Composable probabilistic models can lower barriers to rigorous infectious disease modelling

Sam Abbott[†,1,*] Samuel P. C. Brand[†] Hong Ge[3] Kaitlyn E. Johnson[1]
Simon D. W. Frost[1,5] Anne Cori[4] Sebastian Funk[1]

2025-12-19

Abstract

Recent outbreaks of Ebola, COVID-19 and mpox, and routine surveillance of endemic pathogens such as influenza, have demonstrated the value of modelling for synthesising data to inform decision making. Effective models require integration of expert domain knowledge from multiple domains and outputs to be timely enough to inform policy yet current modelling approaches create barriers to meeting these goals. Methods used to synthesise available data broadly fall into approaches that chain separate models together, offering flexibility but losing information and potentially introducing bias, or rigorous joint models that are often monolithic and difficult to adapt. These barriers have prevented advances across multiple settings where models could have provided actionable insights. Composable models where components can be reused across different contexts and combined in various configurations whilst maintaining statistical rigour could address these limitations. We outline proposed requirements for a composable infectious disease modelling framework and present a proof of concept that addresses these requirements through composable epidemiological components built on Julia's type system and `Turing.jl`. We demonstrate a prototype R interface showing how such frameworks can bridge software ecosystems. Three case studies show how latent process components can be composed with epidemiological models to estimate time-varying reproduction numbers. The first replicates a COVID-19 analysis for South Korea using a renewal process. The second extends these components with reporting delays and day-of-week effects to replicate EpiNow2, a real-time nowcasting tool. The third replicates an ordinary differential equation model analysis of influenza outbreak data. We then discuss strengths, limitations, and alternative approaches. Our approach demonstrates promise for enabling interdisciplinary collaboration by lowering technical barriers for domain experts to contribute directly to model development. Future work is needed to solve remaining composability challenges, explore other options, expand the component library, and explore opportunities for large language model assisted model construction.

[†] These authors contributed equally to this work.

[1] Centre for Mathematical Modelling of Infectious Diseases, London School of Hygiene & Tropical Medicine, United Kingdom
[2] Center for Forecasting and Outbreak Analysis; Centers for Disease Control, United States of America
[3] University of Cambridge, United Kingdom
[4] MRC Centre for Global Infectious Disease Analysis, School of Public Health, Imperial College London, United Kingdom
[5] Microsoft Health Futures, Microsoft Research, United States of America

[*] Correspondence: Sam Abbott <sam.abbott@lshtm.ac.uk>

Author Summary

Mathematical models help synthesise data to inform infectious disease policy for both ongoing monitoring and outbreak response. For evidence to be useful it must be timely and rigorous, yet current modelling approaches struggle to be both. During COVID-19, models integrating multiple data sources informed policy decisions, yet were not adapted for the 2022 mpox outbreak where different expertise was needed. Other novel data sources have been shown to provide useful information but have not been integrated into most models used to inform policy. Combining results from multiple models can improve reliability, but without shared components it is unclear whether differences reflect scientific choices or technical details.

Composable models, where components can be reused across contexts and combined whilst maintaining reliability, could address these issues. When components have common formats, specialists can contribute their knowledge without understanding entire models. This allows different data sources to be analysed together reliably, and for specialist knowledge to be included, whilst remaining flexible.

We propose requirements for composable infectious disease modelling based on these challenges. We demonstrate feasibility through a working example, recreating three published analyses using shared components. Such systems could provide the rigorous evidence needed to inform policy in a timely manner during both ongoing monitoring and outbreak response.

1 Introduction

Modelling infectious disease dynamics can be a valuable tool for synthesising information and developing quantitative evidence for understanding and control [1]. The utility of such models lies in their ability to combine expert knowledge about transmission with the often complex ways in which infections are ultimately observed. During the COVID-19 pandemic, models integrating case counts, prevalence surveys, severity data, and hospital bed occupancy provided evidence that supported policy decisions [2–4], yet these models were not used during the 2022 mpox outbreak as contact structure and behaviour needed explicit representation and these models were not adapted to include this [5]. Similarly, cross-sectional viral load measurements have been shown to provide useful information on epidemic dynamics [6], yet this method has not been incorporated into most models used to inform policy even where such data was routinely available [3,7]. Multi-model efforts synthesise cross-domain expertise by combining forecasts or scenario projections from multiple teams (e.g., [8,9]) and have been shown to improve predictive accuracy [10,11], yet teams rarely collaborate on elements of their models despite shared goals. For evidence to inform policy it must be timely, explicit about methods and limitations, and as simple as possible whilst remaining rigorous [12]. In practice, while rigorous policy-relevant modelling is possible it usually comes at the cost of the flexibility needed for cross-domain collaboration and timely evidence generation.

Probabilistic programming languages (PPLs) such as Stan or LibBi [13,14], and modelling frameworks such as pomp and monty [15,16], have enabled rigorous joint modelling and inference, reduced development effort, improved model quality, and supported workflow best practices compared with manual specification [17–19]. However, they have not enabled the flexibility and cross-domain collaboration needed to provide evidence in a timely manner. This may be because they do not provide straightforward ways to share or reuse model components; the practical unit of reuse remains the whole model or its codebase, not the underlying epidemiological concepts. This fragmentation is evident in the field. For example, at least seven packages estimate effective reproduction numbers

using renewal-based approaches in Stan [20–26], yet none share components despite overlapping features and, in some cases, the same authors. Reproduction number estimation using wastewater highlights challenges when different domain expertise is required: tools were implemented by non-wastewater experts with different levels of focus on sampling details, share no components and it is not clear which modelling choices matter [25,26]. To resolve these issues some efforts have aimed for greater generality: `epinowcast` provides a modular Bayesian framework supporting flexible model specification, and `epidist` [27] extends Bayesian Regression Models using Stan (`brms`) [28] for delay distribution estimation. Yet these, and other similar efforts, remain monolithic and inflexible, difficult for users to contribute to or adapt beyond supported functionality, and limited in scope.

To work around these flexibility constraints, some analyses choose to chain multiple models together, fitting separate models sequentially and passing outputs to subsequent stages [29–31]. This allows for flexibility but prevents full propagation of uncertainty and can introduce biased estimates compared with the rigour of fully joint models [32]. Modelling using data from the UK Office for National Statistics Community Infection Survey (CIS) demonstrates this approach and resulting challenges [3]. From April 2020 to March 2023, the CIS tested over four million swabs from more than 150,000 households at a cost exceeding £500 million. Identifiability concerns meant external modellers were able to access only summarised prevalence estimates rather than underlying observations. This created a cascade where estimates were used as inputs to subsequent analyses: prevalence estimates became inputs for incidence [33] and reproduction number estimates [34], which themselves informed policy analyses [4,35,36]. At each stage, uncertainty from earlier estimates was approximated and modelling assumptions inherited without the ability to evaluate their impact. This may have impacted the evidence used in policy-relevant decisions about variant transmissibility and severity [4,35,36].

Recent developments in computational statistics and scientific computing demonstrate the potential for composable modelling, in which components can be reused across contexts and combined in different configurations whilst maintaining statistical rigour, to address the tension between flexibility and rigour. As an example, advances in `Turing.jl`, a probabilistic programming language built in Julia [37], have introduced submodel interfaces that enable composable probabilistic programming, offering a pathway for epidemiological model composition [38,39]. In other fields, domain-specific languages (programming languages tailored to specific application domains that use terminology and concepts familiar to domain experts rather than general programming constructs) built in Julia have enabled composable models: `HydroModels.jl` [40] for hydrological modelling and `SpeedyWeather.jl` [41]. For the CIS example above, if those analysing the primary data used models composed from submodels, they could publish intermediate and fully deidentified results that others could, more easily, integrate into their modelling using Markov melding or related methods [42]. This would avoid implicit assumptions by making connections between components explicit and improve uncertainty propagation.

In this paper, we first propose requirements for composable infectious disease modelling based on the challenges we have identified, our experience developing infectious disease models during recent outbreaks, and established best practices for Bayesian modelling. We then present a proof of concept that addresses these requirements, combining the flexibility of chaining models together with the statistical rigour of joint models through composable epidemiological components. Our approach enables building block style model construction through standardised interfaces similar to those used in `SpeedyWeather.jl` [41]. Our proof of concept supports composability of a range of model families including ordinary differential equations and mixed equation types, with the potential for different computational backends, implemented as a domain-specific language intended to be implemented

as optional package extensions [43]. We demonstrate our approach using an autoregressive model example to illustrate the proposed compositional pattern (building models by combining composable components) and component swapping capabilities. Through three case studies using the prototype, we show how these latent process components can be composed with epidemiological models to estimate time-varying reproduction numbers: the first replicates a COVID-19 analysis for South Korea using a renewal process [44]; the second extends these components with reporting delays and day-of-week effects to replicate EpiNow2, a real-time nowcasting tool [21]; the third replicates an ordinary differential equation model analysis of influenza outbreak data [45]. Finally, we discuss alternative design approaches, evaluate the strengths and limitations of our compositional approach, and identify key areas for future development.

2 Proposed requirements for composable infectious disease modelling

Based on the challenges outlined above, our experience developing infectious disease models during recent outbreaks, and established best practices for Bayesian modelling [18], we propose requirements for composable infectious disease modelling organised into six themes (Table 1). The requirements in the uncertainty and inference theme and the model structure theme aim to address building and fitting models with appropriate uncertainty propagation. The requirements in the population heterogeneity theme aim to address the need to model diseases across different groups whilst reducing the additional bookkeeping this typically requires. The requirements in the accessibility and adoption theme aim to lower barriers for diverse users, enable incremental uptake, and support the propagation of domain expertise into composed models. The requirements in the interoperability theme aim to ensure components work together and integrate with external tools. The requirements in the modelling workflow theme aim to support the development of validated models that can answer key stakeholder questions in decision maker relevant timelines [12], and enable emerging opportunities for large language model assisted model construction. Whilst we have presented this as a mapping of problems to requirements, in reality there is overlap; for example, standardised interfaces should enable component compatibility, support handling of multiple strata, and ensure proper uncertainty propagation.

Table 1: Proposed requirements for composable infectious disease modelling with motivating problems.

