Manuscript BI

wordcount : 3234 ### Instructions Applications are short descriptions (up to 4000 words total, including tables/figure captions, statements and references list) of new software scripts or packages. Applications can serve as a citable entity for software developers, should provide more of a description than a user manual or vignette (although we strongly encourage the authors to provide a vignette for their code to facilitate the uptake of their methods and allow its intended use), and should include details of at least one worked example. Uploading a package to a site such as CRAN or sourceforge in advance is not considered prior publication and will not hinder your submission to MEE, but if the package has been uploaded to a public repository, please ensure that links to it are suitably anonymised in your manuscript. See examples here.

Abstract

1. Bayesian modeling is a powerful paradigm in modern statistics and machine learning, offering a principled framework for inference under uncertainty. However, practitioners face significant obstacles, including **interoperability** issues, a persistent **accessibility-flexibility trade-off**, the limitations of **domain-specific limitations**, and challenges in **scalability**.

2. ***Interoperability:*** The landscape of Bayesian software is fragmented across programming languages and abstraction levels. Newcomers often gravitate towards high-level interfaces (e.g., *brms*) within familiar environments due to their accessibility for standard models. However, these tools can be restrictive, lacking the flexibility needed for custom or complex models as research needs evolve.

3. ***Accessibility-Flexibility Trade-off:*** However, highgh levels of abstraction frameworks can be restrictive, lacking the flexibility needed for custom or complex models as research needs evolve.To gain the necessary flexibility, researchers must often transition to lower-level probabilistic programming languages (PPLs) like *Stan* (requiring its specific DSL), *PyMC*, *NumPyro*, or *TensorFlow Probability*. This transition imposes a steeper learning curve and requires mastery of specific modeling languages or complex programming frameworks, hindering broader adoption and rapid iteration.

4. ***Domain-Specific Limitations:*** Similar accessibility and flexibility trade-offs exist in domain-specific Bayesian packages (e.g., *STRAND*, *BISON*, *BEAST*, *RevBayes*). While providing accessible, pre-packaged models for specific fields, customizing or extending these models often requires deep engagement with lower-level programming languages or switching tools entirely, limiting methodological innovation within those domains.

5. ***Scalability:*** Computational demands remain a significant bottleneck, limiting the application of Bayesian methods to the large datasets and complex, high-dimensional models prevalent in modern research. While established tools like *Stan* offer highly optimized inference algorithms, they can face challenges with model compilation times and often require substantial code restructuring to leverage hardware acceleration (e.g., GPUs, TPUs) effectively.

6. To address these challenges, we introduce ***Bayesian Inference (BI)***, a new Bayesian modeling software available in both Python and R. It aims to unify the modeling experience by integrating an intuitive model-building syntax (enhancing **accessibility**) with the **flexibility** of multiple, interchangeable inference backends, including hardware-accelerated computation via JAX for improved **scalability**. Its availability in both major data science languages directly tackles the **interoperability** barrier and the pre-build function for specialized model in network analysis, survial models and phylogenetic analysis allow to improved **domain-specific limitations**.

7. By providing a streamlined and efficient environment for the end-to-end Bayesian workflow—from model specification and fitting to diagnostics and prediction, ***BI*** lowers the barrier to entry for sophisticated Bayesian modeling. We aim to empower a broader community of researchers across disciplines to confidently apply advanced Bayesian methods to their complex research problems.

Introduction

Bayesian modeling has emerged as a vital tool in modern statistics and machine learning, providing a framework for robust inference under uncertainty and the possibility to integrate prior knowledge. Despite its potential, the practical application of Bayesian methods is often hindered by significant hurdles within the current software ecosystem, preventing researchers from fully leveraging its capabilities. Key challenges stem from the fragmented nature of software across different programming languages (**interoperability**), gaps between theoretical understanding and practical implementation (**accessibility**), complexities in model specification that force trade-offs between ease-of-use and flexibility (**accessibility-flexibility trade-off**), the constraints of overly specialized tools (**domain-specific limitations**), and persistent computational scalability limitations for complex models or large datasets (**scalability**).

