to no understanding of the underlying mechanistic reasons for the evolution of the measures of interest. While curve fitting has worked reasonably well in prior pandemics, such as the rinderpest pandemic in the UK, simple curve-fitting only works when a disease is spreading under constant conditions. The conditions of the COVID-19 pandemic, by comparison, are highly heterogeneous both geographically and demographically. The implementation of social distancing, masking, and other non-medical interventions were haphazard without consistency across nations or even individual states and counties, meaning the pandemic has had far more complex behaviors than can be captured and predicted by simple models.

Figure 19 shows an example of COVID-19 forecasts from the ensemble model (in red), showing predictions one week, two weeks, three weeks, and finally four weeks in advance of a given point, compared to the actual outcomes witnessed in practice. As can be seen, the results of the ensemble model (and even individual model) forecasts are often quite different from the reality, showing poor model skill.

In addition to poor model skill, we can also see, from Figure 20, that individual models in the CDC ensemble forecast have wildly different results, resulting from different assumptions at the mechanistic level, or for the underlying mathematical functions which approximate these results. There is a clear need for better measures of model skill to help researchers and policymakers sort credible models from those that are not credible and to understand the fitness for purpose that credible models exhibit, including the limitations of that fitness for purpose.

Furthermore, in addition to the issue of model credibility, model reproducibility has been a major challenge during the pandemic. The source code for models is often unavailable, and in cases where it is made available, that availability is often just the base code, or algorithms without the necessary parameterization. In other examples, such as the IHME model, public repositories have not been kept up to date with changes in the model, and sometimes the available code is actually removed at a later date. Without reproducibility, if a model is found to be credible or to have

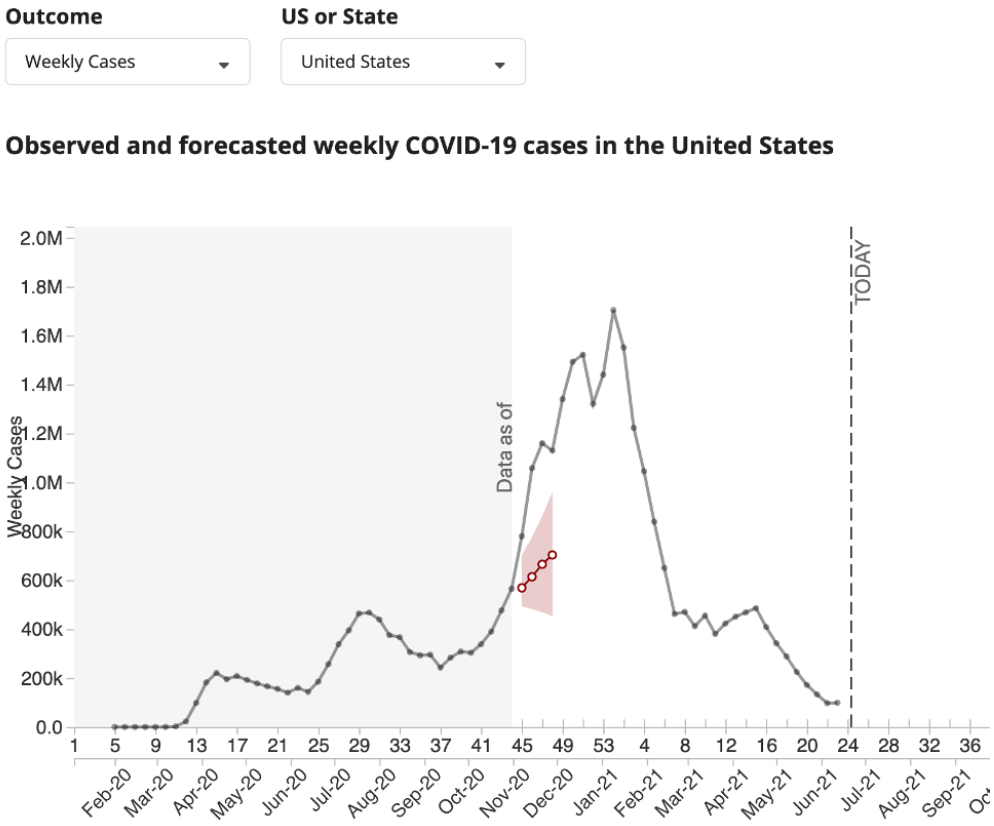


Figure 19: Example COVID-19 Forecast from the CDC Ensemble Model

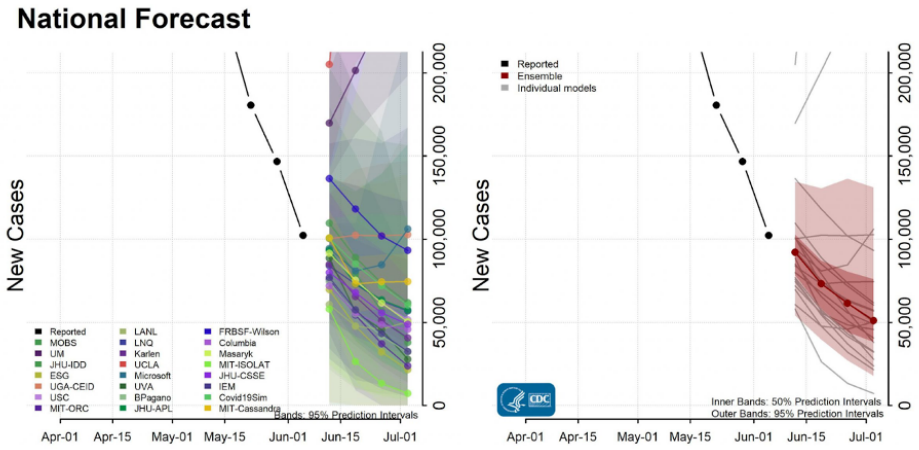


Figure 20: Trajectories of various models in the CDC National Forecast ensemble showing wildly divergent results based on different mechanistic and curve-fitting properties of the underlying models.

skill at prediction, they cannot be effectively reused or generalized for broader use. Additionally, without knowledge of the underlying parameters, the model may not be easily tunable to produce the correct results.

In the following, we describe our efforts in examining current approaches to model skill assessment, and identifying certain areas where these approaches can be improved. First, we consider *baselines*, i.e., simple models against which more sophisticated models can be compared, in order to gauge their performance. Here, we find that a constant-assumption model (which simply predict that the number of COVID-19 cases will not change from its present value) provides a strong baseline, and one that can be surprisingly difficult to beat. We also look at existing *metrics*, which have been used to quantitatively assess model skill against the baseline. We apply these metrics to assess both early and recent COVID-19 forecasting models against both early and recent ground truth data, and in the process, identify limitations of the metrics. Then, we describe an initial qualitative assessment of COVID-19 forecasting models, which has identified significant problems of reproducibility. Finally, as a simple but relevant example of working with a concrete forecasting model, we present some work on fitting the SIR model to COVID-19 case data for the purposes of forecasting, and show that while forecasts can be accurate, this requires unrealistic parameters; in other words, while the model has "curve" skill, it does not have "mechanistic" skill.

8.1. Skill Assessment Baselines

As potential baselines for assessing COVID-19 forecasting model skill, we have considered three simple forecasting models:

- A *Constant model*, which assumes that the number of new weekly COVID-19 cases will remain the same at week $t + h$ as it has been during the current week t (here, h is the *forecast horizon*). Constant-assumption forecasts have already been used as baselines for the assessment of model skill on the CDC COVID-19 ForecastHub (they are produced by a model known as *COVIDhub-Baseline*), and these are oftentimes more accurate than forecasts that are made by some of the more sophisticated models showcased by the CDC.
- A *Linear model*, which assumes that the rate of change in the number of new cases will remain the same. Specifically, for a horizon h, the linear model defines a linear function by using the number of cases reported during weeks t and $t - h$, and uses this function to forecast the number of cases for week $t + h$; i.e., it predicts that the change in case numbers from week t to $t + h$ will be the same as the change in case numbers from week $t - h$ to week t.
- A *Polynomial model* for some degree d, which uses the NumPy `polyfit` function to fit a polynomial to case counts from all of the past $hd + 1$ weeks, from week $t - hd$ through week t, with the week number being the time parameter to the polynomial. We then pass in week $t + h$ to the polynomial to obtain the forecast.

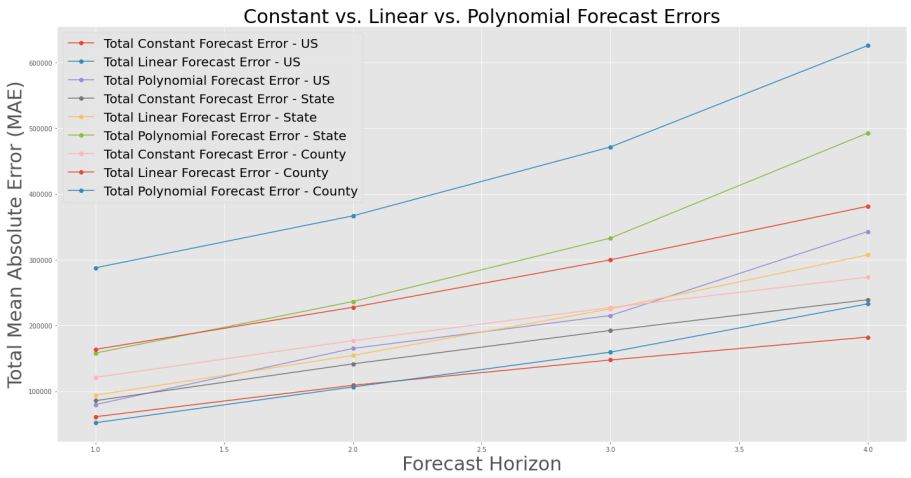


Figure 21: Constant, Linear and Polynomial (for $d = 2$) model forecast errors, as a function of the forecast horizon h. For a given geographic area (the entire United States, some state, or some county), we compute the Mean Absolute Error (MAE); these MAE values are then summed over geographic areas (states or counties).

We evaluated the models using ground truth data made available on the COVID-19 ForecastHub (https://github.com/reichlab/covid19-forecast-hub), through the middle of July 2021. As Figure 21 shows, the Constant model typically outperforms the Linear model, except for short-range US-level forecasts, while the Linear model outperforms the Polynomial model; more generally, we have found that forecast accuracy tends to decrease with increasing degree (presumably, due to overfitting), and the Constant model tends to provide the best baseline.