Theme	Proposed requirement	Motivating problem
Uncertainty & inference	Probabilistic model specification with full uncertainty propagation	Decisions must account for incomplete knowledge about infection risk, transmission, and observation [46]
	Joint modelling and staged inference [42] with proper uncertainty propagation	Chaining models introduces implicit assumptions and cascading biases (see CIS example in the introduction)
	Automatic differentiation [47]	Complex models require efficient fitting methods, and many modern inference approaches rely on gradients [48]
	Switchable inference backends	The best inference method varies by model and data combination; no single approach works universally
Model structure	Clear separation between model components	Infection processes, other latent processes, and observation processes require distinct expertise and should be reasoned about separately
	Easy nesting of models within models	Adding new data sources (such as case observations to a wastewater model) or processes with existing models (such as incubation periods to early outbreak analysis) can require modifying entire models (Figure 1)
	Support for multiple modelling paradigms	Infectious disease models take many forms (such as compartmental models, agent-based simulations, and network transmission models)
Population heterogeneity	Arbitrary stratification for all model components	Diseases affect populations heterogeneously across age, location, and risk group [1]
	Automatic management of multi-group structure	Managing dimensions and interactions across groups (such as age groups and locations) can be complex and error-prone
	Programmatic generation and modification of model components	Extending models to additional groups (such as age groups or locations) and adapting existing components often requires repetitive code

Theme	Proposed requirement	Motivating problem
	Partial pooling [49] in order to share information across groups	Sparse data in individual groups can require borrowing information between them
Accessibility & adoption	Clear, concise modelling language	Diverse users including those without programming expertise need to specify models quickly
	Components contain both the structure and parameter prior distributions	Domain expertise can be difficult to transfer between models or modelling teams
	Support for incremental adoption of the approach	Requiring complete rebuilds of existing models can be wasteful and time-consuming [12]
	Components functional as standalone tools	Components should provide value even outside the compositional framework
Interoperability	Standardised interfaces between components	Components developed independently may be incompatible and uncertainty may not propagate correctly between them
	Data-agnostic model definitions	Models need to generalise across datasets, support prior predictive checks, and enable forecasting
	Composability with machine learning (ML) approaches	Integration with neural networks, Gaussian processes, and other ML tools enables leveraging that expertise [50]
Modelling workflow	A single specification for simulation and inference	Both simulation (for prior predictive checks and forecasting) and inference are needed [18], but implementing both separately can be resource intensive and difficult to keep in sync
	Rapid model iteration	Modelling workflows [18], especially during outbreaks where evidence and requirements can change rapidly [12], require models that are easy to iterate on
	Easy model and model component validation and exploration	Complex models can be difficult to debug, validate, and understand
	Modelling language optimised for large language model assisted construction with automated validation	Large language models may be useful for constructing models but can make errors and often do not fully capture expert knowledge [51]

3 Our approach

Meeting these requirements requires programming with probabilities, for which probabilistic programming languages are designed [17]. We also need a probabilistic programming language that supports automatic differentiation for modern inference, the ability to program over model structure itself to enable model nesting and composition, and access to as wide an ecosystem as possible to avoid lock-in and enable integration with existing scientific computing tools. As far as we are aware, only probabilistic programming languages built in Julia [37] provide the metaprogramming capabilities (writing code that generates or manipulates other code) needed to create domain-specific abstractions that can handle arbitrary stratification, standardised interfaces between components, and programming over the model structure. Among Julia's PPL options, `Turing.jl` [39] best meets our requirements with mature submodel support for nesting models within models, extensive inference algorithm choices, and its implementation as a light abstraction layer (a simplified interface that hides complex implementation details whilst enabling access to underlying functionality) on top of the wider Julia ecosystem. Additional benefits of Julia include eliminating the two-language problem, leveraging multiple dispatch (a programming paradigm where function behaviour is determined by the types of all arguments rather than just the first, enabling automatic selection of the most specific implementation for given inputs) for clean component composition, and accessing the mature SciML ecosystem [52] for differential equations, neural networks [50,53], and other scientific computing tools.

Our approach uses a two-layer architecture with a high-level domain-specific language (DSL) for epidemiological modelling and a low-level implementation using `Turing.jl`. Though importantly we are not locked in to this choice as the DSL is agnostic of the backend used. This separation enables incremental adoption without rebuilding existing models to use our DSL, with all components remaining functional as standalone tools outside the compositional framework. The DSL layer provides clear, concise model specification using epidemiological concepts, enabling domain experts to contribute components encapsulating their specialised knowledge without understanding the low-level details of all framework elements. The backend layer aims to handle the automated bookkeeping of stratification, interface validation, and uncertainty propagation whilst supporting multiple inference approaches and auto differentiation options by leveraging the `Turing.jl` and wider Julia ecosystems. This structure enables integration of diverse data sources by allowing researchers to add a submodel with its own latent and observation process to an existing model, for example adding wastewater measurements to a case-based renewal model or clinical biomarkers to transmission dynamics without modifying the core infection model. It also facilitates staged inference approaches such as Markov melding [42], meaning that components can be fitted separately and combined with proper uncertainty propagation.

To demonstrate cross-ecosystem accessibility, we also developed `EpiAwareR` [54], an R interface to the prototype using `JuliaCall` [55]. This enables R users to access the compositional modelling framework without requiring Julia programming knowledge, specifying models using familiar R syntax while gaining access to the full composable component library.

Figure 1 demonstrates how our proposed approach could enable component reuse across different epidemiological applications. Four example applications (wastewater surveillance, biomarker modelling, early outbreak analysis, and incubation period estimation) each compose models from components of different types. The schematic also highlights two examples of component reuse. The incubation period model appears in all four applications, and the within-host viral kinetics model is shared between biomarker modelling and wastewater surveillance. This highlights how components

developed for one application could be incorporated into others when they share common underlying processes.

3.1 Domain-specific language structure

Our prototype DSL builds on Julia's type system to enable composable epidemiological modelling through two key design patterns. First, abstract types define interfaces that implementations must follow, analogous to contracts specifying what operations a model component must support rather than how it implements them. All model components inherit from a parent `AbstractModel` type, establishing a common foundation whilst allowing specialised behaviour through subtypes. Second, structures (data containers that group related variables and their types together) contain other structures as fields, a struct-in-struct pattern, similar to the approach used in `SpeedyWeather.jl` [41], that allows complex models to be built by nesting simpler components. This pattern enables models to be assembled like building blocks whilst maintaining clear boundaries between different epidemiological processes.

We organise model components into three abstract type hierarchies, each of which inherits from `AbstractModel`, corresponding to distinct epidemiological processes. `AbstractEpiModel` represents infection generation processes such as renewal models or ordinary differential equation transmission dynamics. `AbstractLatentModel` captures time-varying parameters and unobserved processes such as changing reproduction numbers or reporting rates, implemented through structures like autoregressive processes, random walks, or moving averages. `AbstractObservationModel` links latent states to observed data by encoding measurement processes such as reporting delays, aggregation over time periods, and observation error distributions. These structures are data-agnostic, specifying what to do when they encounter data rather than containing data themselves, making model definitions reusable across different datasets and scenarios. Each hierarchy supports multiple concrete implementations that can be swapped to compare modelling assumptions whilst keeping other components fixed.

Models can compose across these hierarchies rather than being restricted to combining components within a single type. For example, an observation model can contain a latent process as a field to represent time-varying ascertainment, or wrap another observation model to add reporting delays. This cross-hierarchy composition extends the building block analogy to include more complex models. For instance, we can construct a delay convolution observation model, `LatentDelay`, using an underlying observation model and a delay distribution model.

The `EpiProblem` structure is a top-level container assembling these components into a complete epidemiological model. It holds an infection process (`epi_model`), a latent process (`latent_model`), an observation process (`observation_model`), and a time span for inference or simulation. The latent model generates time-varying epidemiological parameters, the infection model uses these parameters to simulate disease transmission and generate infections, and the observation model links these latent infections to observed data through measurement processes. However, this is just the top-level structure: each submodel can itself contain any of these abstract types, enabling flexible composition such as observation models that include latent processes for time-varying ascertainment. Structures can be modified in place using tools like `Accessors.jl` [56], enabling both model iteration and patterns such as partially pooled models that update low-level priors based on grouping structures. The abstract type system enables shared methods across all model components, such as print methods (shown below) for displaying model specifications or functions for visualising model directed acyclic graphs. This approach also allows for the creation of mappings between

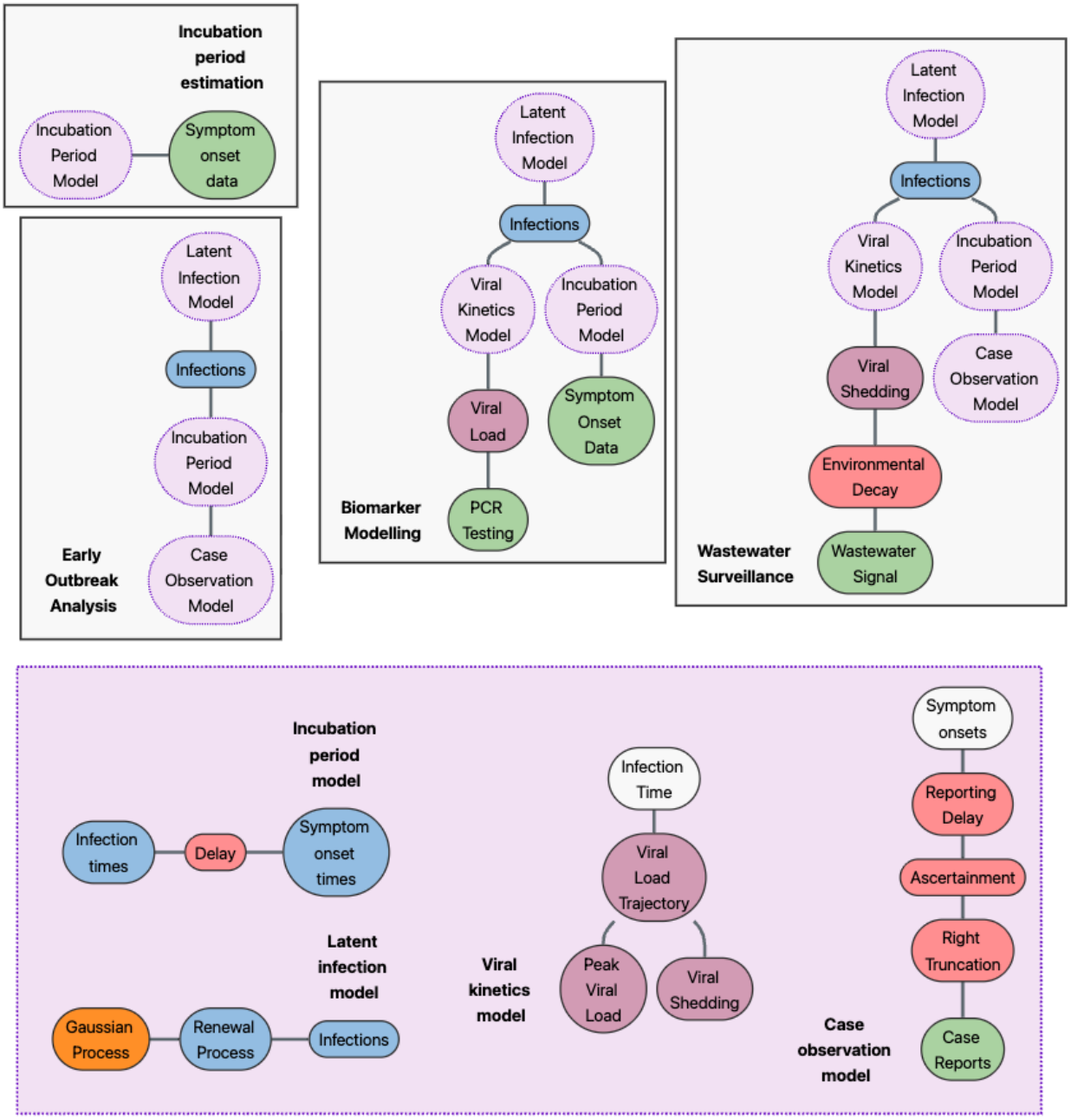

Figure 1: Demonstration of composability showing how four applications share common components. Colours correspond to component types: infection processes (light blue), latent infection quantities (pale blue), statistical processes (orange), epidemiological latent processes (purple), observation modifiers (red), and observation models (green). Four shared submodels are highlighted with purple dashed borders: the Latent Infection model (GP and Renewal, reused across all applications), the Incubation Period model (reused across all applications), the Case Observation model (reporting delay, ascertainment, and right truncation, shared between Early Outbreak Analysis and Wastewater Surveillance), and the Viral Kinetics model (shared between Biomarker Modelling and Wastewater Surveillance). Applications reference these shared models via matching purple dashed boxes. The Incubation Period Estimation application demonstrates how shared models can be calibrated from independent data sources.

submodels such as one to one, one to many, and many to many mappings so that, for example, a single infection process can be linked to multiple observations models by specifying a mapping model.