The first major obstacle is the fragmented landscape of Bayesian software, scattered across different programming languages and varying levels of abstraction, posing significant **interoperability** challenges. Researchers frequently encounter a disparate collection of tools—from

Stan's domain-specific language (DSL) to distinct low-level of abstraction libraries (like *PyMC, TensorFlow Probability (TFP), NumPyro*) and high-level of abstraction libraries (like *BRMS*). This fragmentation complicates workflows and presents a confusing landscape, especially for researchers new to Bayesian analysis. For instance, researchers new to Bayesian analysis may initially gravitate towards tools of high-level of abstraction available within their most familiar programming environment (e.g., *BRMS* in *R*), potentially overlooking more suitable options elsewhere due to the steep initial learning curve or perceived incompatibility (accessibility). This linguistic and platform diversity imposes considerable cognitive overhead, potentially hindering the adoption of the most suitable tool for a given problem due to familiarity biases or the friction of switching ecosystems, ultimately impacting the effective application of Bayesian methods. This initial hurdle of navigating disparate systems naturally leads new practitioners to prioritize tools that appear easiest to learn, raising concerns about the balance between accessibility and the flexibility needed for complex research.

Compounding this fragmentation is the challenge of accessibility and the translation of theoretical knowledge into practice **accessibility-flexibility trade-off**. Indeed, while high-level interfaces like *BRMS* offer an intuitive formula-based syntax, significantly lowering the initial barrier to entry (e.g. generalized linear mixed models using *BRMS*), this accessibility often comes at the cost of flexibility. As research questions become more sophisticated, requiring custom likelihood functions (e.g., multiple likelihoods), intricate prior structures (e.g., XXX), or non-standard model components (e.g., centered-random factors), the limitations of these high-level wrappers become apparent. To gain the necessary expressive power, the researcher must typically transition to lower-level probabilistic programming languages (PPLs) such as *Stan* (requiring mastery of its specific DSL), *PyMC*, *NumPyro*, or *TFP*. This transition imposes a much steeper learning curve, demanding a deeper understanding of probabilistic programming concepts (like computational graphs or tensor manipulation) and often more verbose code. This significant jump in complexity can deter users, divert focus from statistical modeling to software engineering challenges, and ultimately slow down the pace of research, particularly when trying to adapt models within specific scientific fields.

Similar accessibility and flexibility constraints manifest as **domain-specific limitations** within specialized Bayesian packages. Fields like phylogenetics or network analysis benefit from tools such as *BEAST*, *RevBayes*, *STRAND*, or *BISON*, which provide accessible, pre-packaged models tailored to common domain problems. A phylogeneticist might initially find *BEAST* convenient for standard molecular clock models. However, when they wish to incorporate a novel evolutionary hypothesis requiring modification of the core model structure or integrate data types not originally envisioned by the developers, they often encounter rigid constraints. Extending these specialized tools frequently requires deep engagement with their underlying, often complex, codebase (sometimes necessitating proficiency in languages like Java or C++) or abandoning the domain-specific tool entirely in favor of a general-purpose PPL. This forces researchers to either compromise on their methodological innovation or undertake a significant software development effort, potentially switching programming ecosystems and losing the initial convenience, thereby limiting the evolution of modeling practices within specialized domains. Even when model specification is achievable, either in general or specialized

tools, the computational feasibility remains a major concern.