We have also begun work on novel approximation techniques based in Koopman Theory, extended with Deep Neural Networks to enable constructed operators to learn their parameters from empirical data, to improve the results of the Constant, Linear and Polynomial models. However, as we discuss below, we have already found utility in simple approximations (such as the Constant approximation) when judging the model skill exhibited by mechanistic models, and other curve-fitting models.

8.2. Skill Assessment Metrics

We subsequently applied some of the model skill assessment methods from the paper "E. Cramer et al, Evaluation of individual and ensemble probabilistic forecasts of COVID-19 mortality in the US, medRxiv 2021.02.03.21250974, 2021", to forecasts from models on the COVID-19 ForecastHub, as well as the Constant and Linear models. Our current assessment includes models, forecasts, and ground truth data that were not available at the time of publication, in February of 2021. We note that we identified a number of data quality issues in the forecast data that is made available at the COVID-19 ForecastHub aforementioned repository (https://github.com/reichlab/covid19-forecast-hub); for example, some models sometimes recorded two forecasts for the same week and geographic location; all such duplicate forecasts were removed, and not taken into account during evaluation. In Figure 22, we provide *relative Mean Absolute Error (MAE)* metrics for the models that forecast COVID-19 death counts 4 weeks into the future; as explained in the paper, a relative

MAE below 1 indicates that the model outperforms the COVIDhub-Baseline model. This model is very similar to the Constant model, but made forecasts using older versions of ground truth data (which underwent occasional revisions), whereas our Constant model used the most recent ground truth data. Because the most recent ground truth data was used in our model skill assessment, the Constant model slightly outperformed the COVIDhub-Baseline model. More generally, the plot shows that approximately half of the COVID-19 ForecastHub models are not able to outperform the COVIDhub-Baseline model according to the relative MAE metric; this is in line with the results reported in the paper (which evaluated fewer models). This suggests that the outperformed models may not be capturing the right dynamics of the infection, and as a result have low model skill. We have been operationalizing this technique into our framework to compare and contrast models and data that measure the same concept, to allow models to be sorted and triaged by model skill.

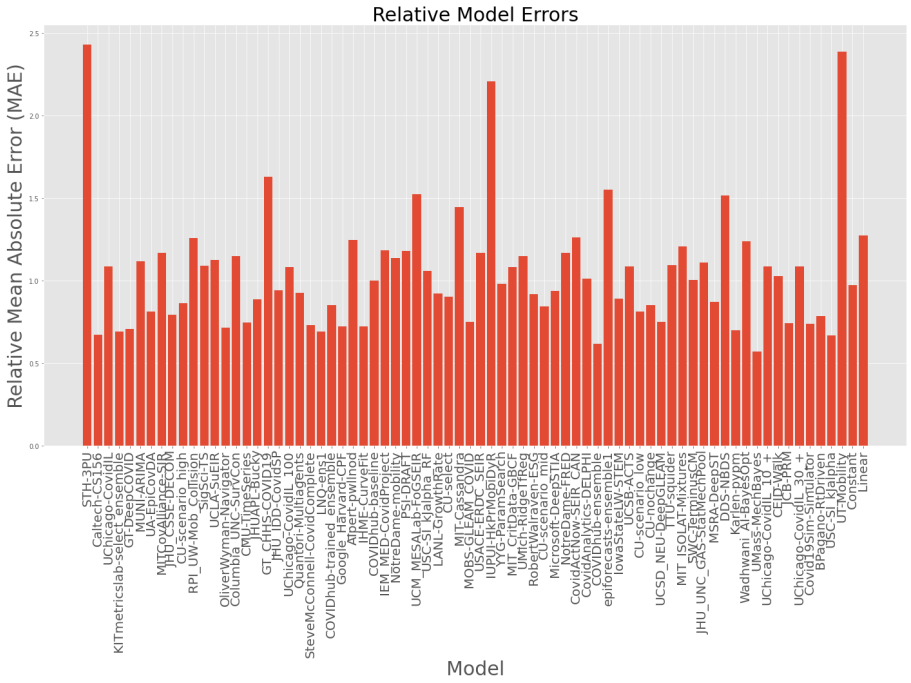


Figure 22: Relative MAE metrics for various models.

In the course of performing the above assessment, we identified a limitation in the metrics that were used in the aforementioned paper. In comparing forecasting models against each other, a key challenge is that not all models provide forecasts for the same time range; for example, a model that was added to the COVID-19 ForecastHub in 2021 will not have forecasts for 2020. If a model provides forecasts only for time periods when forecasting is "easy" (e.g., when the number of cases does not undergo significant fluctuations), then it may appear to outperform a model that provides forecasts for "difficult" time periods. The relative MAE metric, as well as the *relative Weighted Interval Score (WIS)* metric (also described in the paper, and used for assessing models that output forecasts as probability distributions), were designed to address this challenge, by making pairwise comparisons of models against each other, with each pair being evaluated only on forecasts that the two have in common (details are provided in the paper). However, a limitation is that the metrics do not take into account the *number* of forecasts made by models; thus, if there was a

model that produced very accurate forecasts for a very narrow time range (say, the first four weeks of 2021), for only a single geographic location, then that model could outperform all other models according to relative MAE and relative WIS, even though it is not necessarily more credible than other models, as the "sample" of forecasts is very small.

We believe that the limitation probably did not have a strong effect on the results that were published in the "Evaluation of individual and ensemble probabilistic forecasts..." paper: while the evaluated models were not added to the the COVID-19 ForecastHub at the same time, and did not produce the same number of forecasts, there was not a dramatic variation in the forecast counts among different models. However, Figure 22 includes models that were added much more recently, and some models have far fewer forecasts than others. In the future, it would be of interest to design metrics that overcome this limitation.

8.3. Reproducibility

We also began to perform a qualitative assessment of models (particularly, the better-performing ones) in the "Evaluation of individual and ensemble probabilistic forecasts..." paper, in terms of code availability, documentation availability, and reproducibility. Thus far, we looked at available code and documentation artifacts for the *CMU-TimeSeries*, *Covid19Sim-Simulator*, *IHME-CurveFit*, *IHME-SEIR*, *LANL-GrowthRate*, *OliverWyman-Navigator*, *UCLA-SuEIR*, *UMass-MechBayes*, and *YYG-ParamSearch* models. Among these, we have not been able to find a single model for which results could be reproduced in a reasonable amount of time. Some of the aforementioned models models (Covid19Sim-Simulator, LANL-GrowthRate, OliverWyman-Navigator) did appear to provide code; for others, code was provided (though for IHME-CurveFit and UCLA-SuEIR, it was difficult to find, as no URL was included in the publication that described the model), but depended on libraries that were not available (IHME-SEIR), or did not have adequate documentation (CMU-TimeSeries, IHME-CurveFit, UMass-MechBayes) or did not capture the model's full functionality (YYG-ParamSearch); we also note that in the case of IHME-CurveFit and YYG-ParamSearch, it was explicitly stated that the repository is no longer maintained. These obstacles to reproducibility, unfortunately, do not reflect well on current modeling practices (at least, within the COVID-19 forecasting domain), and suggests that improvements are needed.

8.4. Forecasting via the SIR Model

Although we were (thus far) unable to find a reproducible model for a more detailed evaluation, we have examined the suitability of the classical SIR model for COVID-19 forecasting. Specifically, we fit the model to datasets that were provided by Geometric Data Analysis, Inc. (GDA), capturing daily COVID-19 case counts in Florida and Georgia, within the May - September 2021 timeframe. We used some earlier portion of the time series to fit the `beta` and `gamma` parameters of the SIR model, such that for any given day t, the sum $I(t) + R(t)$ of the infected and recovered individuals (as predicted by the model) would match as closely as possible the actual cumulative case count (i.e., the total number of registered infections, up to that day). Fitting was performed via the nonlinear least squares method. The fitted model was then used to generate $I(t) + R(t)$ values for the later portion of the time series, and these were converted into daily case counts (by subtracting $I(t-1) + R(t-1)$ from $I(t) + I(t)$ for each day t).

Unfortunately, we had difficutly obtaining good forecasts when using realistic assumptions, such as a population of 10.62 million for Georgia. However, through a brute-force search over population and $I0$ values, we found that better fits could be obtained by using much larger populations, as we show in Figure 23. This illustrates the scenario where a model may have relatively high model skill in terms of forecasting accuracy, its credibility is limited by a lack of realism. While this is not a surprising result for the SIR model, which is fairly simple, we suspect that this is true of many existing models.

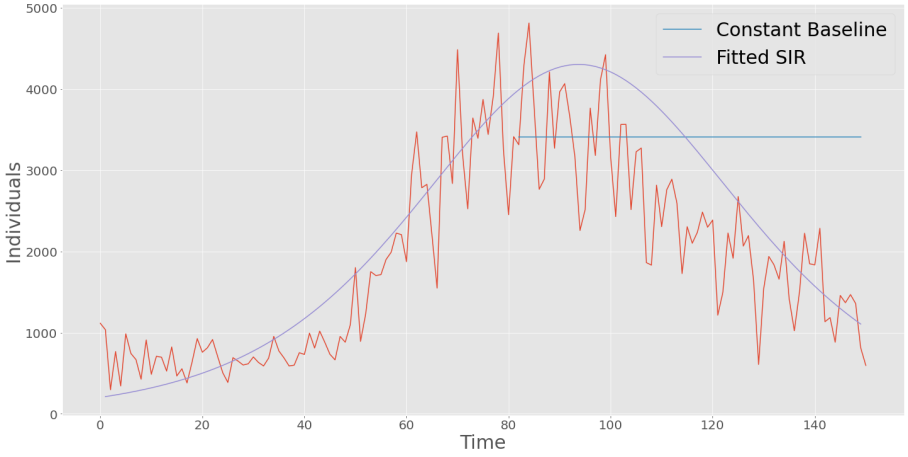


Figure 23: Fitting the SIR model to COVID-19 cases data, for the purposes of forecasting. 55% of the data points (earlier in the pandemic) were used to fit the model. The predicted daily case counts (purple curve) approximately match the data (orange curve) not only for these "past" data points, but also, for the remaining "future" points. Initial conditions were $S0 = 40,000,000$, $I0 = 20$ and $R0 = 0$, even though Georgia's population is only $10,620,000$, and there were already $27,270$ cumulative cases on May 1, 2020, the first day covered by the dataset; we note that we "shifted" the cumulative cases data by subtracting $27,270$ from each value, such that 0 cumulative cases were given for May 1.