To demonstrate the approach we focus on `AbstractLatentModels`, and build an ARIMA(2,1,1) process through composition, as shown in the ARIMA panel of Figure 2. In the case studies (Section 4) this is then composed with infection and observation models to estimate time-varying reproduction numbers. We start with an autoregressive order two (AR(2)) process. Mathematically, the AR(2) process is:

$$Z_t = \rho_1 Z_{t-1} + \rho_2 Z_{t-2} + \epsilon_t, \quad \epsilon_t \sim \text{Normal}(0, \sigma)$$

In our prototype DSL, this is defined using the `AR` struct. We use priors inspired by [57] for illustration. Prior distributions are specified using `Distributions.jl` [58], the standard probability distributions package in the Julia ecosystem, which provides a unified interface for probability distributions that is interoperable with both our framework and `Turing.jl`.

```
using EpiAware, Distributions
ar2 = AR(;
    damp_priors=[truncated(Normal(0.2, 0.2), 0, 1),
        truncated(Normal(0.1, 0.05), 0, 1)],
    init_priors=[Normal(0, 0.2), Normal(0, 0.2)],
    ϵ_t=HierarchicalNormal(std_prior=HalfNormal(0.1))
)
```

This constructor has created the following struct definition.

```
ar2
```

```
AR(damp_prior = Product(Truncated(0.2, 0.2, 0.0, 1.0)),
   init_prior = TuringScalMvNormal([0.0, 0.0], 0.2),
   p = 2,
   ϵ_t = HierarchicalNormal(mean = 0.0,
                            std_prior = HalfNormal(0.1),
                            add_mean = false))
```

Prior samples from this model are shown in Figure 3 A. Another common latent model is the moving average model

$$Z_t = \epsilon_t + \theta \epsilon_{t-1}, \quad \epsilon_t \sim \text{Normal}(0, \sigma)$$

The `MA` struct defines this in the same way that the `AR` did for the AR process.

```
ma1 = MA(;
    θ_priors=[truncated(Normal(0.0, 0.2), -1, 1)],
    ϵ_t=HierarchicalNormal(std_prior=HalfNormal(0.1))
)
```

Prior samples from this process are shown in Figure 3 B. A popular combination of these models is the autoregressive moving average (ARMA) model that can have different orders for both the AR

and MA components. An ARMA(2,1) can be defined as:

$$Z_t = \rho_1 Z_{t-1} + \rho_2 Z_{t-2} + \epsilon_t + \theta \epsilon_{t-1}$$

The ARIMA panel of Figure 2 shows this composition as a directed graph where AR(2) connects to MA(1) to form the combined process. In our DSL this can be represented as a composition of the AR and MA structs by updating the AR error term:

```
using Accessors
arma21 = @set ar2.ϵ_t = ma1
```

The result of this step has been to update the `ar2` struct so that the definition of the moving average model is nested inside it.

```
arma21
```

```
AR(damp_prior = Product(Truncated(0.2, 0.2, 0.0, 1.0)),
   init_prior = TuringScalMvNormal([0.0, 0.0], 0.2),
   p = 2,
   ϵ_t = MA(θ = Product(Truncated(0.0, 0.2, -1.0, 1.0)),
            q = 1,
            ϵ_t = HierarchicalNormal(mean = 0.0,
                                     std_prior = HalfNormal(0.1),
                                     add_mean = false)))
```

The combined model dynamics are shown in Figure 3 C. Similarly, autoregressive integrated moving average (ARIMA) models extend ARMA by adding differencing operations that transform the series

$$\Delta Z_t = Z_t - Z_{t-1}$$

So that the ARIMA(2,1,1) model is defined as an ARMA(2,1) model for the first order differences:

$$\Delta Z_t = \rho_1 \Delta Z_{t-1} + \rho_2 \Delta Z_{t-2} + \epsilon_t + \theta \epsilon_{t-1}$$

We compose the ARMA model with a differencing operation using the `DiffLatentModel` wrapper:

```
arima211 = DiffLatentModel(arma21, Normal(0, 0.2); d=1)
```

Prior samples from the full model are shown in Figure 3 D. Alternatively, EpiAware provides an `arima()` constructor function that simplifies this specification.

Other latent models extend modelling options through combining models additively, multiplicative scaling, and piecewise processes. This approach enables representation of arbitrary latent processes through composition. In the case studies (Figure 2), we use these latent processes to generate trajectories of time-varying reproduction numbers R_t. These are then linked with infection models via `AbstractEpiModel` implementations such as `Renewal`, which computes expected infections using the discrete-time renewal equation $I_t = R_t \sum_s g_s I_{t-s}$ where g_s is the generation interval distribution. These latent process models are also used to capture time-varying ascertainment rates in observation models.

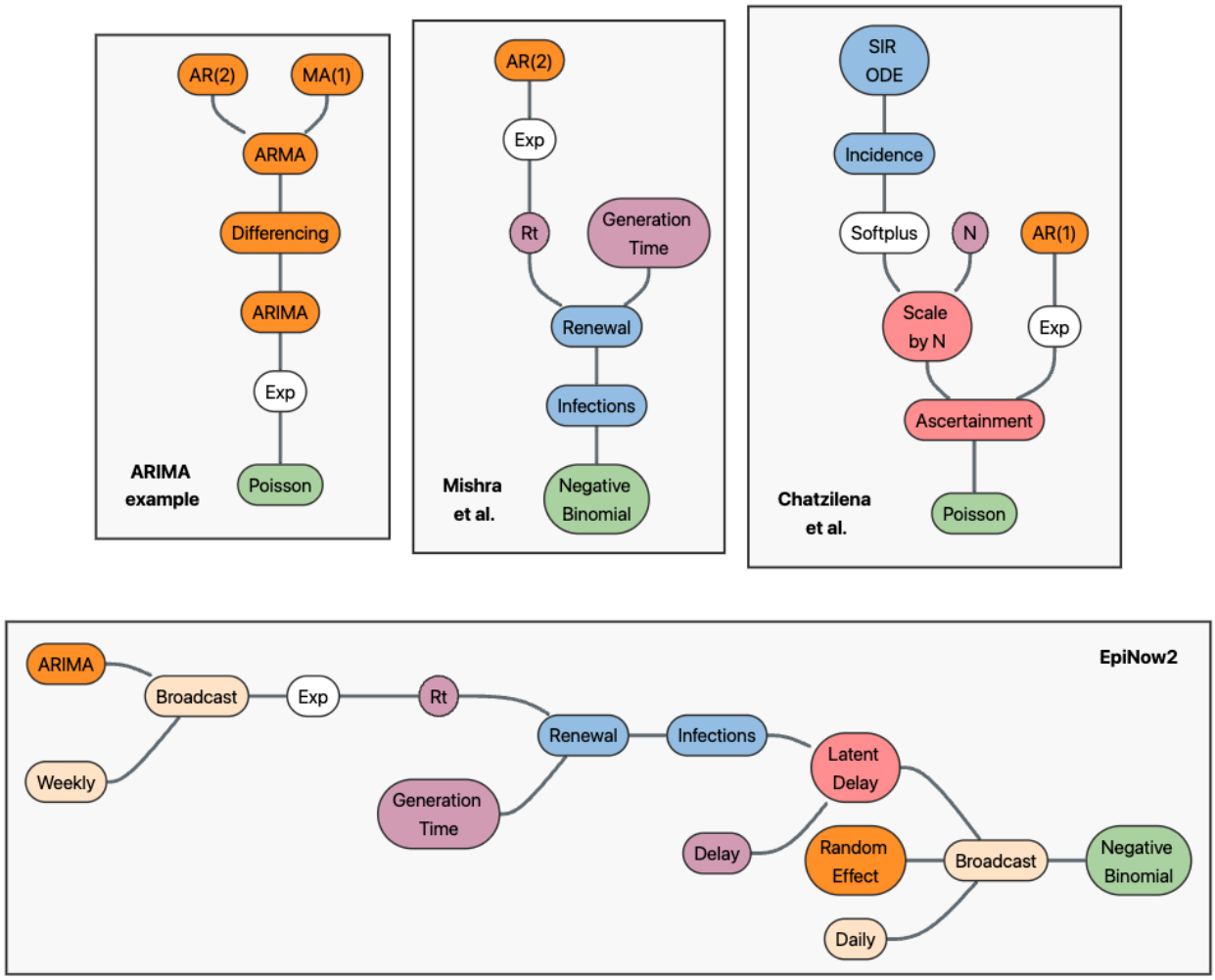

Figure 2: Model structure for the ARIMA example and three case studies showing component reuse. The ARIMA panel shows how AR(2) and MA(1) components compose into ARIMA(2,1,1) with Exp transformation and Poisson observation. Mishra et al. reuses the AR(2) component for modelling time-varying Rt combined with a Renewal infection process and Negative Binomial observation. EpiNow2 extends this by reusing the full ARIMA structure with weekly broadcasting, adding Latent Delay for reporting delays and day-of-week ascertainment effects. Chatzilena et al. demonstrates an alternative infection process using a SIR ODE with Softplus transformation, scaled by population size N, and AR(1)-based time-varying ascertainment. Colours indicate component types: statistical models (orange), infection processes (light blue), latent infection quantities (pale blue), epidemiological parameters (purple), observation modifiers (red), broadcasting operations (peach), transformations (white), and observation models (green).

3.2 Backend implementation: `Turing.jl` interface

Our DSL translates to `Turing.jl` models through generate functions that dispatch on abstract type hierarchies. Each abstract type (`AbstractLatentModel`, `AbstractEpiModel`, `AbstractObservationModel`) has a generate function interface that concrete model types must implement a method to dispatch upon. When a generate function receives a model component, Julia's multiple dispatch automatically selects the appropriate implementation based on the component's type, producing `Turing.jl` [39] code using the `@model` macro. Unit tests are used to ensure concrete implementations satisfy interface requirements, enabling all components to interoperate correctly regardless of which specific model types are composed together. This approach means new model types integrate seamlessly without modifying existing code, and users can swap components to compare modelling assumptions whilst keeping other parts of the model fixed.

The generated `Turing.jl` models serve dual purposes: simulation by sampling from prior distributions, or inference by conditioning on observed data and applying `Turing.jl`'s suite of algorithms including gradient-based methods like the No U-Turn Sampler (NUTS) [48]. These are standard `Turing.jl` models with no special constraints beyond those imposed by the `Turing.jl` framework itself, supporting all `DynamicPPL.jl` [59] operations such as parameter fixing, model conditioning, and posterior predictive sampling. Generated components can be used directly as standalone models or nested within other `Turing.jl` models using the `@submodel` macro, enabling incremental adoption where only parts of a model use the compositional framework whilst other parts use custom `Turing.jl`.

Many computational components are backend-agnostic, containing no probabilistic programming constructs and therefore portable to other frameworks. For example, we use a common pattern, `accumulate_scan`, which builds on the base accumulate function to model iterative temporal processes through step functions. These step functions are structs built on the `AbstractAccumulationStep` interface that implement a callable (a struct that can be invoked like a function) defining the single-step update rule. Mathematical utilities such as functions for converting between reproduction numbers and growth rates, discretising continuous delay distributions, and reparameterising observation error models are similarly backend-agnostic. This separation enables a package extension pattern where standard Julia packages provide domain-specific functionality (e.g. `CensoredDistributions.jl` [60]) whilst the compositional layer is added as an optional extension, allowing users to adopt the framework incrementally without requiring their entire workflow to commit to the compositional approach.

Returning to our example from the DSL section, the autoregressive process can be mapped from its high-level representation to a `Turing.jl` model using the `generate_latent` function and multiple dispatch, which generates the following `Turing.jl` model:

```
@model function EpiAwareBase.generate_latent(latent_model::AR, n)
    p = latent_model.p
    @assert n>p "n must be longer than order of the autoregressive process"

    ar_init ~ latent_model.init_prior
    damp_AR ~ latent_model.damp_prior
    @submodel ϵ_t = generate_latent(latent_model.ϵ_t, n - p)

    ar = accumulate_scan(ARStep(damp_AR), ar_init, ϵ_t)
```

```
    return ar
end
```

The key line `@submodel ε_t = generate_latent(latent_model.ε_t, n - p)` enables composition by delegating to whatever error model was provided (here, the MA(1) component). The AR dynamics are implemented through a custom accumulation step that maintains the autoregressive state.

```
function (ar::ARStep)(state, ϵ)
    new_val = dot(ar.damp_AR, state) + ϵ
    new_state = vcat(state[2:end], new_val)
    return new_state
end
```

This step function works with `accumulate_scan` to build the AR series by applying the autoregressive equation at each time step. The MA model has a similar structure with its own internal step function. The `accumulate_scan` pattern enables composable iteration steps, allowing complex processes like renewal models with susceptible depletion to be built by composing simple step operations. This design means we only need to write the single-step operation without worrying about the iteration process, making components more modular and reusable.