Finally, computational **scalability** continues to be a significant bottleneck, limiting the application of Bayesian methods to the large datasets (e.g., millions of observations) and complex, high-dimensional models (e.g., thousands of parameters) prevalent in modern research across fields like genomics, neuroscience, and machine learning. While established tools like *Stan* feature highly optimized inference algorithms (particularly its NUTS sampler) and offer effective multi-core parallelization, they can still face challenges with long C++ compilation times for complex models and may require substantial code restructuring or external tooling to efficiently leverage hardware accelerators like GPUs or TPUs for certain computations. Conversely, emerging frameworks built on *JAX* (powering *NumPyro* and parts of *TFP*) promise substantial speedups via automatic differentiation, JIT compilation, and native support for parallel hardware architectures. However, integrating these powerful backends seamlessly into user-friendly, flexible modeling front-ends that don't require deep expertise in the JAX ecosystem itself is an ongoing challenge. Domain-specific tools often inherit the scalability limitations of the frameworks they are built upon, failing to provide a universally efficient solution across different model types and data sizes.

Therefore, there is an evident and pressing demand for a Bayesian modeling framework that synergistically addresses these interconnected limitations. To address these interconnected challenges, we introduce ***Bayesian Inference (BI)***, a new Bayesian modeling software designed to unify the modeling experience across the two dominant data science languages, Python and R. *BI* tackles the **interoperability** barrier head-on by offering native interfaces in both environments. It aims to resolve the **accessibility-flexibility trade-off** by providing an intuitive model-building syntax familiar to users of statistical modeling languages, while enabling advanced customization and leveraging multiple, interchangeable inference backends for flexibility. To combat **domain-specific limitations**, *BI* includes pre-built functions and structures tailored for specialized models in areas like network analysis, survival analysis, and phylogenetic analysis, while still allowing extension and modification within its general framework. Crucially, *BI* enhances **scalability** by integrating with hardware-accelerated computation via *JAX* (using *NumPyro* or *TFP* as backends), enabling efficient execution on CPUs, GPUs, and TPUs. By providing a streamlined, efficient, and unified environment for the end-to-end Bayesian workflow—from model specification and fitting to diagnostics and prediction—***BI*** lowers the barrier to entry for sophisticated Bayesian modeling, aiming to empower a broader community of researchers across disciplines to confidently apply advanced Bayesian methods to their complex research problems.

Software Presentation

Bridging Ecosystems: Enhancing Interoperability

Recognizing the significant bifurcation of the data science community between the R and Python ecosystems, *BI* directly confronts the **interoperability** challenge by offering

native, feature-equivalent implementations in both languages. While minor syntactic differences necessarily exist to adhere to the idiomatic conventions of each language (e.g.,Python class object use leadin to function calls to be performed through dot call (e.g. `bi.dist.normal(0,1)`) and R class 5 object use lead function calls to be performed through dollar call (e.g. `bi$dist$normal(0,1)`), the core model specification syntax, the procedural workflow for analysis, and the underlying computational engines remain fundamentally consistent. This dual-language availability significantly lowers the barrier for researchers to adopt *BI*, allowing them to work entirely within their preferred and most familiar programming environment without sacrificing access to a common, powerful Bayesian modeling framework. Furthermore, this design facilitates smoother collaboration between research teams or individuals who may primarily use different languages, reducing the friction often associated with translating models, sharing code, or reproducing results across distinct software ecosystems. It aims to create a shared methodological platform, irrespective of the user's primary language preference.

Balancing Ease-of-Use and Expressive Power: Resolving the Accessibility-Flexibility Trade-off

BI is designed to navigate the critical **accessibility-flexibility trade-off** by providing multiple layers of abstraction and utility, catering effectively to users with varying levels of Bayesian modeling expertise and diverse complexity requirements.

Simplified Backend Interaction via Intuitive Syntax: At its computational core, *BI* leverages the power and efficiency of established Probabilistic Programming Languages (PPLs) like *NumPyro* and *TensorFlow Probability (TFP)*, both of which are built upon the *JAX* framework for high-performance numerical computation and automatic differentiation. However, *BI* deliberately abstracts away much of the inherent complexity of these lower-level tools. This high-level interface handles the translation of the model specification into the underlying computational graph required by the backend (*NumPyro* or *TFP*), explicit probabilistic graph construction, managing *JAX* transformations (like `jit`, `vmap`, `grad`), and implementing sampling algorithms for standard to moderately complex models. This significantly enhances **accessibility** for a broader range of users.