9. University of Arizona

9.1. Introduction

From July 2021 to December 2021, The University of Arizona team designed, implemented and evaluated an information extraction framework aimed at extracting information about model interfaces, targeting the Jataware Dojo model registry. This included extending an existing tool for extracting expressions of general causal relationships, identifying and extracting descriptions of model functions, and reading for model interfaces described in model code documentation (e.g., as found in README markdown documents). The UA team also continued support of the World Modelers ConceptAligner. The following report describes each of these components, the current state of development, and where appropriate, describes evaluation.

9.2. Text Extraction and Linking for Interface Identification

The main focus of work by UA team during the project was on adapting the AutoMATES text reading pipeline to support targeted reading of model documentation that can extract information directly relevant to the Jataware Dojo model registry interface. In September 2021, the UA team coordinated with Jataware and agreed to focus on the TWIST and Flee models for a case study while developing rules to target the metadata in Dojo. This work included targeting extractions from academic papers about the models (identified by the model authors) as well as developing new extraction capability for reading from README markdown files that are often provided as part of the model codebase release.

The UA team started development of extraction rules to target README markdown. Example markdown files from TWIST, Flee and several other models were used for rule development. These were manually annotated to identify explicit markup use conventions, such as using markdown emphasis to introduce terms and variables. Heuristic rules were developed to identify instances of shell-script or command-line execution calls along with identification of command-line arguments. Potential future development targets that were identified but not extracted are tables and mermaid diagrams.

The UA team combined the markdown extraction rules with other AutoMATES text extraction and developed an export facility to compile all extraction types into a format similar to the metadata format exported by the Dojo API. This combined extraction is exported as a JSON file. The results of performing extraction on the TWIST and Flee documentation are available here:

- TWIST: `https://drive.google.com/file/d/10aQJJDZfA2DEoQUM80Pm3DdPaXG2XFWS/view?usp=sharing`
- Flee: `https://drive.google.com/file/d/10cxf2mWX1eokMDMAwAAocYkNJRvgHjfk/view?usp=sharing`

The UA team performed a manual evaluation of the relevancy of the extractions relative to the existing Dojo metadata for the manually-registered TWIST and Flee models. The results of the assessment are included directly in the TWIST and Flee assembled mentions linked above, via the ”annotation” field:

- ”match”: 1 indicates a successful match to a dojo entry, 0 indicates not related to an existing dojo entry.
- ”acceptable”: 1 indicates the information is judged to be model-relevant (even if match = 0).
- ”dojo-entry”: when the extraction is judged to be a match, this field names the dojo entry that the extraction is relevant to.

The following summarizes the results of the assessment:

- TWIST:
  - The AutoMATES extraction pipeline identifies 220 extractions.
  - 31 of these were judged as direct matches to Dojo registration metadata entries.
  - An additional 130 AutoMATES extractions were judged as containing direct model-

relevant information that is not currently used in the Dojo registry.

- Flee:
  - A total of 454 extractions
  - 23 extractions match the Dojo registration metadata directly.
  - 335 model-relevant extractions not matching Dojo directly.

In both cases, the remaining extractions were judged to be false positives.

Overall, it appears that the extractions can provide domain experts with an informative, concise summary of the model documentation relevant for registration. It is worth noting that the criteria used for direct matching were strict; with human domain-expert reviewers, additional extractions will likely provide useful information for registration. The instructions for running the text extraction pipelines are available on the github wiki:

- https://github.com/ml4ai/automates/wiki/Text-Reading:-Usage

9.3. Additional Reading Pipeline Improvements

In support of the development of the rules for model interface extraction, the UA team also made a number of improvements to the text reading pipeline. These are summarized here:

- Added support for coreference resolution, which overal improves extraction fidelity.
- Extracting function descriptions from text. Function descriptions typically include assertions about the relationships (often causal) between variables (see last section for summary of progress on causal relation extraction). Rules were created to identify variables roles within function descriptions, including aggregating descriptions that span multiple sentences.
- Developed a set of extraction rules aimed at identifying when text is describing a ”model component”. Supported component types include: model name, description, model repository, list of files, execution command.
- Extended support for wikidata grounding. This included adding the ’subclassof’ relation and improved query speed with caching
- Improved integration with the University of Wisconsin COSMOS system. COSMOS is used by AutoMATES for input pdf processing. COSMOS splits text on pages into blocks, which often split sentences. The team implemented a general method to reconstruct split sentences and properly index extractions. This included automatically identifying when text is split across blocks, reconstructing split sentences, and extending the extraction metadata to support cross-block references and computing text mention span offsets.

9.4. ConceptAligner

The ConceptAligner provides a service for linking variables, at different levels of granularity, that are used within the World Modelers (WM) modeling ecosystem. In general, WM variables denote modeling concepts from two different levels of granularity: ”top-down” variables are generally derived from machine reading of domain literature, and tend to correspond to abstractions over physical processes (spatially and temporally), while ”bottom-up” variables are derived from implemented domain models, and include model inputs, parameters and outputs as well as other

indicators.

The ConceptAligner provides an API whereby users can query for top-down concepts that appear most closely related to bottom-up variables, and vice-versa. The tool consists of three components: a scraper, an indexer, and a searcher. The scraper walks the exposed Dojo metadata (and previously, the SuperMaaS metadata). The metadata fields are converted into a vector representation via an embedding model. Scraping is run as an offline process. After scraping, the vectors are efficiently indexed using a module for fast, approximate nearest neighbors search. Finally, the searcher provides an API to support queries for searching for similar concepts between top-down and bottom-up concepts/variables based on string descriptions (variable name or concept description). The ConceptAligner tool is hosted at the following url: https://linking.cs.arizona.edu/api

9.5. Efficiently Extracting Causal Influence Mentions from Text

Models are about causes, so an important capability for model extraction from text is the ability to identify expressions of causal precedence relations. In previous work, the UA team developed a deep-learning based framework for identifying causal precedence relations in text. The framework was developed using a relatively small annotated corpus of just under causal precedence relations within the biochemistry literature. The framework is given a segment of text that contains two events identified by the Reach machine reader as representing biochemical events, and the framework then predicts the type of relation between the events. The input includes the segments of text for each of the events as well as the text between the events.

The model achieves relatively strong performance, but this fall the UA team sought to address two challenges: (1) in order for the framework to be practically applied for processing large copora, it must be able to rapidly process text while not degrading in performance, and (2) the training corpus is relatively small, so we seek additional pre-training methods that can improve performance through domain adaptation.

State of the art language models are based on the very large pre-trained BERT model. BERT consists of 110 million parameters and inference time per average document is relatively high. Prior to the start of the WM SuperMaaS, the team had explored distilling a BERT model into a small LSTM model, hoping the distilled LSTM would have better performance than only fine-tuning an LSTM. However, the distillation results were not satisfactory.

This fall the UA team explored using MiniBERT, a version of BERT obtained by reducing the number of layers and the hidden layer size of BERT. The verison of MiniBERT used in the study has only 5.6 million parameters, addressing challenge (1). The team also explored methods for domain adaptation, to address challenge (2). As a strong baseline, the team use BioBERT, a version of BERT that is pre-trained on a large portion of the open access PubMed paper corpus (https://github.com/dmis-lab/biobert). The team pre-trained MiniBERT using the BookCorpus (https://arxiv.org/abs/1908.08962) and English Wikipedia corpora and also compared the adapted version of MiniBERT to a version that is further trained on the PubMed corpus using a masked language modeling task before fine-tuning on the causal relation detection task (BioMiniBERT).

Model	Dev F1 (std)	Test F1 (std)	# Params	# Param Emb	CPU time
BiLSTM-small	0.505 (0.001)	0.509 (0.13)	0.7M	0.2M	0.010
BiLSTM-large	0.512 (0.007)	0.539 (0.018)	5.5M	0.2M	0.025
MiniBERT	0.463 (0.002)	0.508 (0.023)	5.6M	3.9M	0.017
BioMiniBERT	0.506 (0.013)	0.547 (0.029)	5.6M	3.9M	0.017
BERT	0.529 (0.024)	0.541 (0.004)	110M	23M	0.13
BioBERT	0.559 (0.016)	0.554 (0.008)	110M	23M	0.13

Table 1: Performance, size and speed of different language models used in causal precedence identification framework.

1 summarizes the performance results for the different model variants that were explored. The first and second columns represents the mean (and standard deviation) F1 scores on the causal precedence development and test corpora, respectively. The third column represents the overall number of parameters in the model, while the fourth column represents the number of parameters in only the embedding model. The last column represents the average time the model uses to process each sequence (in seconds) on a CPU. The average sequence length was 32.17 tokens.

BERT and BioBERT have the strongest performance across all models. As can be seen in the Test F1, the domain-adapted BioBERT has a small but significant improvement in its ability to adapt to the causal precedence task compared to BERT. However, both models have 110 million parameters, about 20 times larger than MiniBERT and BiLSTM-large, and take a relatively large amount of time to process each sequence.

The BiLSTM still poses a strong baseline on this task. Increasing the BiLSTM from 0.7 million to 5.5 million parameters (via increasing the hidden layer size) significantly increases the test performance from 0.509 to 0.539. Surprisingly, MiniBERT (pretrained on the BookCorpus and English Wikipedia datasets with the masked language modeling task does not perform as well – even slightly worse on average compared to MiniBERT-small.

BioMiniBERT takes MiniBERT and then further pretrains using 10,000 papers from PubMed along with the masked language modeling task before fine-tuning on the causal relation dataset. Compared to MiniBERT's F1 of 0.508, BioMiniBERT's F1 is 0.547, showing that in-domain pretraining can significantly improve the model's performance, even improving over BERT, a model with well over an order of magnitude more parameters, and cutting processing time by an order of magnitude.

9.6. Recommendations for Future Work / Extensions

The results of the extraction analysis suggest that the framework is retrieving potentially useful information that can be used to pre-populate the Dojo metadata form used for registering models. The following are suggestions for future work and extensions to the approach:

- Create a Dojo endpoint for ingesting the AutoMATES extractions and automatically pre-populating the Dojo metadata fields.
- Investigate methods for scoring/ranking proposed metadata extractions. Scoring methods

should be assessed by their ability to identify relevant information.
- Perform an assessment on the reduction in effort made by humans doing manual registration.

10. Jataware

10.1. Project Objectives

Complex systems such as population refugee migration, conflict, and food security are difficult to model. Understanding them requires a combination of deep expertise in climate modeling, hydrology, social science, and other domains. These domain specific models are created and curated by siloed researchers who are not typically well-positioned to provide topical, on-demand analyses. The depth of knowledge contained in these models can be extremely useful for policy decision making, however analysts and policy makers must interface with domain modelers directly in order to leverage these powerful tools.