The full ARIMA(2,1,1) model we defined in the DSL section can then be generated using the same approach, producing the following `Turing.jl` model:

```
@model function EpiAwareBase.generate_latent(latent_model::DiffLatentModel, n)
    d = latent_model.d
    @assert n>d "n must be longer than d"
    latent_init ~ latent_model.init_prior

    @submodel diff_latent = generate_latent(latent_model.model, n - d)

    return _combine_diff(latent_init, diff_latent, d)
end
```

The `DiffLatentModel`'s `@submodel diff_latent = generate_latent(latent_model.model, n - d)` calls the ARMA model, which in turn calls its composed AR and MA components, then applies differencing through cumulative summation. This recursion through `@submodel` enables arbitrary composition while maintaining separation between components.

To demonstrate fitting, we combine our ARIMA(2,1,1) with a log-parameterised Poisson observation model, modelling the growth rate using our ARIMA process. From here we use `Turing.jl` functionality directly to illustrate the interoperability between our DSL and the underlying PPL.

```
using DynamicPPL, Turing

@model function arima_with_obs(arima_spec, n_timesteps)
    @submodel Z_t = generate_latent(arima_spec, n_timesteps)
    y_obs ~ product_distribution(LogPoisson.(Z_t))
    return y_obs
end
```

We can then generate synthetic data with fixed parameters which we sample from the prior

distribution using the `rand`, and `fix` functions and calling the model to simulate the observations. We first define the model.

```
n_timesteps = 40
gen_model = arima_with_obs(arima211, n_timesteps)
```

Then sample from it,

```
simulated_params = rand(gen_model)
```

Now we have parameters we can simulated some data using the generative model by fixing the random variables using the sampled parameters and the `fix` function. We can then call it, like any normal function, to get simulated observations for `y`.

```
fixed_model = fix(gen_model, simulated_params)
y_observed = fixed_model()
```

For inference, we condition on the generative model using simulated observations and the `condition` function or here the equivalent `|` notation. Now we have a model conditioned on data we can fit it using our choice of approach supported by `Turing.jl`. Here we decide to use the No-U-Turn Sampler (NUTS) [48], a popular variant of MCMC.

```
conditioned_model = gen_model | (; y_obs = y_observed)
chains = sample(conditioned_model, NUTS(), MCMCThreads(), 2000, 4)
```

We can then compare our posterior distributions to the true sampled values from our ARIMA(2, 1, 1) model using `PairPlots.jl` [61] with the `CairoMakie.jl` [62] backend for visualisation (Figure 3 E). The posterior distributions recover the simulated parameter values.

4 Case studies

We demonstrate our prototype compositional modelling DSL by recreating three published epidemiological analyses (Figure 2): "On the derivation of the renewal equation from an age-dependent branching process" [44], "EpiNow2: Estimate Real-Time Case Counts and Time-Varying Epidemiological Parameters" [63], and "Contemporary statistical inference for infectious disease models using Stan" [45]. Figure 2 highlights component reuse across the case studies: Mishra et al. reuses the AR(2) component from the ARIMA example, the EpiNow2 replication reuses the full ARIMA structure, and Chatzilena et al. demonstrates an alternative Susceptible-Infected-Recovered (SIR) infection process.

We provide an overview of each case study in the following subsections, with full implementation details in the Supplementary Information. The first two case studies have also been replicated using the `EpiAwareR` R interface [54], with R implementations available in the package vignettes, demonstrating cross-ecosystem accessibility. All code and data for reproducing the analyses are available [64].

4.1 On the derivation of the renewal equation from an age-dependent branching process: an epidemic modelling perspective

Mishra et al. [44] estimate time-varying effective reproduction numbers (R_t, the average number of secondary infections caused by an infected individual at time t) from case data by combining the

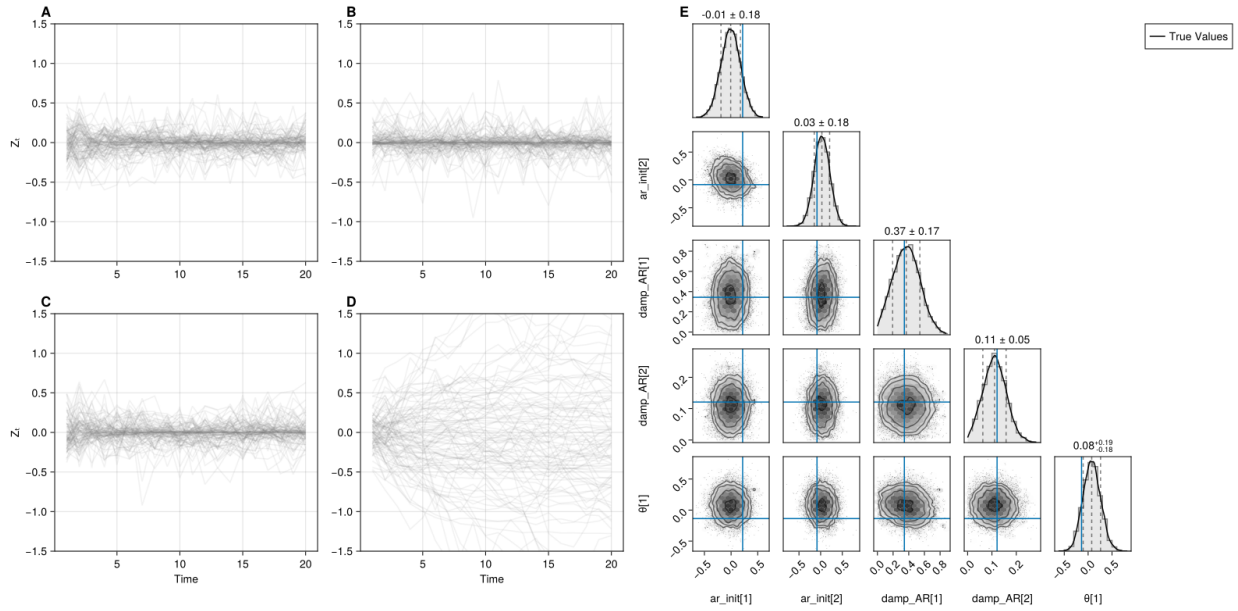

Figure 3: Prior predictive samples and posterior inference for latent time series models. (A) AR(2) process showing autoregressive dynamics. (B) MA(1) process showing moving average behaviour. (C) ARMA(2,1) combining AR and MA components. (D) ARIMA(2,1,1) with differencing. (E) Posterior density for ARIMA(2,1,1) parameters showing pairwise relationships between AR damping coefficients and MA coefficient, with true parameter values (blue) used to generate synthetic data.

renewal equation with a negative binomial observation model.

Our replication composes an AR(2) latent process for log R_t (Figure 2), a renewal infection process [65] using a discretised serial interval distribution [66], and a negative binomial observation model for case counts. We use `AR` to generate the latent log R_t trajectory, `Renewal` to compute expected infections via the renewal equation $I_t = R_t \sum_s g_s I_{t-s}$, and `NegativeBinomialError` to link expected infections to observed cases with overdispersion. We assemble these components and use Pathfinder [67] to initialise NUTS sampling.

Figure 4 shows model components and posterior analysis. Panels A-C demonstrate prior predictive checks for each component in isolation: the AR(2) latent process for log R_t, the renewal model for latent infections, and the negative binomial observation model. Panel D compares the continuous serial interval distribution with its discretised form. Panels E-F show posterior predictive distributions for daily cases and time-varying R_t, recovering the main finding that R_t in South Korea peaked at approximately 10 before rapidly dropping below 1 in early March 2020.

4.2 EpiNow2: Estimate real-time case counts and time-varying epidemiological parameters

`EpiNow2` [63] is a widely used package for real-time situational awareness in infectious disease surveillance, estimating case counts and epidemiological parameters whilst accounting for reporting delays, right-truncation, and day-of-week effects.

Our replication reuses the ARIMA(2,1,1) and negative binomial components from the main text and Section 4.1, extending them with piecewise constant weekly R_t values, incubation period and

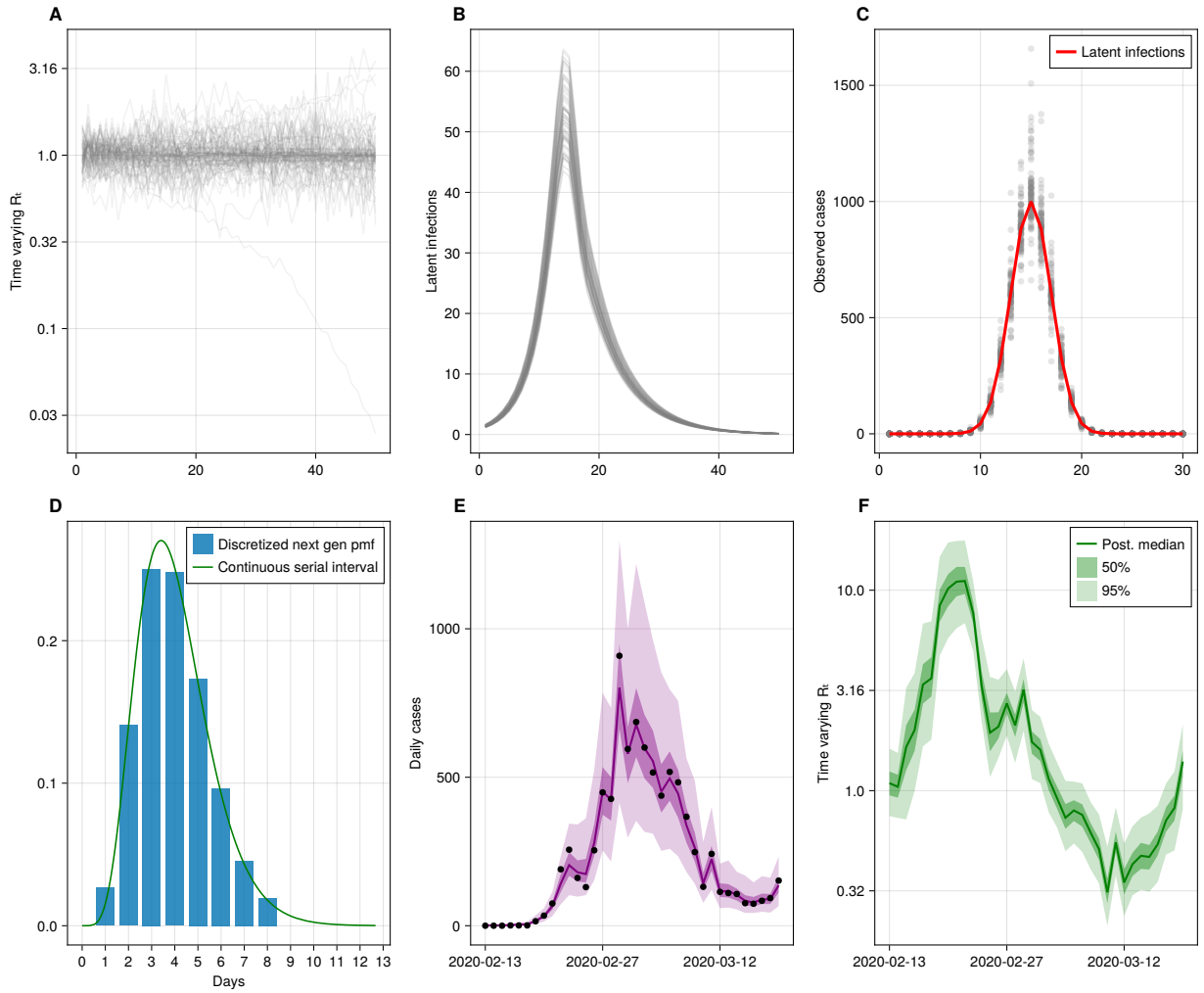

Figure 4: Model components and posterior analysis for Case Study 1. (A) Prior samples from the AR(2) latent process for log R_t over 50 days. (B) Prior samples from the renewal model conditional on a fixed R_t trajectory. (C) Prior samples from the negative binomial observation model around a latent infection curve. (D) Comparison of the continuous serial interval distribution with its discretised pmf. (E) Posterior predictive distribution for daily cases with 50% and 95% credible intervals. (F) Posterior predictive distribution for time-varying R_t on a log scale.

reporting delay convolutions [68], and day-of-week reporting effects (Figure 2). Unlike Section 4.1 which uses the serial interval, this case study uses the generation time distribution; whilst serial intervals are often used as a proxy, this can be problematic [69] and is primarily done when generation time estimates are unavailable [70]. We use `broadcast_weekly` to convert the ARIMA(2,1,1) process to piecewise constant weekly R_t values, `LatentDelay` to convolve latent infections with incubation period and reporting delay distributions, and `ascertainment_dayofweek` to multiply expected observations by day-of-week effects via a softmax transformation.