Listing 1 `Differences in prior specification between Numpyro, TFP and BI`

```
numpyro.sample("mu", dist.Normal(0, 1)).expand([10])

yield Root(tfd.Sample(tfd.Normal(loc=1.0, scale=1.0), sample_shape=10))

bi.dist.normal("mu", 0, 1, shape = (10,))
```

Pre-built Components for Complex Model Features: To enhance **flexibility** without unduly sacrificing the accessibility provided by the high-level syntax, *BI* includes a library

of pre-built, computationally optimized functions implemented directly in *JAX*. These components encapsulate common but potentially complex modeling structures, allowing users to incorporate them easily within the standard formula interface. Key examples include: * `CenteredRandomEffects()` / `NonCenteredRandomEffects()`: Facilitates the specification of hierarchical (multi-level) model components, offering both centered and non-centered parameterizations. The non-centered version, often crucial for efficient sampling in hierarchical models (particularly with sparse data), is provided without requiring the user to manually implement the reparameterization logic. * `GaussianProcess()`: Provides a straightforward interface for incorporating Gaussian processes to model spatial, temporal, phylogenetic, or other forms of structured correlation or dependency. This component handles the complexities of kernel definition (e.g., RBF, Matérn), hyperparameter priors, and efficient computation of the GP covariance matrix and likelihood contributions internally. * `Spline()` / `BasisExpansion()`: Allows for the inclusion of flexible non-linear relationships using spline bases (e.g., B-splines, thin-plate splines) directly within the model formula. These components can be seamlessly integrated into the model specification, allowing users to build sophisticated models declaratively.

Integrated End-to-End Workflow: *BI* is designed to encapsulate the entire Bayesian modeling workflow within a cohesive object-oriented structure, promoting a streamlined and reproducible analysis pipeline. Typically, a user interacts with a primary `BI` object, through which they can sequentially:

- **Handle Data:** Load, preprocess, and associate dataset(s) with the model object.
- **Define Model:** Specify the model structure, including the likelihood(s), priors for all parameters (with sensibledefaults often available), and incorporate any pre-built components using the intuitive formula syntax.
- **Specify Inference:** Select the desired inference engine (e.g., NUTS via *NumPyro* or *TFP*, potentially VariationalInference methods) and configure its parameters (e.g., number of chains, warmup iterations, target acceptance rate).Crucially, the choice of computational backend (*NumPyro*, *TFP*) can often be switched with minimal changes to the modelspecification code.
- **Run Inference:** Execute the model fitting process, which triggers the backend PPL to perform sampling or optimization.Progress indicators and diagnostics are typically provided.
- **Analyze Posterior:** Access, summarize, and diagnose the posterior distributions of parameters. This includes methodsfor calculating posterior means, medians, credible intervals, convergence diagnostics (e.g., $\hat{R}$, Effective Sample Size- ESS), and retrieving raw posterior samples for custom analysis.
- **Generate Predictions & Check Fit:** Compute posterior predictions for the observed data (for posterior predictivechecks) or for new, unseen data points. Methods for evaluating model fit (e.g., WAIC, visual checks) are integrated.

- **Visualize Results:** Generate standard diagnostic plots (e.g., trace plots, rank plots, posterior distributions) andvisualizations of model parameters, effects, and predictions using integrated plotting functions that leverage commonlibraries (e.g., `ggplot2` in R, `matplotlib/seaborn/arviz` in Python). This unified structure minimizes the need forusers to juggle multiple disparate software tools or manually transfer data and results between different stages of theanalysis, thereby enhancing efficiency and reproducibility.