This project aimed to address this fundamental problem by challenging the notion that domain modelers must be the direct intermediaries for and interpreters of their models. Instead, this project's primary objective was the creation of an extensible modeling as a service platform. This platform would enable the registration and execution of arbitrary domain specific models to provide decision support for non-domain experts and policy makers. Such a system would normalize model outputs from heterogenous formats and schemas into a standardized, consumable one. The platform would offer a standardized set of metadata about each model, including descriptions about the model and its internal processes, as well as information about the model maintainer, its inputs, outputs, and its "tunable knobs" or parameters.

Jataware's primary responsibility on this project was to prototype such a model platform and implement means to scale it to numerous models. Jataware was tasked to produce code, workflows, and lessons learned for domain model registration and execution that could be shared with the broader modeling community. Additionally, Jataware was tasked with performing system integration between the models as a service platform and other software components, including datamarts and visualization engines.

10.2. Work Performed

In support of this effort, Jataware created Dojo. Dojo is a suite of software tools that allows domain experts to register their models using an intuitive web based terminal emulator. Additionally, Dojo provides a mechanism for analysts and domain experts to register and transform datasets for use in downstream modeling workflows. Dojo is comprised of four key components:

- A robust API for retrieving model metadata, submitting model execution runs, and fetching model execution results
- A data registration human machine interface (HMI) for registering indicator datasets such as the World Development Indicators from the World Bank
- A model registration human machine interface (HMI) and workbench to enable domain modelers to containerize their models and annotate them for future model executions
- A model execution engine built on top of Apache Airflow

10.3. Modeling API

This API was built using relevant paradigms from prior similar efforts such as the ISI MINT project and the Galois SuperMaaS system, leveraging pre-existing agreed upon metadata schemas where applicable. This API includes a set of endpoints designed to enable the fetching of model metadata including:

- **Model description:** a rich description of the model, its functionality, and its limitations
- **Maintainer information:** contact information for the creator of the model
- **Parameters:** details and descriptions about each "tunable knob" that the model can accept. For example, a crop model may allow the user to model crop production under various fertilizer regimes.
- **Model outputs:** descriptions and units for each output variable for a given model

This information was used by the visualization team to automatically generate an analyst-oriented HMI for each model. Additionally, this API enables end users and 3rd party systems such as Uncharted's Causemos tool to kick off model runs and fetch model results. The API enables the user to select a set of parameters for a given model run. The user may poll the API to determine when the model execution has completed; once completed, they may fetch results from Amazon Web Services (AWS) S3 cloud storage. Additionally, end users may fetch run logs from each model execution for debugging purposes.

10.4. Data Registration HMI

Under this subcontract, Jataware also created a data registration HMI. This HMI enables non-technical, data-literate end users to convert NetCDF, Geotiff, and Excel files into normalized, geocoded Parquet. Parquet is a compression and performance oriented columnar format. Data will ultimately have a fixed set of columns plus arbitrary qualifier columns. The fixed columns are `[timestamp, country, admin1, admin2, admin3, lat, lng, feature, value]`. Here `qualifier_1` is the name of the qualifier which qualifies `feature2`. Note that the fixed columns are nullable, but data that does not have at least some notion of time or place is not particularly useful.

timestamp	country	admin1	admin2	admin3	lat	lng	feature	value	qualifier_1
1546318800	Ethiopia	Afar	Afar Zone 3	Gewane	10.16807	40.64634	feature1	1	
1561953600	Ethiopia	Afar	Afar Zone 3	Gewane	10.16807	40.64634	feature1	2	
1546318800	Ethiopia	Afar	Afar Zone 3	Gewane	10.16807	40.64634	feature2	100	maize
1561953600	Ethiopia	Afar	Afar Zone 3	Gewane	10.16807	40.64634	feature2	90	maize

Table 2: Example normalized dataset

A key feature of this HMI is its ability to take arbitrary structured datasets and convert them into this highly consumable format. Additionally, latitudes and longitudes are reverse geocoded into their respective place names from country down to admin level 3. Epoch timestamps are generated

from arbitrary time formats based on the user's annotation of their dataset. This HMI supports dates that are split into multiple columns. For example: a "year", "month" and "day" column will be combined into a single epoch timestamp.

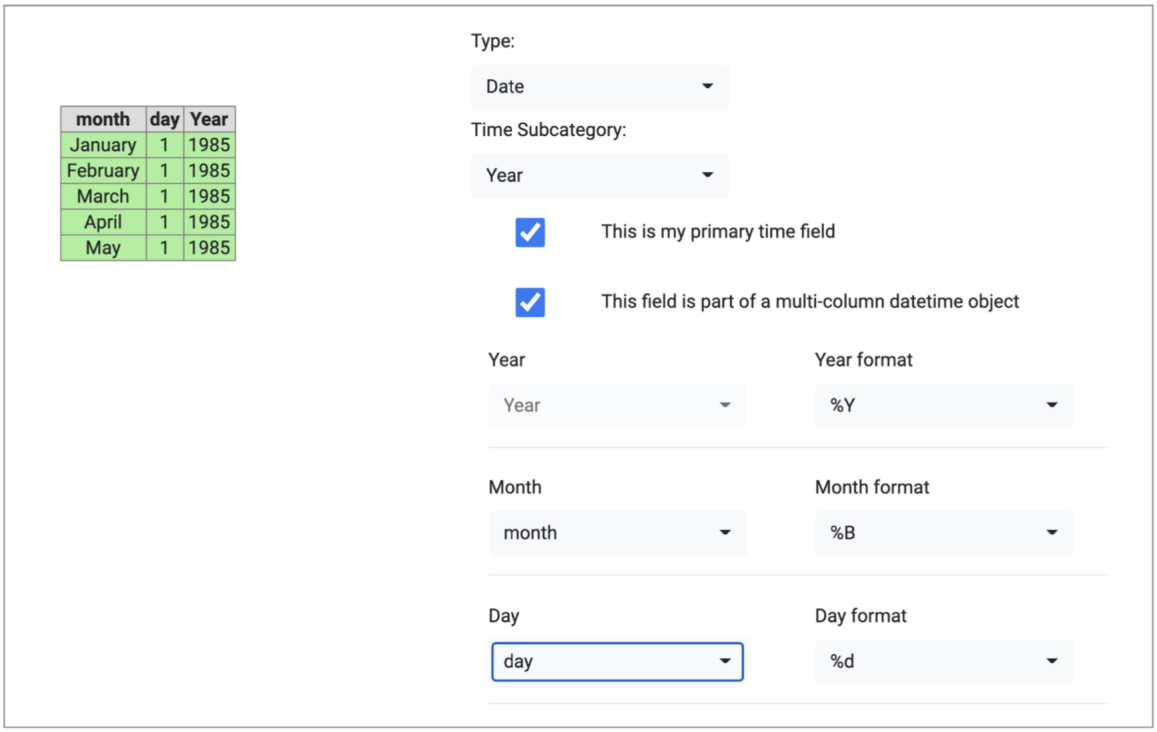


Figure 24: data annotation HMI

10.5. Model Registration HMI

Jataware created a model registration HMI whose purpose is to enable domain modelers to containerize their models in an environment they would be comfortable using: a web-based terminal emulator. Modelers provide basic metadata about their model via a series of forms and then install their model and its dependencies into an Ubuntu-based containerization environment. Once the modeler has completed the installation of their model they then execute the model to test the accuracy of their installation. As the user executes the model, they are given the opportunity to annotate the command line interface (CLI) directive used to execute the model.

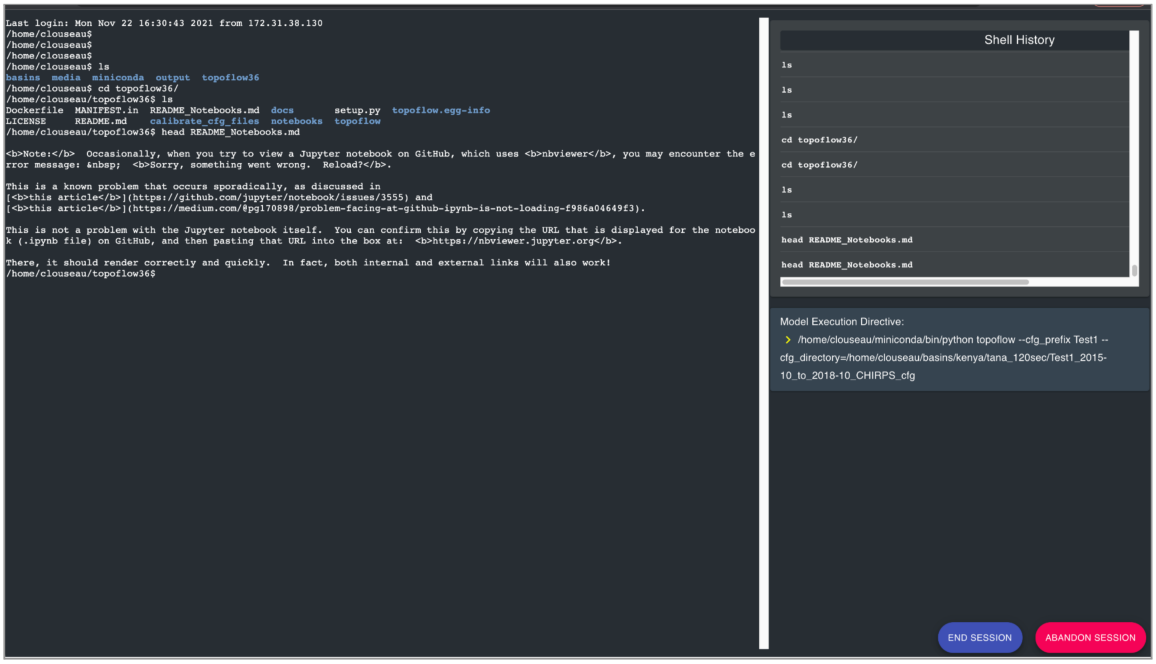


Figure 25: Web-based terminal editor

Annotation of the model directive is a key step to being able to successfully run a model without the modeler in the loop. Dojo contains a unique HMI component designed to allow the domain modeler to "mark up" their execution directive and instruct the system in how to pass the model its parameters.