Figure 5 shows model components and posterior analysis. Panels A-C show prior predictive checks for the piecewise constant weekly ARIMA process, the renewal model, and the composite observation model with delays and day-of-week effects. Panel D displays the incubation period and reporting delay distributions. Panels E-F show posterior predictive distributions, demonstrating that the model explicitly accounts for reporting delays whilst recovering similar R_t dynamics to Section 4.1.

4.3 Contemporary statistical inference for infectious disease models using Stan

Chatzilena et al. [45] demonstrate Bayesian inference for compartmental disease models using Stan, fitting an SIR model to single strain influenza outbreak data with both a simple Poisson observation model and a stochastic model with time-varying ascertainment to account for model misspecification and observation error.

Our replication introduces the SIR model as an alternative infection generating process to the renewal model used in the previous case studies (Figure 2). We compose an SIR compartmental model using `ODEProcess` from the SciML ecosystem [71], a Poisson observation model, and a stochastic observation model using `Ascertainment` for time-varying effects parameterised as an AR(1) process. We use a softplus transformation to ensure numerical stability when the ODE solver returns small negative values.

Figure 6 shows model components and posterior analysis. Panels A-B demonstrate prior predictive checks for the SIR dynamics and stochastic observation model including time-varying ascertainment. Panel C compares posterior R_0 distributions, with both models recovering $R_0 \approx 2$, consistent with *Chatzilena et al.* Panels D-F show posterior compartment trajectories and predictive cases. The deterministic model posterior predictive trajectories (Figure 6 E) appear less well calibrated than those from the stochastic model (Figure 6 F), with several data points falling outside the 95% prediction envelopes. This indicates that the flexible observation model compensates for potential misspecification in the deterministic model.

5 Discussion

In this paper we have identified barriers in epidemiological modelling arising from the tension between statistical rigour and the flexibility needed for cross-domain collaboration and timely policy-relevant evidence, proposed compositional infectious disease models as a solution, laid out a set of requirements for such frameworks, and showcased a proof of concept approach that meets these requirements. Our proof of concept consists of a domain specific language (DSL) built on top of the `Turing.jl` probabilistic programming framework, enabling compositional model construction through reusable components. This approach enables building block style model construction, maintaining the statistical rigour of joint models whilst providing the flexibility of chaining together simpler models. The autoregressive model example illustrated how more complex models emerge from simple component combinations using our proposed struct-in-struct pattern (where structures,

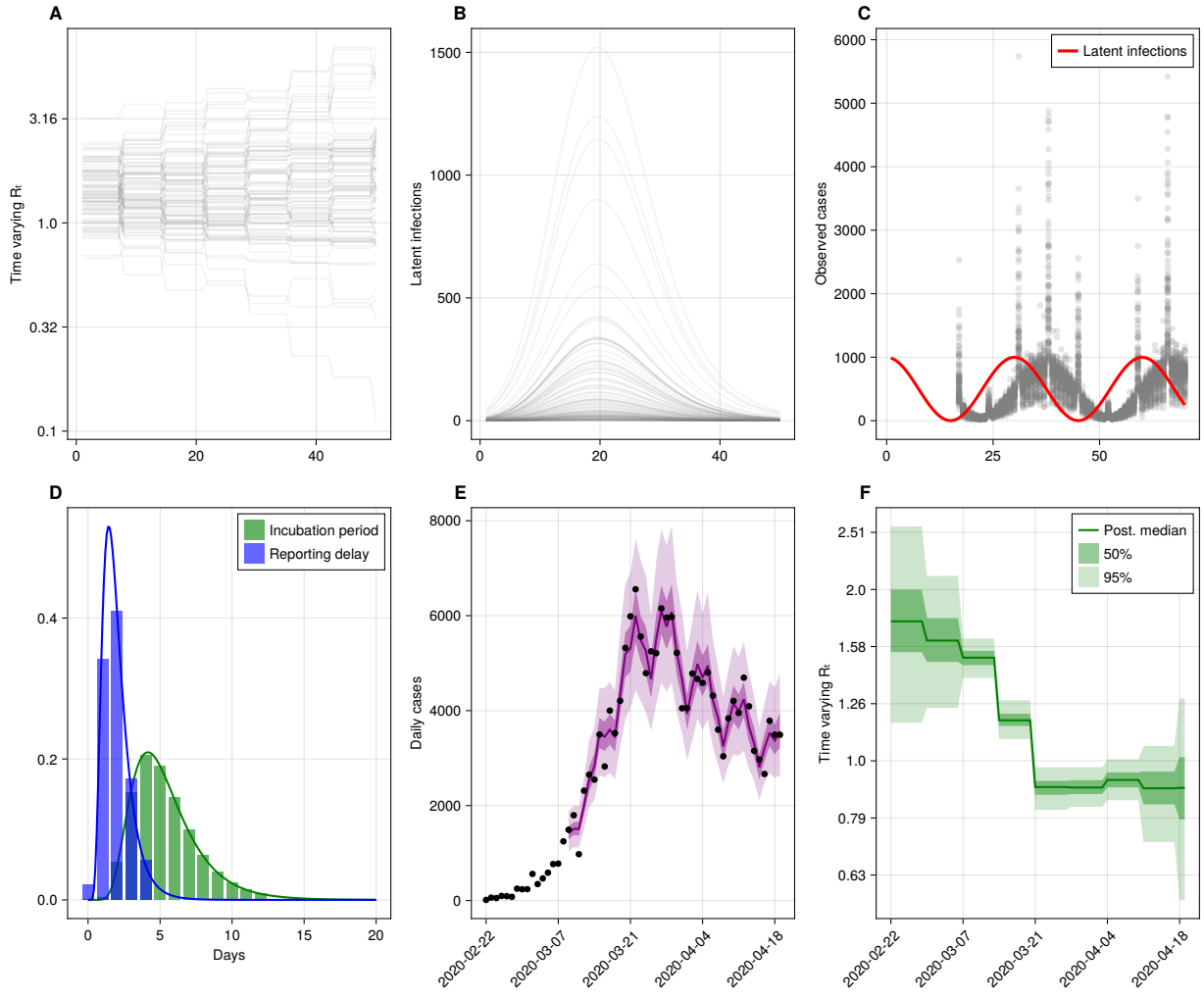

Figure 5: Model components and posterior analysis for Case Study 2. (A) Prior samples from the piecewise constant by week ARIMA(2,1,1) latent process. (B) Prior samples from the renewal model. (C) Prior samples from the composite observation model including delays and day-of-week effects. (D) Incubation period and reporting delay distributions. (E) Posterior predictive distribution for daily cases. (F) Posterior predictive distribution for time-varying R_t.

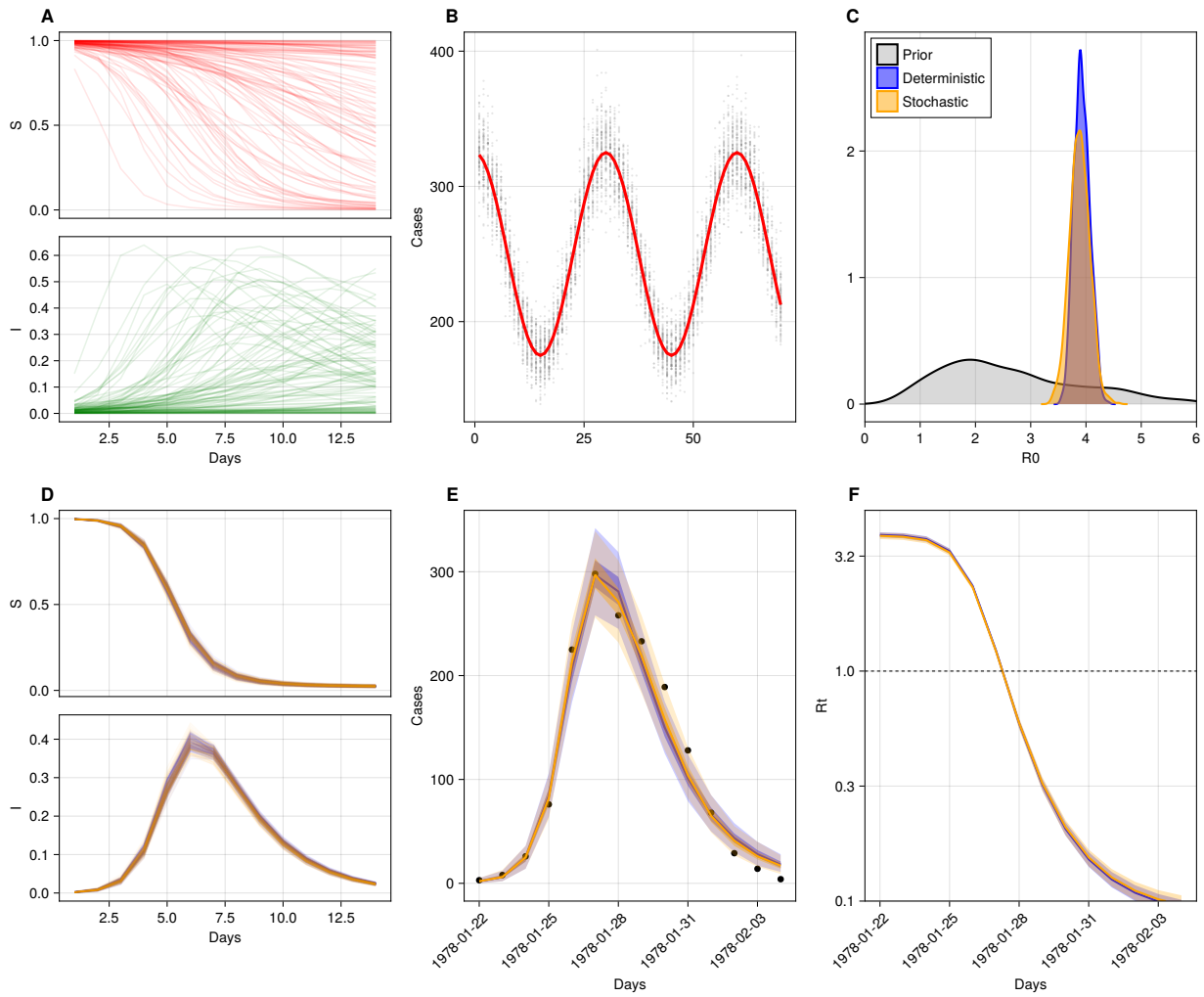

Figure 6: Model components and posterior analysis for Case Study 3. (A) Prior S (top) and I (bottom) compartment trajectories. (B) Prior samples from the stochastic observation model including time-varying ascertainment and Poisson sampling around a latent infection curve. (C) Prior (grey) and posterior R_0 distributions comparing deterministic (blue) and stochastic (orange) models; x-axis truncated at 6 to focus on the posterior, cutting the prior tail. (D) Posterior S (top) and I (bottom) compartment trajectories for both deterministic (blue) and stochastic (orange) models. (E) Posterior predictive cases for both models, with median and 50% credible intervals compared to observed data (black points). (F) Posterior R_t over time on a log scale for both models with reference line at $R_t = 1$.

data containers that group related variables and their types together, contain other structures as fields). Our three case studies (Figure 2), which were based on previous studies [21,44,45], demonstrate how these latent process components composed with infection and observation models can address real-world epidemiological problems across a range of modelling approaches.