Extensive Model Library and Documentation: *BI* ships with 22 well-documented implementations of various standard and advanced Bayesian models. This includes Generalized Linear Models (GLMs), Generalized Linear Mixed Models (GLMMs), survival analysis models (e.g., Cox proportional hazards), and models relevant to phylogenetic comparative methods. Each implementation is accompanied by detailed documentation that encompasses 1) general principles, 2) underlying assumptions, 3) code snippets, and 4) mathematical details, enabling users to gain a deeper understanding of the modeling process and its nuances. Whether users are interested in hierarchical models, time-series analysis, or cutting-edge network modeling approaches, our library caters to a variety of analytical needs. This accessibility fosters an environment where users can confidently explore and implement Bayesian methods, ultimately enhancing their research capabilities. This curated library serves not only as a collection of ready-to-use tools but also as a valuable pedagogical resource, demonstrating best practices for constructing, fitting, and interpreting models within the *BI* framework, and providing robust templates for users aiming to develop novel model variants.

Table 1: Table of documented models.

Documented Models
Linear Regression for continuous variable
Multiple continuous Variables.qmd
Interaction between continuous variables
Categorical variable
Binomial model
Beta binomial model
Poisson model
Gamma-Poisson
Multinomial model
Dirichlet model
Zero inflated
Varying intercepts
Varying slopes
Gaussian processes
Measuring error
Missing data
Network model
Sender receiver network model

Documented Models
Block model network
Network Computations
Network Based diffusion approach

For example, the state-of-the-art network models implemented in BI include methods for censoring and exposure control, block modeling, multiplex and multilayer networks, network-based diffusion approaches, and computation of network metrics. Each approach is presented in detail in the documentation, complete with their custom functions. This comprehensive documentation ensures that users can easily navigate the software and identify the models that best suit their research requirements. Furthermore, these diverse approaches can be combined to create a single model, such as a multiplex model with controls for censoring and exposure biases, along with the computation of social network measures (code snippets bellow). This versatility showcases the flexibility and power of the software, allowing users to tailor their analyses to meet specific research needs, which is not possible with other specialized Bayesian network frameworks such as STRAND and BISON. These frameworks feature prebuilt models that either do not allow for direct computation of network measures within the model (BISON requires two separate models) or do not support network measure computation at all (STRAND).

Addressing Domain-Specific Limitations within a General Framework

While general-purpose PPLs offer maximal flexibility, they often lack the specialized functions and structures needed for efficient and intuitive implementation of models common in specific scientific domains. Conversely, dedicated domain-specific tools often suffer from restrictive assumptions or limited extensibility. *BI* seeks to bridge this gap by incorporating specialized, pre-built *JAX* functions and model structures tailored for common patterns in specific fields, yet keeping them fully integrated within the general, extensible modeling framework:

- **Network Analysis:** Includes components like `BlockModelEffects` for stochastic block models, `DyadicEffects` for modeling pairwise interactions while accounting for sender/receiver effects, and potentially functions like `NetworkCovariate` to directly incorporate network topology metrics (e.g., centrality, degree) as predictors or within the model structure (e.g., network-correlated errors).
- **Phylogenetics:** Provides pre-defined structures for common models of trait evolution on phylogenies leveraging on the `GaussianProcess()` component with phylogeny-specific covariance structures derived from the tree topology and branch lengths.
- **Survival Analysis:** Offers specialized likelihood functions and incorporating time-varying covariates or effects, common requirements in survival and reliability analysis.

By providing these optimized building blocks within its general syntax, *BI* allows researchers in these fields to rapidly implement standard domain models using familiar concepts. Crucially,

however, they retain the full flexibility of the *BI* framework to combine these domain-specific components with other model features (e.g., complex non-linear effects via splines, hierarchical structures across groups of networks or phylogenies) or to customize or extend them using *BI*'s underlying mechanisms if needed—a capability often missing in more narrowly focused domain-specific packages. This design aims to foster methodological innovation *within* specialized domains by lowering the barrier to implementing more complex or novel models.