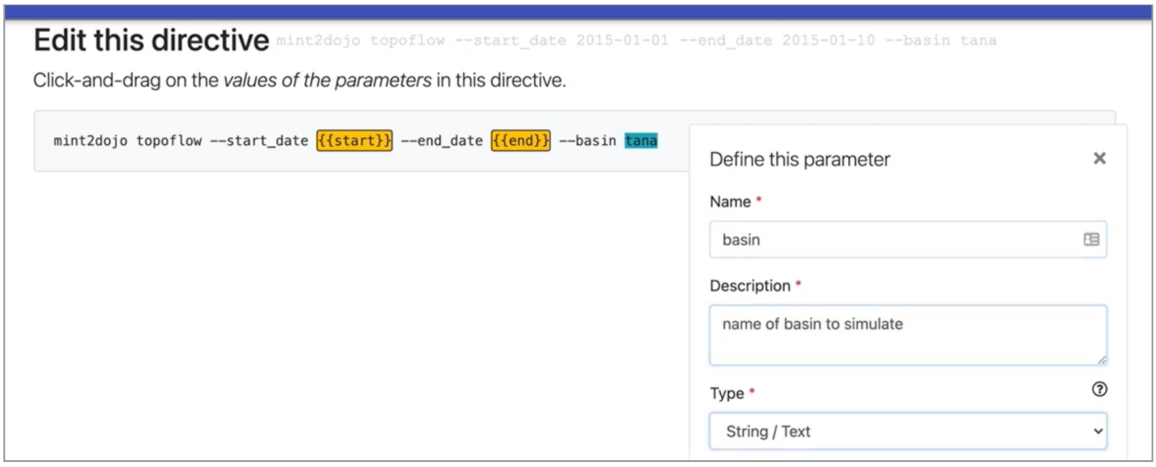


Figure 26: Model directive annotation interface

Modelers may annotate configuration and settings files via the same interface. This is accomplished by a set of specific commands embedded into the terminal emulator CLI. These are:

command	description	example usage
edit	opens a simple text editor	edit `edit myfile.txt`
config	opens the configuration file annotation tool	`config parameters.json`
accessory	tags an output accessory file such as an image or video; caption is optional	`accessory floods.mp4 "Video of flood activity in selected basin."`
tag	tags an output file and opens the output file annotation tool	**`tag results.csv`**

Table 3: Terminal emulator commands

Through the "tag" command, modelers are able to annotate model outputs so that they are normalized to the schema described in the **Data Registration HMI** section above. Through the "accessory" command, modelers indicate various accessory or media files that are associated with a model run.

Once the modeler has completed installation, execution, and annotation of their model they are able to publish it. This includes finalization of the metadata stored to the **Modeling API** and the publication of the final model Docker container to Dockerhub.

10.6. Model Execution Engine

Dojo relies on a model execution engine built on top of Apache Airflow. Jataware strategically decided to leverage and expand on this excellent open source workflow scheduler instead of building their own. This reduced effort spent on "solved" technical issues and enabled Jataware to focus on how to correctly parameterize models on an ad hoc basis and how to efficiently store and communicate model results.

Each model run is represented as a directed acyclic graph (DAG) that consists of several graph nodes.

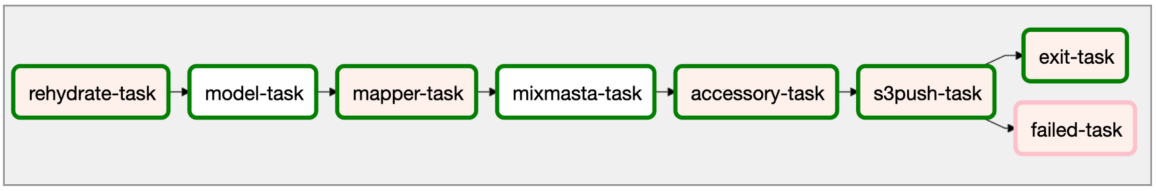


Figure 27: Model execution DAG

These task nodes include:

1. Rehydration of any model configuration files based on end-user parameter selection.
2. Model container execution based on user selected parameters
3. Fetching the model output annotation mapping file
4. Normalizing model outputs to the previously described schema
5. Storage of model accessories
6. Storing model results to AWS S3
7. Updating run status in the **Modeling API** and notification to Causemos

10.7. Results Obtained

The results of this project include a full-featured end-to-end modeling system. Domain modelers have provided positive feedback on this system, indicating its promise for future modeling efforts. Currently there are dozens of models registered to the system, spanning domains as diverse as hydrology, climate, agriculture, conflict, logistics, and economics

10.8. Estimates of technical feasibility

The primary hurdle to transitioning this technology to a production setting is twofold: the identification of appropriate transition partners and the ability to motivate domain modelers to continue to use the system. Potential transition partners for the technology include universities, NGOs, and research consortiums. However, since many models require substantial computing resources, it may be challenging to find transition partners that are able to provide this modeling system at scale. As a mitigation step, models that must be run in high performance computing centers should be registered to this system as data services where cached results are fetched directly from the modelers via an API or FTP since it is not feasible to expect Dojo to be deployed in a high performance computing environment in the foreseeable future.

To motivate domain modelers to continue developing and registering models to Dojo after this project ends there must be some effort to market the system to them in terms of its ability to facilitate reproducible research but also as a means to expand the reach of their analyses and models beyond academia and into the realm of policy decision making. However, modelers are often reluctant to relinquish strict control over the narratives surrounding analyses of their model outputs. Therefore, work must be done to find means to "reach back" to modelers for periodic results validation.

11. Uncharted

The Causemos platform is being developed using a user-centric approach under the World Modelers program for rapidly developing causal simulation models of real-world complex systems for evidence-based planning. Additions to the Causemos analytical platform described in this report broaden the ability of end users to leverage knowledge encapsulated in sophisticated domain models and structured datasets.

11.1. Workflows

Assemble and quantify a qualitative model. Analysts use Causemos to capture their mental model of the system relevant to their analysis (e.g. food system determining food security) while being supported by existing assembled knowledge. The resulting causal analysis graph (CAG) is semi-automatically quantified and then used to test various scenarios (see Figure 1). During the quantification step, the system provides a heads-start by mapping datacubes to concepts in the CAG leveraging the algorithm developed by the University of Arizona.

The analyst reviews the default mappings and when not satisfied with a default quantification, uses the data explorer to find a different datacube.

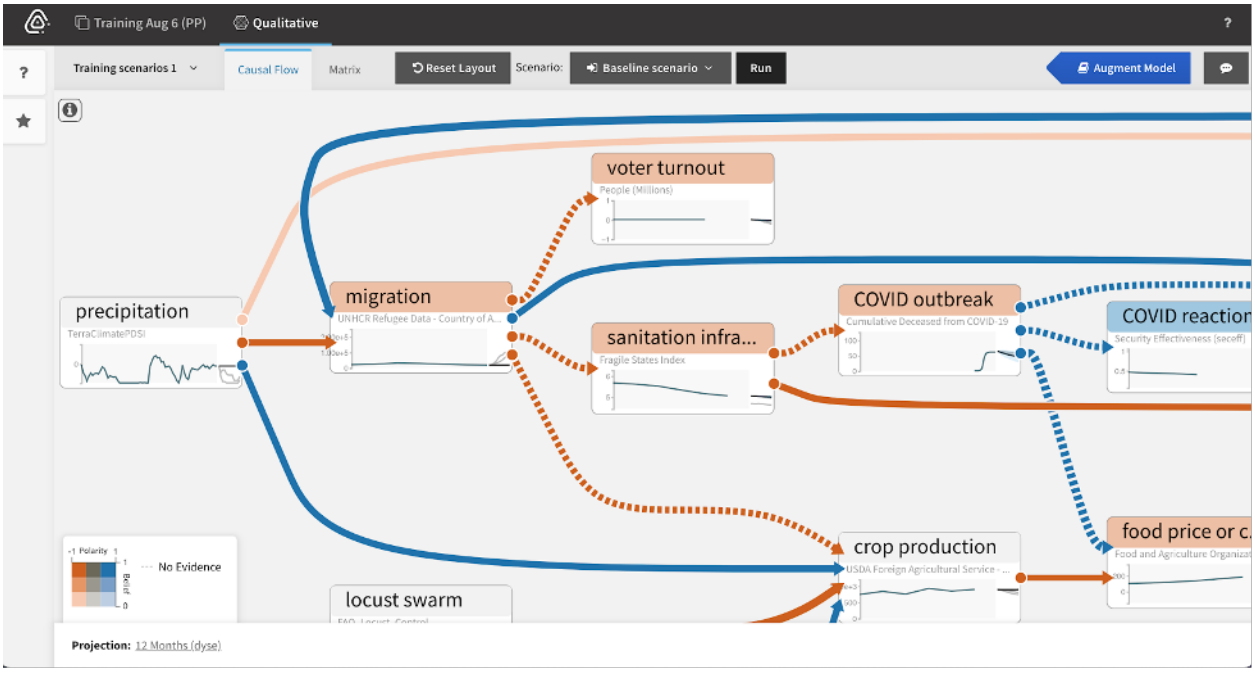


Figure 28: Quantified qualitative model. Datacubes used to quantify the concepts can be seen in the nodes: the datacube name is displayed under the concept name and the data is displayed in the grey/historical section of the chart (the white section is for projections).

Discover relevant data. The analyst can discover relevant data and models using multi-faceted search in the Causemos data explorer (see 29). Datacubes are searchable based on their types, regional and temporal scope, variable names, source, tags and more. Once a datacube is found, the analyst can quickly understand, examine, and judge its relevance to the situation at hand by leveraging the rich metadata.

Previously in the program, the focus was on building out tooling to automate the extraction of metadata from various sources. This approach proved to be inadequate and resulted in poor quality of extracted metadata. In response, a human-in-the-loop strategy proved to provide much better results. Currently, the metadata that is collected about models and indicators comes from a mix of data extraction and manually entered descriptions. For example, when a model or dataset is registered, the user is asked to write descriptions for the model as a whole, each input parameter, as well as each output. These descriptions serve two purposes. First, the rich descriptions can be analyzed by an algorithm and mapped to concepts in an ontology, these concepts would then be used when matching data with CAG-nodes elsewhere in Causemos. Secondly, analysts that use the data, can read these descriptions to learn about the methodology, data collection process, how the simulation is created, how it can be controlled via various input parameters, and what the outputs represent.

Aside from the manual entry of descriptions, the data is also analysed directly to extract out relevant metadata that's helpful when browsing the large number of indicators and models that are available. This includes the temporal span and resolution of the data, the geospatial span including which geopolitical regions are available, as well as, whether the data is fine grained, gridded data

or simply aggregated to said geopolitical boundaries.