A limitation of our proposal is that the requirements were developed without broader community input. However, they were based on our experience building infectious disease modelling frameworks and applied modelling. A key strength of our proposed approach is its modular design, which enables faster model development and component reuse whilst allowing comparison of modelling assumptions without significant reimplementation. For instance, adding a new data source such as wastewater surveillance to an existing case-based model requires only defining a wastewater submodel with its own latent and observation process, avoiding the need to reimplement existing components. This modularity also enables interdisciplinary collaboration, as domain experts such as virologists, clinicians, or environmental scientists can contribute specialised components encoding their knowledge without needing to review other components or understand the programming details. This design supports routine forecasting, where validated components can be refined across seasons for established forecasting and monitoring efforts [72–74]. It also supports outbreak response, where analysts could rapidly adapt existing models to novel pathogens by swapping infection processes whilst retaining validated observation components. For multi-model efforts, a diversity of approaches often improves robustness of findings (e.g., [4,75]), but the lack of common components risks masking differences in model assumptions from differences in implementation, preventing attribution of different results to their underlying causes [76]. However, our proposed approach is a proof of concept and not designed for ongoing use or widespread adoption. We only partially implemented support for automated mappings between latent and observed states, partial pooling approaches, and the current component library remains limited to relatively basic epidemiological patterns. The DSL layer manipulation tooling also cannot coordinate updates of related parameters such as model order and corresponding priors without manual intervention. Whilst our proof of concept has these limitations, the three case studies establish viability for real-world epidemiological problems.

A final limitation of our approach is the learning curve required to adopt this compositional pattern, though this should be mitigated as researchers can embed elements within their existing models and implement their own compositional elements with minimal understanding of the overall architecture.

Key strengths of building on `Turing.jl` [39] include mature submodel support for nesting models within models, extensive inference algorithm choices, and integration with the wider Julia ecosystem through `Distributions.jl` [58] and other packages. However, building on an evolving probabilistic programming language involves practical considerations. The `Turing.jl` PPL is not fully stable and since developing our DSL, the `@submodel` macro has changed syntax with breaking changes in how it handles prefixes and variables within submodels, meaning our current implementation is not compatible with the latest release. `Turing.jl` also has limited handling of numerical instability, so large counts being simulated can cause errors, and vectors that combine missing values with other values, which require manual handling. Profiling tools can be difficult to use because the probabilistic programming abstractions and macro expansions obscure the relationship between source code and runtime behaviour, making it challenging to identify performance bottlenecks. That said, recent `Turing.jl` changes may benefit composability: the reduced difference between distributions and submodels could enable more flexible model construction, and new conditioning workflows avoid passing observations through model layers.

Implementing in Julia [37] provides access to the full features of a programming language rather than a restricted modelling syntax, multiple dispatch for clean component composition, and the mature

SciML ecosystem [52] for differential equations and scientific computing. A key strength of the Julia ecosystem for compositional work is that domain experts develop specialised tools that interoperate through shared interfaces. However, this distributed development also creates challenges as it can be difficult to locate appropriate tools, and when issues emerge users may not know whether problems originate in `Turing.jl`, our DSL layer, or elsewhere in the ecosystem. Computational overhead from abstraction layers is also a concern, though new automatic differentiation backends like Enzyme [77] and Mooncake [78] could reduce costs in future implementations. Whilst our approach requires implementation in Julia, which is not currently common in epidemiology, the `EpiAwareR` R interface [54] demonstrates that such frameworks can bridge software ecosystems, and similar interfaces could be built for Python using `PythonCall.jl` [79].

Alternative approaches to compositional modelling exist in the literature. Category theory [80] provides formal mathematical foundations for composability, enabling complex models to be built from smaller building blocks, with the composed model being correct by construction. The AlgebraicJulia ecosystem demonstrates how category theory can be operationalised in software, with `AlgebraicPetri.jl` [81] and `AlgebraicDynamics.jl` [82] implementing operadic composition for hierarchical model construction. Symbolic-numeric frameworks such as `ModelingToolkit.jl` [83] and `Catalyst.jl` [84] from the SciML ecosystem [52] offer acausal component-based models with performance improvements through automated parallelisation and equation simplification, and link to AlgebraicJulia through shared reaction network representations [85]. In comparison, our approach currently lacks the formal compositional guarantees of category theory and automated optimisations of symbolic-numeric frameworks. However, current AlgebraicJulia tools focus on composing dynamical systems and provide limited direct support for probabilistic modelling; in contrast, our approach targets the full range of probabilistic models used in infectious disease inference. Although category theory could in principle provide guardrails for composing probabilistic models, the software does not yet support this. More complex compartmental models could be composed from reusable reaction components using `AlgebraicPetri.jl` or `Catalyst.jl`, with our approach managing the probabilistic aspects such as priors and observation models.

In comparison to our proposed compositional DSL approach, domain-specific infectious disease tools currently provide useful features but are generally designed for a limited number of use cases and do not cover the full range of epidemiological modelling approaches needed in practice [20–22,24]. Of the currently available tools, agent-based modelling approaches show the most promise for compositional modelling [86,87]. However, ABM calibration remains challenging, though differentiable ABMs may improve this [88], and there are settings where ABM approach may not be appropriate. A key aspect of our proposal is being backend agnostic: our DSL could support agent-based backends [89], and could also support hybrid approaches [90]. Generic modelling frameworks can be easier to generalise but difficult to adapt to include domain specific elements [91]. Probabilistic programming languages (PPLs) commonly used in infectious disease modelling also do not meet our proposed composability requirements. Stan is probably the most commonly used, but is optimised for implementing complete models rather than composing submodels [13]. Domain-specific PPLs such as `odin`/`monty` and `greta` trade generality for specialisation, none of the available options fully support nesting models within models, and there is limited support for state-of-the-art inference methods [16,92,93]. Alternative PPLs could support compositional approaches. `Gen.jl` [94] focuses on lower-level implementation of probabilistic programs compared to `Turing.jl`, with composition support through nested generative function calls and combinators of generative functions. Despite this, Gen-based tools demonstrate useful functionality for epidemiological modelling, such as `AutoGP.jl` [95], which implements real-time inference and automated composition of Gaussian process kernels. Genify [96] provides an approach that translates Julia code into Gen models, potentially easing adoption. Python-based

probabilistic programming languages such as NumPyro [97] and Oryx [98] built on JAX [99] could enable similar composability [101], with potential advantages from JAX's focus on efficiency and GPU scaling. However, the smaller JAX-specific ecosystem compared to general Python packages, JAX's optimisation for neural network tasks, and Python PPLs' more programmer-oriented syntax that diverges from mathematical notation may create barriers for epidemiological modellers. As well as imperative approaches like `Turing.jl`, where models are written as procedural code, declarative graph-based approaches such as `JuliaBUGS.jl` [102] and `RxInfer.jl` [103] specify models through explicit conditional dependencies between variables. This graph-based representation offers several advantages including more transparent model structure and assumptions, and efficient inference through algorithms that leverage the model's graphical structure. However, the declarative approach trades off the flexibility of `Turing.jl`'s imperative style for procedural code, such as loops over recursive updates for dynamical systems, which may limit the range of epidemiological models that can be composed. Building a domain-specific backend on `Distributions.jl` rather than `Turing.jl` could enable compatibility with multiple PPLs including `JuliaBUGS.jl` whilst trading off `Turing.jl`'s expressive submodel interface.

Areas for future work on our approach include gathering community feedback on the proposed requirements, and expanding the component library to address epidemiological applications across multiple scales and data types. More work is also needed to allow for modifying deeply nested model specifications without the user needing to be aware of the structure of the nested model. Exploring the alternative approaches discussed above also warrants further investigation. This includes integrating our approach with AlgebraicJulia tools for composing dynamical systems, extending category theory based software to support probabilistic models, using symbolic-numeric frameworks for automated optimisation, and exploring alternative PPL backends. Methodological advances are also needed for joint estimation of interdependent epidemiological parameters and integration of individual and population-level observations. Further developing and integrating modular Bayesian inference methods, including Markov melding [42] for combining submodels fitted separately and cut likelihoods [104] for controlling feedback between modules, could enable methodological sharing that is currently difficult. Better understanding where diversity in modelling assumptions is important in multi-model efforts, and separating such differences from implementation details, also requires further study. Future surveillance programmes could also be designed with composable modelling frameworks in mind, enabling estimates and model components to be shared and reused from initial data collection through to policy-relevant analyses. Compositional frameworks also create opportunities for large language model integration. Language models could serve as model construction agents, using a composable framework to construct epidemiological models from component libraries [51]. The explicit structure and validation tools of a composable framework should enable language models to reason about model design and propose structural adaptations more easily and with less room for error. Further work is needed to assess this.

Ecosystem developments would also support composable modelling and could reduce the need for domain-specific abstractions. `Turing.jl` could better support composability through improvements to submodel handling, for example by returning random variables from nested submodels without requiring explicit return statements, enabling flexible conditioning on submodels, and simplifying post-processing when variable names change due to prefixing. Other ecosystem improvements such as better numerical stability handling, debugging tools that preserve probabilistic model semantics, and generalised inference chaining at the PPL level would also benefit composable approaches. Improved tooling for the Bayesian workflow [18,19] would also strengthen the case for composable modelling in Julia. Improving inference method efficiency through GPU support and within-thread parallelisation would enable scaling composable models to larger problems. One sensible development

path would be to build a `Turing.jl` based composable version of the Bayesian Regression modelling package [28], an R package for flexible Bayesian regression modelling, which could provide a non-domain-specific testbed for composable components as well as being directly usable with our domain specific approach. Finally, ongoing work on compiling Julia applications to standalone executables could enable composable interfaces in other languages with fewer dependencies.

Composable infectious disease modelling offers a path to maintain statistical rigour whilst enabling the rapid model development needed to inform policy decisions. By enabling domain experts to contribute specialised components without understanding entire modelling frameworks, such approaches could accelerate integration of diverse expertise for both routine surveillance and outbreak response. Composable frameworks are also likely key for enabling robust large language model assisted model construction, where explicit component structure and validation tools could reduce errors. The proof of concept presented here indicates that this approach is feasible but substantial work remains to realise this potential. Given the unpredictable nature of future infectious disease threats, investment in adaptable modelling infrastructure that can incorporate diverse data sources and domain knowledge to provide timely evidence for policy is critical.

5.1 Acknowledgements

We thank the epiforecasts team for helpful comments and suggestions. We thank Poppy for growling as needed.

5.2 Data and software availability

All code and data for reproducing the analyses are available [64].

5.3 Funding information

We acknowledge the financial support from CDC Grant NU38FT00008 (K.J., S.A, S.F.). This project was made possible by cooperative agreement CDC-RFA-FT-23-0069 from the CDC's Center for Forecasting and Outbreak Analytics. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the Centers for Disease Control and Prevention. We acknowledge the financial support from Wellcome 210758/Z/18/Z (S.F.).