Inheriting Scalability through Modern Backends A fundamental design principle of *BI* is to directly address the **scalability** challenge by building upon modern, high-performance computational backends. By utilizing *JAX*-based libraries like *NumPyro* and *TFP*, *BI* inherits their significant computational efficiencies, which are critical for tackling the large datasets and complex models prevalent in contemporary research like **Just-In-Time (JIT) Compilation:**, **Hardware Acceleration**. This strategic reliance on the *JAX* ecosystem allows *BI* users to tackle computationally intensive problems that might be infeasible or prohibitively slow using frameworks lacking comparable optimization and hardware acceleration capabilities. By abstracting the backend complexities while retaining their power, *BI* significantly enhances the practical **scalability** of sophisticated Bayesian inference for a wider audience.

In the subsequent section, we will demonstrate through concrete examples how these design features of *BI* coalesce to provide a streamlined, flexible, and powerful solution, effectively addressing the limitations identified in the existing Bayesian software landscape.

Discussion

The XXX framework is built on top of the popular Python programming language, with a focus on providing a user-friendly interface for model development and interpretation. Our framework is designed to be modular and extensible, allowing users to easily incorporate their own custom models and data types into the framework. We have also developed a set of tutorials and examples to help users get started with the framework and to demonstrate its capabilities.

One of the key features of this software is its comprehensive library of 16 predefined Bayesian models, covering a wide range of common applications and use cases. These models are accompanied by detailed explanations, making it easier for users to understand the underlying assumptions and apply the models to their specific research questions. In addition to these built-in models, the software includes several custom functions tailored for advanced statistical and network modeling. These functions support specialized tasks such as centered random effects, block modeling, and the computation of network measures. ## Aknowledgements ## References

Listing 2 Differences in random effect specification between Numpyro, TFP and BI

```
# Numpyro version of rendom centered effect
    a = numpyro.sample("a", dist.Normal(5, 2))
        b = numpyro.sample("b", dist.Normal(-1, 0.5))
        sigma_cafe = numpyro.sample("sigma_cafe", dist.Exponential(1).expand([2]))
        sigma = numpyro.sample("sigma", dist.Exponential(1))
        Rho = numpyro.sample("Rho", dist.LKJ(2, 2))
        cov = jnp.outer(sigma_cafe, sigma_cafe) * Rho
        a_cafe_b_cafe = numpyro.sample(
            "a_cafe,b_cafe", dist.MultivariateNormal(jnp.stack([a, b]), cov).expand([20])
        )
    a_cafe, b_cafe = a_cafe_b_cafe[:, 0], a_cafe_b_cafe[:, 1]

# TFP version of rendom centered effect
    alpha = yield Root(tfd.Sample(tfd.Normal(loc=5.0, scale=2.0), sample_shape=1))
    beta = yield Root(tfd.Sample(tfd.Normal(loc=-1.0, scale=0.5), sample_shape=1))

    sigma = yield Root(tfd.Sample(tfd.Exponential(rate=1.0), sample_shape=1))
    sigma_alpha_beta = yield Root(
        tfd.Sample(tfd.Exponential(rate=1.0), sample_shape=2)
    )

    Rho = yield Root(tfd.LKJ(dimension=2, concentration=2.0))
    Mu = tf.concat([alpha, beta], axis=-1)
    scale = tf.linalg.LinearOperatorDiag(sigma_alpha_beta).matmul(tf.squeeze(Rho))

# BI version of rendom centered effect
    Sigma_individual = bi.exponential('Sigma_individual', [ni], 1 )
    L_individual = bi.lkjcholesky('L_individual', [], ni, 1)
    z_individual = bi.normal('z_individual', [ni,K], 0, 1)
    alpha = random_centered2(Sigma_individual, L_individual, z_individual)
```