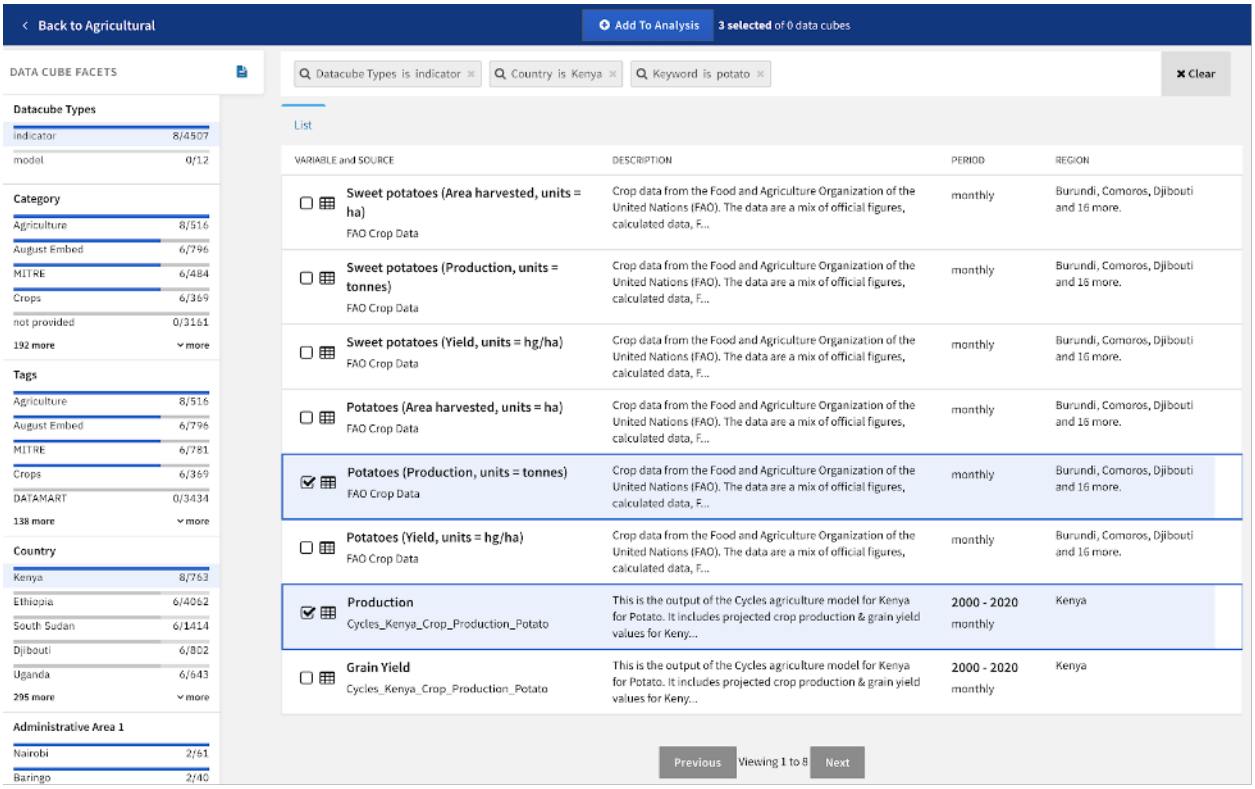


Figure 29: Data explorer with multi-faceted search. In this example, the analyst searched for structured data (datacube type: indicator) about potato crop (keyword: potato) in Kenya (Country:Kenya). The Potatoes Production datacube selected can be seen in 30.

Comparative analysis with quantitative data. More insights can be extracted from datacubes with the comparative analysis capabilities. The analyst can look for historical and regional analogs by splitting the data by year or by region (see 30). However, feedback from analysts stressed the need to provide more ways to break down the data.

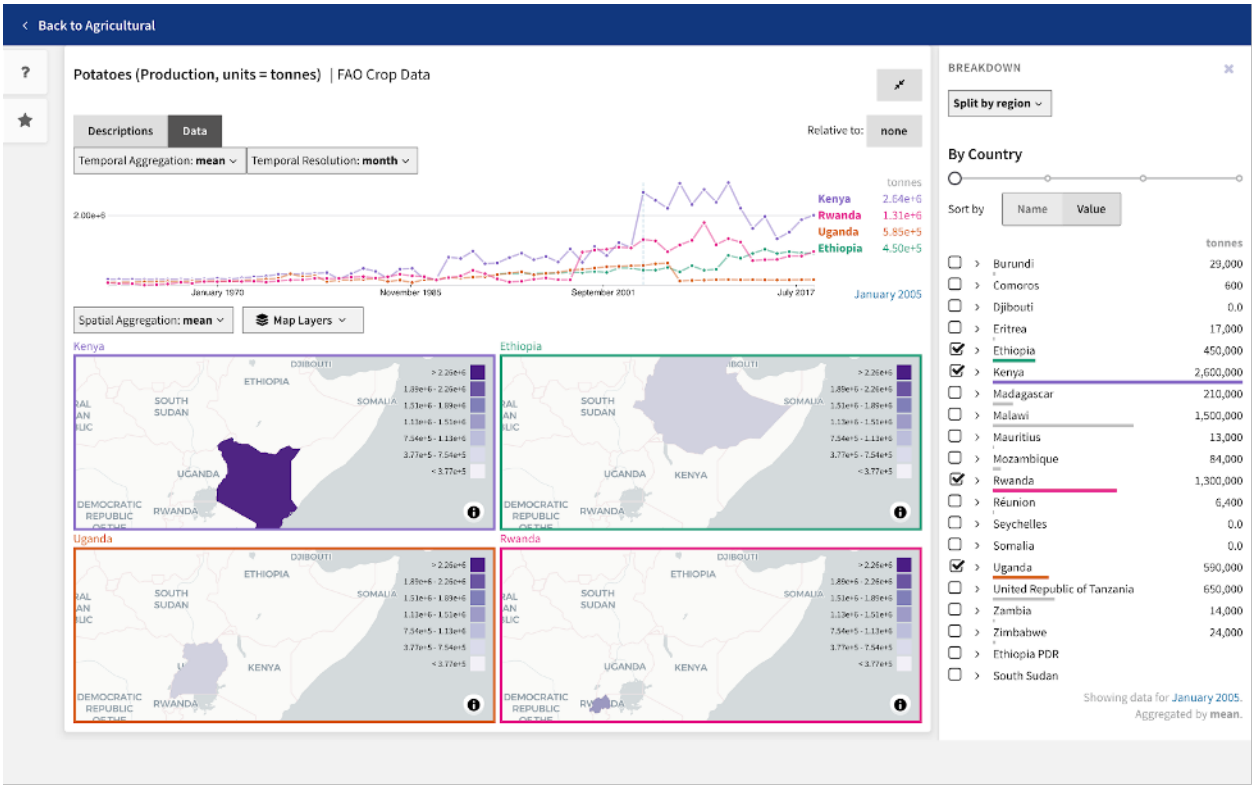


Figure 30: Comparative analysis with a datacube. The screenshot shows the 'split by region' feature: the data for the four regions selected in the side panel are compared temporally in the timeline and geographically with small multiple maps.

Discover, understand and analyze scenarios with domain models. The data explorer also helps analysts to discover domain models available in the system. Filtering by datacube type quickly reveals all domain models and then facets can be used to refine results further. Understanding a model involves inspecting the metadata and analyzing existing model scenarios. The Descriptions tab is shown by default when opening a model datacube to provide the definitions needed to make sense of the data. Switching to the Data tab shows the data visualizations for the default scenario. Causemos facilitates the analysis of multiple scenarios by providing a parallel coordinate plot where each line represents a scenario (i.e. model run). Selected scenarios can be compared using the same comparative analysis capabilities described above for indicator datacubes. The 'relative to' mode allows the comparison of the scenarios relative to a reference (e.g. the baseline scenario).

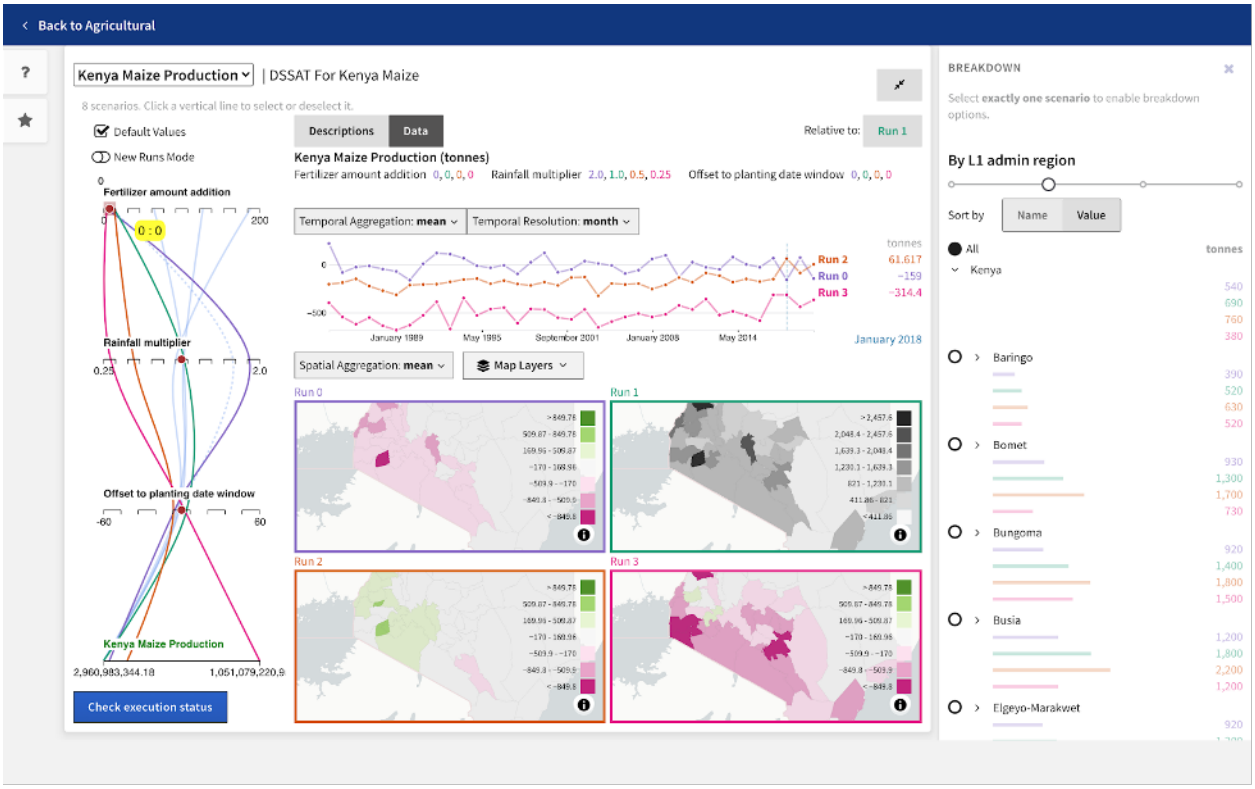


Figure 31: Comparing scenarios produced by a domain model in Causemos. Using the 'Relative to' feature, the visualizations show the difference from the baseline scenario, which is the green run, shown in grey on map. We can see that crop production in Kenya in 2018 would have been lower for much dryer and much wetter scenarios, but higher in the western corner of Kenya with conditions slightly dryer than usual.