5.4 AI disclosure

Claude Sonnet 4.5 (https://www.anthropic.com/claude/sonnet) and Claude Opus 4.5 (https://www.anthropic.com/claude/opus) were used to assist with preparation of the manuscript text and as part of code review for the `EpiAware` and `EpiAwareR` packages and the analysis applying them to the case studies presented here.

References

1. Heesterbeek H, Anderson RM, Andreasen V, Bansal S, De Angelis D, Dye C, et al. Modeling infectious disease dynamics in the complex landscape of global health. Science. 2015;347. doi:10.1126/science.aaa4339

2. Birrell PJ, Blake J, Kandiah J, Alexopoulos A, Leeuwen E van, Pouwels KB, et al. Real-time modelling of the SARS-CoV-2 pandemic in England 2020–2023: A challenging data integration. J R Stat Soc Ser A Stat Soc. 2025; qnaf030. doi:10.1093/jrsssa/qnaf030

3. Office for National Statistics. COVID-19 Infection Survey. https://www.ons.gov.uk/surveys/informationforhouseholdsandindividuals/householdandindividualsurveys/covid19infectionsurvey; 2020.

4. Davies NG, Abbott S, Barnard RC, Jarvis CI, Kucharski AJ, Munday JD, et al. Estimated transmissibility and impact of SARS-CoV-2 lineage B.1.1.7 in england. Science. 2021;372. doi:10.1126/science.abg3055

5. Endo A, Murayama H, Abbott S, Ratnayake R, Pearson CAB, Edmunds WJ, et al. Heavy-tailed sexual contact networks and monkeypox epidemiology in the global outbreak, 2022. Science. 2022;378: 90–94. doi:10.1126/science.add4507

6. Hay JA, Kennedy-Shaffer L, Kanjilal S, Lennon NJ, Gabriel SB, Lipsitch M, et al. Estimating epidemiologic dynamics from cross-sectional viral load distributions. Science. 2021;373. doi:10.1126/science.abh0635

7. Anderson R, Donnelly C, Hollingsworth D, Keeling M, Vegvari C, Baggaley R, et al. Reproduction number (r) and growth rate (r) of the COVID-19 epidemic in the UK: Methods of estimation, data sources, causes of heterogeneity, and use as a guide in policy formulation. https://royalsociety.org/-/media/policy/projects/set-c/set-covid-19-R-estimates.pdf; 2020.

8. Brisson M, Kim JJ, Canfell K, Drolet M, Gingras G, Burger EA, et al. Impact of HPV vaccination and cervical screening on cervical cancer elimination: A comparative modelling analysis in 78 low-income and lower-middle-income countries. The Lancet. 2020;395: 575–590. doi:10.1016/s0140-6736(20)30068-4

9. Cramer EY, Huang Y, Wang Y, Ray EL, Cornell M, Bracher J, et al. The United States COVID-19 Forecast Hub dataset. Scientific Data. 2022;9: 462. doi:10.1038/s41597-022-01517-w

10. Reich NG, Lessler J, Funk S, Viboud C, Vespignani A, Tibshirani RJ, et al. Collaborative hubs: Making the most of predictive epidemic modeling. American Journal of Public Health. 2022;112: 839–842. doi:10.2105/ajph.2022.306831

11. Howerton E, Contamin L, Mullany LC, Qin M, Reich NG, Bents S, et al. Evaluation of the US COVID-19 Scenario Modeling Hub for informing pandemic response under uncertainty. Nature Communications. 2023;14: 7260. doi:10.1038/s41467-023-42680-x

12. Whitty CJM. What makes an academic paper useful for health policy? BMC Medicine. 2015;13. doi:10.1186/s12916-015-0544-8

13. Stan Development Team. Stan modeling language users guide and reference manual. 2025. Available: https://mc-stan.org

14. Murray LM. Bayesian state-space modelling on high-performance hardware using LibBi. 2013. Available: https://arxiv.org/abs/1306.3277

15. King AA, Nguyen D, Ionides EL. Statistical inference for partially observed Markov processes via the R package pomp. Journal of Statistical Software. 2016;69: 1–43. doi:10.18637/jss.v069.i12

16. FitzJohn R, Hinsley W, Knock E, Baguelin M. Monty: Monte carlo models. 2025. Available: https://github.com/mrc-ide/monty

17. Meent J-W van de, Paige B, Yang H, Wood F. An introduction to probabilistic programming. arXiv preprint arXiv:180910756. 2018. doi:10.48550/arXiv.1809.10756

18. Gelman A, Vehtari A, Simpson D, Margossian CC, Carpenter B, Yao Y, et al. Bayesian workflow. arXiv preprint arXiv:201101808. 2020. doi:10.48550/arXiv.2011.01808

19. Gelman A, Vehtari A, McElreath R. Statistical workflow. 2025.

20. Scott JA, Gandy A, Mishra S, Bhatt S, Flaxman S, Unwin HJT, et al. Epidemia: An r package for semi-mechanistic bayesian modelling of infectious diseases using point processes. 2021. Available: https://arxiv.org/abs/2110.12461

21. Abbott S, Hellewell J, Sherratt K, Gostic K, Hickson J, Badr HS, et al. EpiNow2: Estimate real-time case counts and time-varying epidemiological parameters. https://epiforecasts.io/EpiNow2/; 2020.

22. Sam Abbott, Lison A, Funk S, Pearson C, Gruson H, Guenther F, et al. Epinowcast: A Bayesian framework for real-time infectious disease surveillance. 2025. doi:10.5281/zenodo.5637165

23. Oxford Centre for Statistics in Machine Learning. Epimap: Renewal equation models for local COVID-19 dynamics. 2020. Available: https://github.com/oxcsml/epimap

24. BDI Pathogens. EpiLine: Estimating epi-curves and distributions from case line list data. 2022. Available: https://github.com/BDI-pathogens/EpiLine

25. Lison A, McLeod R, Huisman JS, Munday JD, Ort C, Julian TR, et al. Robust real-time estimation of pathogen transmission dynamics from wastewater. medRxiv. 2025. doi:10.1101/2025.10.23.25338640

26. Johnson KE, Vega Yon G, Brand SPC, Bernal Zelaya C, Bayer D, Volkov I, et al. Bayesian generative modeling for heterogeneous wastewater data applied to COVID-19 forecasting. 2025.

27. Adam Howes, Park SW, Sam Abbott. Epidist: Estimate epidemiological delay distributions with brms. 2024. doi:10.5281/zenodo.14213017

28. Bürkner P-C. brms: An R package for Bayesian multilevel models using Stan. Journal of Statistical Software. 2017;80: 1–28. doi:10.18637/jss.v080.i01

29. Huisman JS, Scire J, Angst DC, Li J, Neher RA, Maathuis MH, et al. Estimation and worldwide monitoring of the effective reproductive number of SARS-CoV-2. eLife. 2022;11. doi:10.7554/elife.71345

30. Liu Y, Morgenstern C, Kelly J, Lowe R, Jit M, CMMID COVID-19 Working Group. The impact of non-pharmaceutical interventions on SARS-CoV-2 transmission across 130 countries and territories. BMC Med. 2021;19: 40. doi:10.1186/s12916-020-01872-8

31. Günther F, Bender A, Katz K, Küchenhoff H, Höhle M. Nowcasting the COVID-19 pandemic in bavaria. Biom J. 2021;63: 490–502. doi:10.1002/bimj.202000112

32. Lison A, Abbott S, Huisman J, Stadler T. Generative bayesian modeling to nowcast the effective reproduction number from line list data with missing symptom onset dates. Britton T, editor. PLOS Computational Biology. 2024;20: e1012021. doi:10.1371/journal.pcbi.1012021

33. Abbott S, Funk S. Estimating the increase in reproduction number associated with the Delta variant using local area dynamics in England. medRxiv. 2022. doi:10.1101/2021.11.30.21267056

34. McCabe R, Danelian G, Panovska-Griffiths J, Donnelly CA. Inferring community transmission of SARS-CoV-2 in the United Kingdom using the ONS COVID-19 Infection Survey. Infectious Disease Modelling. 2024;9: 269–283. doi:10.1016/j.idm.2024.01.007

35. McCrone JT, Hill V, Bajaj S, Bernstein RE, Geoghegan JL, Pybus OG, et al. Context-specific emergence and growth of the SARS-CoV-2 Delta variant. Nature. 2022;610: 154–160. doi:10.1038/s41586-022-05200-3

36. Nyberg T, Ferguson NM, Nash SG, Webster HH, Flaxman S, Andrews N, et al. Comparative analysis of the risks of hospitalisation and death associated with SARS-CoV-2 omicron (B.1.1.529) and delta (B.1.617.2) variants in England: A cohort study. The Lancet. 2022;399: 1303–1312. doi:10.1016/S0140-6736(22)00462-7

37. Bezanson J, Edelman A, Karpinski S, Shah VB. Julia: A fresh approach to numerical computing. SIAM Review. 2017;59: 65–98. doi:10.1137/141000671

38. Nicholson G, Blangiardo M, Briers M, Diggle PJ, Fjelde TE, Ge H, et al. Interoperability of statistical models in pandemic preparedness: Principles and reality. Stat Sci. 2022;37. doi:10.1214/22-sts854

39. Fjelde TE, Xu K, Widmann D, Tarek M, Pfiffer C, Trapp M, et al. Turing.jl: A general-purpose probabilistic programming language. ACM Transactions on Probabilistic Machine Learning. 2025. doi:10.1145/3711897

40. Jing X, Yang X, Luo J, Zuo G. Code examples for the paper "a flexible, differentiable framework for neural-enhanced hydrological modeling: Design, implementation, and applications with HydroModels.jl". Zenodo; 2025. doi:10.5281/zenodo.15549719

41. Klöwer M, Gelbrecht M, Hotta D, Willmert J, Silvestri S, Wagner GL, et al. SpeedyWeather.jl: Reinventing atmospheric general circulation models towards interactivity and extensibility. Journal of Open Source Software. 2024;9: 6323. doi:10.21105/joss.06323

42. Goudie RJB, Presanis AM, Lunn D, De Angelis D, Wernisch L. Joining and splitting models with Markov melding. Bayesian Analysis. 2019;14: 81–109. doi:10.1214/18-BA1104

43. Julia Language. Conditional loading of code in packages (extensions). https://pkgdocs.julialang.org/v1/creating-packages/#Conditional-loading-of-code-in-packages-(Extensions); 2024. Available: https://pkgdocs.julialang.org/v1/creating-packages/#Conditional-loading-of-code-in-packages-(Extensions)

44. Mishra S, Scott JA, Harrison E, Zhu H, Ferguson NM, Bhatt S. A COVID-19 transmission model with time-varying transmission rate. arXiv preprint arXiv:200616487. 2020. doi:10.48550/arXiv.2006.16487

45. Chatzilena A, Zyl G van, Manson AL, Earl AM, Jamieson FB, Joloba ML, et al. Contemporary tuberculosis transmission in Lagos, Nigeria: Spatial patterns and model calibration using genomic epidemiology. Epidemics. 2019;29: 100363. doi:10.1016/j.epidem.2019.100363

46. Swallow B, Birrell P, Blake J, Burgman M, Challenor P, Coffeng LE, et al. Challenges in estimation, uncertainty quantification and elicitation for pandemic modelling. Epidemics. 2022;38: 100547. doi:10.1016/j.epidem.2022.100547

47. Baydin AG, Pearlmutter BA, Radul AA, Siskind JM. Automatic differentiation in machine learning: A survey. Journal of Machine Learning Research. 2018;18: 1–43.

48. Hoffman MD, Gelman A. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. Journal of Machine Learning Research. 2014;15: 1593–1623. Available: https://www.jmlr.org/papers/volume15/hoffman14a/hoffman14a.pdf

49. Gelman A, Hill J. Data analysis using regression and multilevel/hierarchical models. Cambridge: Cambridge University Press; 2007.