Run new scenarios. If existing scenarios do not satisfy analysts' needs, new scenarios can be specified, and execution launched within Causemos (see 32).

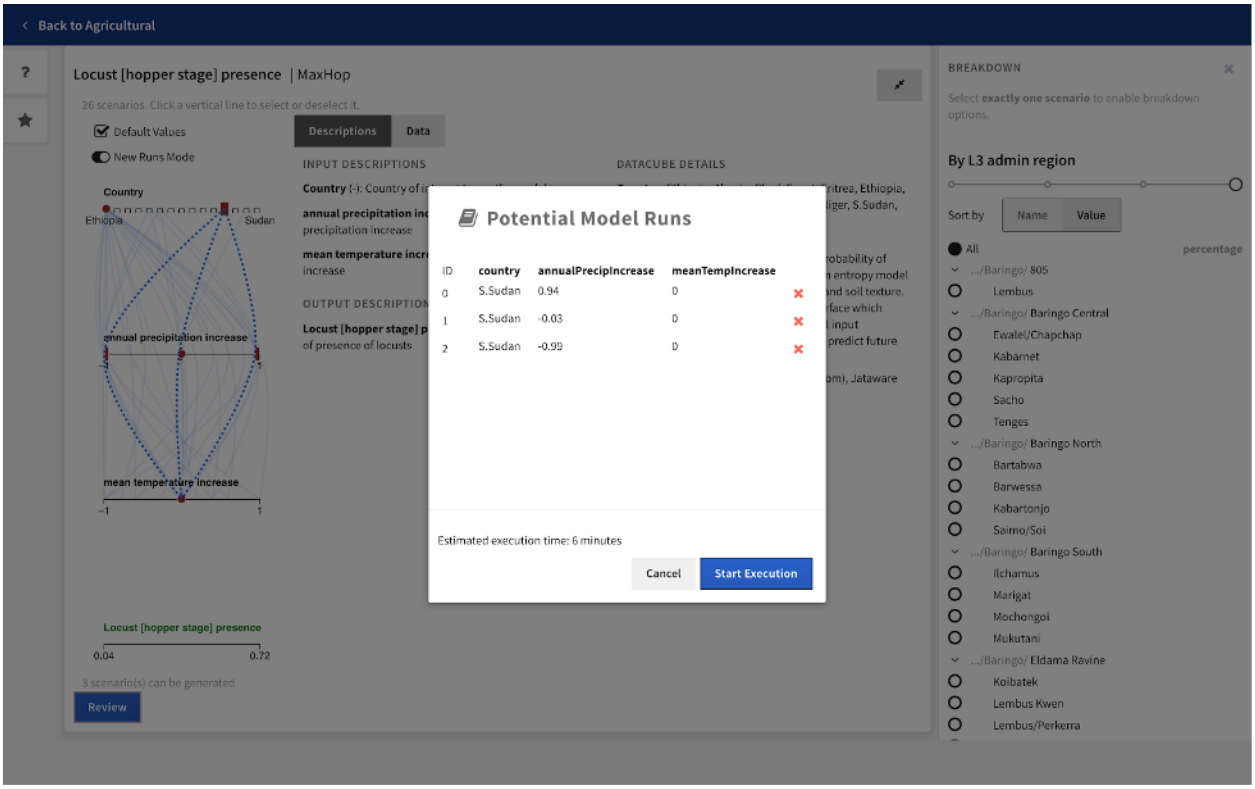


Figure 32: The analyst can specify and run new scenarios using the parallel coordinate plot also used for scenario exploration

Capture insights. The analyst can capture findings and insights throughout the analysis. For example, the analyst can save and annotate a particular state of a model analysis visualization where an insight is visible. Such annotations facilitate provenance tracking.

11.2. Embed experiments

Starting in June 2021, the World Modelers program started conducting evaluations where the new tools are embedded in the analysts' environment and workflows for a full month. In these experiments, the participants dedicate most of their time conducting a realistic analytical tasking and the resulting recommendations are presented to decision-makers at the end of the four weeks. The June 2021 embed was intended to be a dry run for the August 2021 embed. In June, two analysts from MITRE worked on assessing the stability/fragility in Djibouti around election time. The dry run experiment generated useful feedback and recommendations to improve Causemos in general. However, since the new features to bring in domain models and structured data with richer metadata were still in active development, they could not be used in the normal workflow during this dry run. The feedback reiterated the need for the new features, in particular, better data coverage, interpretability and better default mapping of data to concepts.

The August embed involved two analysis teams, one in East Africa working on nutrition security in the region and one North American think tank aiming to produce recommendations to mitigate the risk of violence during the next elections in Kenya in 2022. The initial version of model

publishing capabilities was made available. We conducted training sessions with domain modelers and analysts. Issues and feedback were addressed as much as possible throughout the experiment. Key takeaways included the need to improve the model registration and model publishing flow (including better management of model/dataset/run, better context for specific annotations), to design solutions to support special type of input parameters like geographical and freeform input, to make run exploration more scalable, and to provide ways to visualize more than one output variable at once in order to facilitate insights.

11.3. Technical architecture and data flows

Causemos pulls model metadata from a public registry. Once in Causemos, model scenarios can be created in the analytics platform. When a scenario is launched, it results in another entry in the list of outputs in the model's corresponding datacube. The scenario definitions are stored in ElasticSearch for search and traceability, and then sent upstream with a REST-API to invoke model execution. Model executions are triggered by sending scenario specifications to the model-controller. Upon completion the output and corresponding run metadata are then stashed into S3 in Parquet format, the controller then sends a notification to Causemos to indicate a change in the execution status.

On receiving the status notification, Causemos constructs its own visualization processing plan from the scenario and model metadata specification, figuring out how to summarize across temporal and spatial dimensions, for example if longitude and latitude are provided Causemos will create a tiling-pipeline as part of the execution plan. Once the plan has been constructed it is forwarded to a scheduling manager, and subsequently sent into a Prefect/Dask cluster to run the execution plan in parallel. Statuses are tracked throughout the execution to ensure awareness. On completion the datacube is marked as "available" and results are written into S3, these can include Gridded maps from tiling pipeline, regional aggregations for geopolitical maps, monthly and yearly timeseries across regional dimensions, and other calculations based on model metadata.

A data microservice is in place to retrieve data from S3 and provides additional normalizations and transformations for Causemos visualization needs.

Indicator data goes through a similar workflow but without the manual execution step in Causemos. When indicator data is added to the system, the model controller sends a notification to Causemos to indicate there is new indicator data and metadata to be processed. On receiving the notification, the data is processed in the same pipeline and the results are written to S3.

Breaking down data. It has always been apparent that the data being added into Causemos would always be more than a simple time series. Geospatial data was immediately identified as highly important to visualize and has always played a key role in how we process and display the data. But, while geospatial data is common across all models and datasets being used in Causemos, there are also individual dimensions that also need to be handled. For instance, a crop production model would provide data on different types of crops, while a refugee displacement model may list the start and end locations, or the cause of displacement. Similarly, event data is extremely rich with dozens of these types of categories, and there are endless ways of breaking down the data.

When initially planning model execution, a concept of "drilldown" parameters was proposed. These "drilldown" parameters would be defined by a model, such that multiple runs across the allowable values could be conducted, and the data aggregated together. There were several issues identified with this approach. Model runtimes and processing times were the first concerns. Executing a model dozens of times to exhaust the parameter values, and then stitching the data together would prove to be very costly, both in terms of processing time and the size of the data. Secondly, if models were to define more than one "drilldown" parameter, the number of combinations and model runs required would grow exponentially. Lastly, datasets, unlike models, already have all the data available and combined into one. The issue there wouldn't be about requesting various types of data, but instead, annotating it in some categorical manner. A task that would ultimately need to be tackled for model run output as well, if the data from multiple runs were to be combined.

A decision was made, to focus on this last problem, it being common to datasets as well as model output. Instead, the responsibility of providing data across multiple categorical values would be at the model owner's discretion.

The format that was chosen to represent this data would allow each data point to be marked with any number of categorical values, or "qualifier values". The values would then be grouped into logical headings. Data with geopolitical locations already followed a similar schema, where each data point was assigned a location under the heading of "country" or "state". The headings could be concepts such as "event type" or "crop type" with values such as "riot" or "barley", respectively. When processing the raw data, the data pipeline could focus on one, some, or even none of these headings and aggregate the data accordingly. The resulting data could then be broken down to answer questions such as "How many riots have there been in the past 10 years?" or "How much maize is produced as a percentage of total crop production in Ethiopia?"

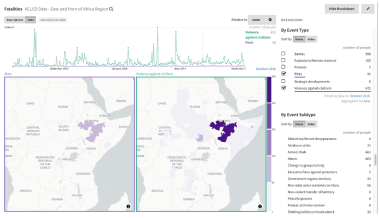


Figure 33: A breakdown of data from ACLED displaying the number of fatalities due to Riots vs Violence against civilians in October 2016

Data pre-computation. When dealing with data, a question arises on whether it is best to pre-compute the data and prepare it for visualization ahead of time or leave it in a raw format to allow more flexible, analyst driven, filtering and aggregation. When the data is large, the only hope is to process the data ahead of time to break it up into more manageable chunks. However, when dealing with data such as the type described above, the number of different ways to break down the data can quickly grow out of control. It seems then, that in these cases, it would be best to filter and aggregate the data at will, to extract the specific subset that we are interested in. The challenges naturally arise when the data gets too large to process outside of large multi-node distributed computing platforms. Since supporting large amounts of data was seen as a priority, the

effort was put into identifying the key combinations of qualifier headings for which data should be precomputed.

As mentioned previously, geospatial information has always been seen as a cornerstone of the analysis that is performed with data in Causemos. As such, being able to break down data by a geopolitical region in addition to a qualifier heading naturally became the focus. The pipeline would group the data by each region and qualifier value combination to produce more granular aggregate values than if it simply grouped by region.

Our future plans in this area include investigating data storage and query systems that would allow us to store the large amounts of annotated numerical data while also allowing us to quickly extract subsets of data. If the resulting subset of data was small enough, then the aggregation could quickly be done at runtime without any need for pre-computed data.

Pipeline robustness. During the June and August embed experiments it became apparent that our data ingestion pipeline needed to be more robust. Most importantly, the scheduler needed to be improved to allow the use of a second execution cluster, and the in-memory queue replaced with a persistent one. Aside from these new requirements, there also needed to be more automation, especially around error handling.

As part of this effort, Causemos would start keeping track of the work being done in the pipeline. While the pipeline manager would internally keep track of detailed job status and progress, this would now be propagated to the rest of Causemos. With this connection in place, Causemos can now receive processing time information, and any errors that arose. The execution times will be leveraged in Causemos to provide more accurate model runtime estimates. Retrying failed jobs with adjustments to the execution plan would also become largely automated, and future work would entail developing logic that would make these incremental changes to the execution plan.

An execution summary is now also provided to Causemos that provides information about the data that was processed. Among others, this includes: a list of optional execution steps that the pipeline took, the number of data points in the time series, the size of the data, and the geospatial and temporal coverage of the data. Some of this data is collected from the model owners during the registration process, however, it was discovered that it may not always be accurate, depending on the parameters of the model run, or other factors related to the model.

Model registration and publication. When developing the model registration and publication workflow, the process was divided such that a model would first be registered with tooling provided by Jataware, following which, the model visualization could be tweaked and published in Causemos. Various metadata about the model would be captured from the model owner and then sent to Causemos once registered. While registering the model, the user would be asked to specify information about the model's input parameters, output data, and any qualifier information. The parameter metadata would then be used in Causemos to construct the interface for setting these parameter values. The model owner could decide whether each parameter had a fixed list of choices, was a continuous numerical value within some range, represented a time range, or allowed any country in the world.

The way that this process was designed however, proved to cause much user confusion and mistakes. Model owners who were registering models for the first time would be asked to make selections about the types of parameters without knowing exactly what they were used for and what each option meant. This resulted in parameters being unintentionally marked as having an explicit list of choices with only one option, or a field where the analyst was asked to type in the name of a country that had to exactly match the name of a country. Once the selections were made and the model registered, the model owner would be stuck with the undesirable interface in Causemos and would need to go through the whole registration process over again to fix their error.

To rectify these issues, a redesign of the whole registration and publication process was needed. Rather than having two separate steps, one outside of Causemos, there should be a more fluid experience where the user can seamlessly go between the two systems to get immediate visual feedback on what actions their selections are having. This way, once a model is registered, the user can be sure that everything is as intended. Currently, this redesign has only been planned out, and the work on it has not been started yet. Aside from this larger redesign, the parameter input interfaces have also been improved to better suit the sort of data that models would require. One example of this is the ability to have the user specify a geographical bound box on a map and have the coordinates sent to the model as input (see Figure 34).

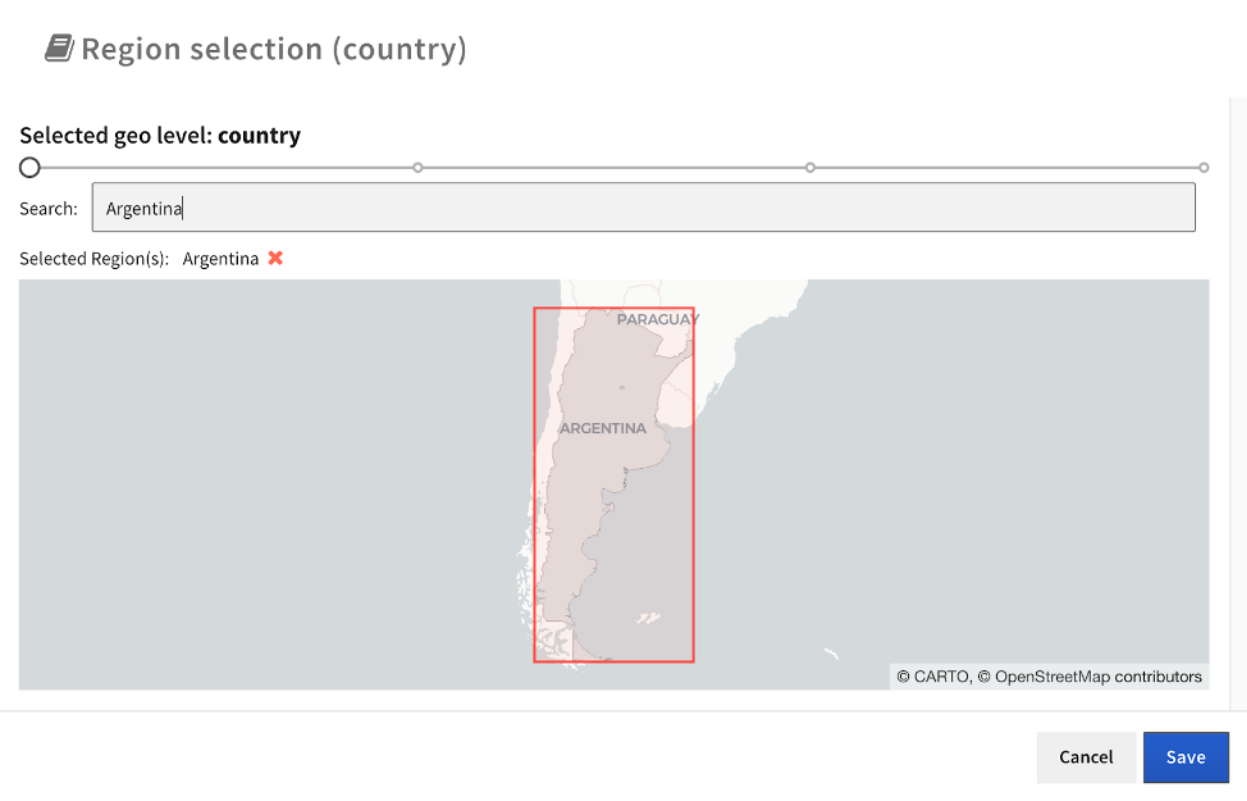


Figure 34: An analyst can use a map to search for a country and have the surrounding bounding box coordinates be sent to a model as input.

11.4. Future Work

A major area of focus for the future will be working with qualifier data. As mentioned previously, there are several limitations currently since the visualized data has to be precomputed. When there are many different qualifiers the number of combinations that have to be precomputed is simply too large. However, if the raw data was stored in such a manner that would allow us to quickly filter it, we could produce data for any sort of query specified by the analyst. We have concerns about storing the data in a raw format since some datasets and model run outputs push sizes of over 1GB. As such, we will first focus on smaller sets of data that could be stored and searched quickly without having to worry about these scaling issues.

Other plans for qualifiers include using the concept for tracking additional numerical data that could be leveraged in the data pipeline or simply used as additional visualizations. If the data in the qualifier data columns were to contain "weight", we could implement weighted aggregations which would give modelers greater control over how their data is processed. If the columns contained uncertainty measures such as a coefficient of variation or the min and max values for each data point, those data points could be aggregated together in parallel with the data to provide an overall uncertainty measure. Currently, there can be a false sense of precision with data produced from models since any uncertainty is not communicated to the user.

Another area that we need to address is the model registration and publishing process. This is a flow that is currently poorly communicated to the user and very error prone. We need to redesign the registration and publication process for a more fluid experience where the model owners receive immediate feedback about the selections that they have made. When a mistake is made, the user should be able to immediately notice that and go back in the process to adjust their selections so that everything is as intended. This process could take a considerable effort as some technical assumptions will need to be re-examined which could impact core systems.

Finally, there will be a continuous effort put into further automation of the data ingestion pipeline. Jobs that fail could be incrementally adjusted to remove failing subcomponents for the execution plan. With logic like this in place, a job that would have otherwise outright failed could now partially succeed which could produce sufficient and possible even desired results. In these cases, a currently manual process would be largely automated. In other cases where there was an actual error that should be addressed, these would still be tracked and evaluated manually as before.

12. Future Recommendations

Research and development of the methodologies, techniques, and technologies required for the SuperMaaS framework necessarily relies on the existance of suitable domain modelers with their own models to continually test and refine the framework, and to build a cohesive ecosystem for its use by practitioners. The cooperation and collaboration of other performers in the ecosystem of data- and model-driven analysis and response is not a given, and during this and related research, only a small part of the potential domain space has been covered. To ensure that SuperMaaS can continue to grow it is essential to build an ecosystem and community of practice committed to producing:

1. Documentation and specifications of APIs to interact with framework and ecosystem, even as model standards change.
2. Access to pre- and post-registered model artifacts for reproducability, along with anonymized logs of successful model registration to serve as templates and workflows. A centralized "github for models" would be extremely useful.
3. Examples of expert modeler and analyst workflows in SuperMaaS to reproduce post-registration workflows and patterns.

12.1. Surrogate Modeling

Often times, for both single- and multi-domain modeling, directly solving these models proves too computationally complex. We hope to address this with future versions of SuperMaaS through the use of a variety of surrogate modeling methods, which can then be parameterized and instantiated given appropriate training data. Surrogate modeling is an engineering method often employed when the mechanistic model describing the process is too complex to solve in the available time, or cannot be traditionally constructed. Surrogate models form an approximate metamodel, or emulator, of the process of interest mimicking the behavior of the process (or a mechanistic model for the process) as closely as possible while also providing computational efficiency during evaluation. Surrogate models are constructed using data-driven, bottom up approaches, specific to each surrogate model type. While the internals of a surrogate model can accurately mimic the input/output behavior of the system in question. They can also provide explainability through the evaluation of feature importance, and input/output causality relationships.

Koopman operator theory shows that the evolution of any set of observables in a dynamical system can be expressed through the action of an infinite dimensional linear operator known as the Koopman operator. This operator forms a canonical representation of any dynamical system and, in principle, can allow both the efficient solution of otherwise complex non-linear systems and the application of linear analysis methods on non-linear systems. Prior work has shown that when properly utilized, Koopman operators can even infer properties of dynamical systems that are either partially or completely unknown, or that are simply too complex to express using standard methods of analysis.