50. Philipps M, Schmid N, Hasenauer J. Current state and open problems in universal differential equations for systems biology. npj Systems Biology and Applications. 2025;11: 101. doi:10.1038/s41540-025-00550-w

51. Aygün E, Belyaeva A, Comanici G, Coram M, Cui H, Garrison J, et al. An AI system to help scientists write expert-level empirical software. arXiv preprint arXiv:250906503. 2025. doi:10.48550/arXiv.2509.06503

52. SciML: Open source software for scientific machine learning. https://sciml.ai/; 2024. Available: https://sciml.ai/

53. Innes M. Flux: Elegant machine learning with julia. Journal of Open Source Software. 2018;3: 602. doi:10.21105/joss.00602

54. Funk S. EpiAwareR: R interface to the EpiAware compositional infectious disease modelling framework. 2025. Available: https://github.com/sbfnk/EpiAwareR

55. Li C. JuliaCall: Seamless integration between R and Julia. 2024. Available: https://CRAN.R-project.org/package=JuliaCall

56. Simeonov J, Carlsson K, Contributors. Accessors.jl: Updating immutable data simple. https://github.com/JuliaObjects/Accessors.jl; 2024. Available: https://juliaobjects.github.io/Accessors.jl/stable/

57. Mishra S, Berah T, Mellan TA, Unwin HJT, Vollmer MA, Parag KV, et al. On the derivation of the renewal equation from an age-dependent branching process: an epidemic modelling perspective.

58. Besançon M, Papamarkou T, Anthoff D, Arslan A, Byrne S, Lin D, et al. Distributions.jl: Definition and modeling of probability distributions in the JuliaStats ecosystem. Journal of Statistical Software. 2021;98: 1–30. doi:10.18637/jss.v098.i16

59. Tarek M, Xu K, Trapp M, Ge H, Ghahramani Z. DynamicPPL: Stan-like speed for dynamic probabilistic models. arXiv preprint arXiv:200202702. 2020. Available: https://arxiv.org/abs/2002.02702

60. EpiAware Team. CensoredDistributions.jl: Censored distributions for Julia. 2025. Available: https://github.com/EpiAware/CensoredDistributions.jl

61. Thompson W. PairPlots.jl: Beautiful and flexible visualizations of high dimensional data. 2023. Available: https://github.com/sefffal/PairPlots.jl

62. Danisch S, Krumbiegel J. Makie.jl: Flexible high-performance data visualization for Julia. Journal of Open Source Software. 2021;6: 3349. doi:10.21105/joss.03349

63. Abbott S, Hellewell J, Thompson R, Sherratt K, Gibbs H, Bosse N, et al. Estimating the time-varying reproduction number of SARS-CoV-2 using national and subnational case counts [version 2; peer review: 1 approved, 1 approved with reservations]. Wellcome Open Research. 2020;5. doi:10.12688/wellcomeopenres.16006.2

64. Abbott S, Brand SPC, Ge H, Johnson KE, Cori A, Funk S. Code and data for: Composable probabilistic models can lower barriers to rigorous infectious disease modelling. 2025. doi:10.5281/zenodo.17884675

65. Cori A, Ferguson NM, Fraser C, Cauchemez S. A new framework and software to estimate time-varying reproduction numbers during epidemics. American journal of epidemiology. 2013;178: 1505–1512.

66. Wallinga J, Lipsitch M. How generation intervals shape the relationship between growth rates and reproductive numbers. Proceedings of the Royal Society B: Biological Sciences. 2007;274: 599–604.

67. Zhang L, Carpenter B, Gelman A, Vehtari A. Pathfinder: Parallel quasi-newton variational inference. Journal of Machine Learning Research. 2022;23: 1–49.

68. Charniga K, Park SW, Akhmetzhanov AR, Cori A, Dushoff J, Funk S, et al. Best practices for estimating and reporting epidemiological delay distributions of infectious diseases. PLoS computational biology. 2024;20: e1012520.

69. Park SW, Sun K, Abbott S, Sender R, Bar-On YM, Weitz JS, et al. Inferring the differences in incubation-period and generation-interval distributions of the delta and omicron variants of SARS-CoV-2. Proceedings of the National Academy of Sciences. 2023;120: e2221887120.

70. Zhao S, Lin Q, Ran J, Musa SS, Yang G, Wang W, et al. Preliminary estimation of the basic reproduction number of novel coronavirus (2019-nCoV) in china, from 2019 to 2020: A data-driven analysis in the early phase of the outbreak. International journal of infectious diseases. 2020;92: 214–217.

71. Rackauckas C, Nie Q. DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. The Journal of Open Research Software. 2017;5: 15. doi:10.5334/jors.151

72. Reich NG, Brooks LC, Fox SJ, Kandula S, McGowan CJ, Moore E, et al. A collaborative multiyear, multimodel assessment of seasonal influenza forecasting in the United States. Proc Natl Acad Sci U S A. 2019;116: 3146–3154. doi:10.1073/pnas.1812594116

73. Cramer EY, Huang Y, Wang Y, Ray EL, Cornell M, Bracher J, et al. The United States COVID-19 forecast hub dataset. medRxiv. 2021. doi:10.1101/2021.11.04.21265886

74. Centers for Disease Control and Prevention. Estimates of rt for COVID-19 and influenza. 2025. Available: https://www.cdc.gov/cfa-modeling-and-forecasting/rt-estimates/index.html

75. Brooks-Pollock E, Sherrat K, Sherrat K, Sherrat K, Sherrat K, CMMID COVID-19 Working Group. Modelling that shaped the early COVID-19 pandemic response in the UK. Philosophical Transactions of the Royal Society B. 2021;376: 20210001. doi:10.1098/rstb.2021.0001

76. Brockhaus EK, Wolffram D, Stadler T, Osthege M, Mitra T, Littek JM, et al. Why are different estimates of the effective reproductive number so different? A case study on COVID-19 in germany. PLoS Comput Biol. 2023;19: e1011653. doi:10.1371/journal.pcbi.1011653

77. Moses WS, Churavy V. Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients. Advances in neural information processing systems. 2020. pp. 12472–12485. Available: https://arxiv.org/abs/2010.01709

78. Dalle G, Hill A. A common interface for automatic differentiation. 2025. Available: https://arxiv.org/abs/2505.05542

79. Rowley C. PythonCall.jl: Python and Julia in harmony. 2022. Available: https://github.com/JuliaPy/PythonCall.jl

80. Fong B, Spivak DI. Seven sketches in compositionality: An invitation to applied category theory. Cambridge: Cambridge University Press; 2019. doi:10.1017/9781108668804

81. Libkind S, Baas A, Halter M, Patterson E, Fairbanks JP. An algebraic framework for structured epidemic modelling. Philosophical Transactions of the Royal Society A. 2022;380: 20210309. doi:10.1098/rsta.2021.0309

82. Libkind S, Baas A, Patterson E, Fairbanks J. Operadic modeling of dynamical systems: Mathematics and computation. Proceedings of the fourth international conference on applied category theory (ACT 2021). 2021. pp. 192–206. doi:10.4204/EPTCS.372.14

83. Ma Y, Gowda S, Anantharaman R, Laughman C, Shah V, Rackauckas C. ModelingToolkit: A composable graph transformation system for equation-based modeling. 2021. Available: https://arxiv.org/abs/2103.05244

84. Loman YAI Torkel E. AND Ma. Catalyst: Fast and flexible modeling of reaction networks. PLOS Computational Biology. 2023;19: e1011530. doi:10.1371/journal.pcbi.1011530

85. Baez JC, Pollard BS. A compositional framework for reaction networks. Reviews in Mathematical Physics. 2017;29: 1750028.

86. Kerr C, Stuart R, Abeysuriya R, Cohen J, Sanz-Leon P, Muellenmeister A, et al. Starsim: A flexible framework for agent-based modeling of health and disease. Proceedings of the 23rd python in science conference (SciPy 2024). 2024. doi:10.25080/ukpu4584

87. Gallagher K, Bouros I, Fan N, Hayman E, Heirene L, Lamirande P, et al. Epidemiological agent-based modelling software (Epiabm). Journal of Open Research Software. 2024;12. doi:10.5334/jors.449

88. Chopra A, Rodríguez A, Subramanian J, Quera-Bofarull A, Krishnamurthy B, Prakash BA, et al. Differentiable agent-based epidemiology. International conference on autonomous agents and multiagent systems. 2023.

89. Datseris G, Vahdati AR, DuBois TC. Agents.jl: A performant and feature-full agent-based modeling software of minimal code complexity. Simulation. 2022;98: 267–279. doi:10.1177/00375497211068820

90. Bostanci I, Conrad T. Integrating agent-based and compartmental models for infectious disease modeling: A novel hybrid approach. Journal of Artificial Societies and Social Simulation. 2025;28: 5. doi:10.18564/jasss.5567

91. Overton CE, Abbott S, Christie R, Cumming F, Day J, Jones O, et al. Nowcasting the 2022 mpox outbreak in England. PLoS Comput Biol. 2023;19: e1011463. doi:10.1371/journal.pcbi.1011463

92. FitzJohn R, Contributors. Odin: ODE generation and solving. 2025. Available: https://github.com/mrc-ide/odin

93. Golding N, Contributors. Greta: Simple and scalable statistical modelling in R. 2025. Available: https://github.com/greta-dev/greta

94. Cusumano-Towner MF, Saad FA, Lew AK, Mansinghka VK. Gen: A general-purpose probabilistic programming system with programmable inference. Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation. ACM; 2019. pp. 221–236. doi:10.1145/3314221.3314642

95. Saad FA, Patton BJ, Hoffmann MD, Saurous RA, Mansinghka VK. Sequential monte carlo learning for time series structure discovery. Proceedings of the 40th international conference on machine learning. PMLR; 2023. pp. 29473–29489. doi:10.48550/arXiv.2307.09607

96. Tan Z-X, Becker MR, Mansinghka VK. Genify.jl: Transforming Julia into Gen to enable programmable inference. Workshop on languages for inference (LAFI, co-located with POPL 2021). 2021. Available: https://popl21.sigplan.org/details/lafi-2021-papers/5/Genify-jl-Transforming-Julia-into-Gen-to-enable-programmable-inference

97. Phan D, Pradhan N, Jankowiak M. Composable effects for flexible and accelerated probabilistic programming in NumPyro. arXiv preprint arXiv:191211554. 2019. doi:10.48550/arXiv.1912.11554

98. JAX-ML Team. Oryx: Probabilistic programming and deep learning in JAX. https://github.com/jax-ml/oryx; 2020.

99. Bradbury J, Frostig R, Hawkins P, Johnson MJ, Leary C, Maclaurin D, et al. JAX: Composable transformations of Python+NumPy programs. 2018. Available: http://github.com/jax-ml/jax

100. Center for Forecasting and Outbreak Analytics, US Centers for Disease Control and Prevention. PyRenew: A package for Bayesian renewal modeling with JAX and NumPyro. https://github.com/CDCgov/PyRenew; 2024.

101. JAX-ML Team. Coix: Composable inference for JAX. https://github.com/jax-ml/coix; 2024.

102. Sun X, Gabler P, Thomas A, Ge H. JuliaBUGS.jl: A graph-based probabilistic programming language using BUGS syntax. Presented at Workshop on Languages for Inference (LAFI, co-located with POPL 2024); 2024. Available: https://github.com/TuringLang/JuliaBUGS.jl

103. Bagaev D, Podusenko A, De Vries B. RxInfer: A julia package for reactive real-time bayesian inference. Journal of Open Source Software. 2023;8: 5161.

104. Plummer M. Cuts in Bayesian graphical models. Statistics and Computing. 2015;25: 37–43. doi:10.1007/s11222-014-9503-z