Reactive Probabilistic Programming for Scalable Bayesian Inference

Dmitry Bagaev

Reactive Probabilistic Programming for Scalable Bayesian Inference

Dmitry Bagaev

Probabilistic Programming?

Probabilistic programming

Why not just programming?

Feed some data

velocity = 1.1

position = 0.4

windspeed = 8.3

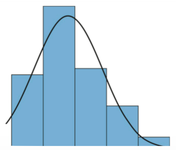
Get some result

Probabilistic programming

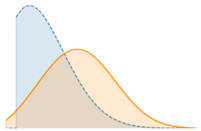
Why not just programming?

Feed some data???

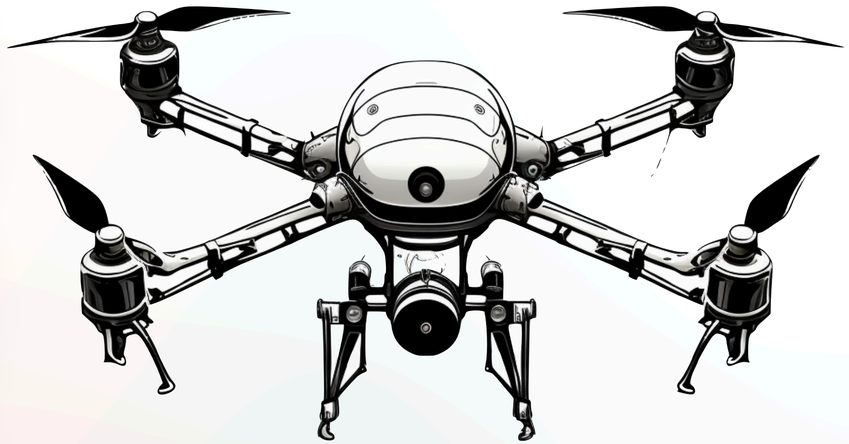
velocity =

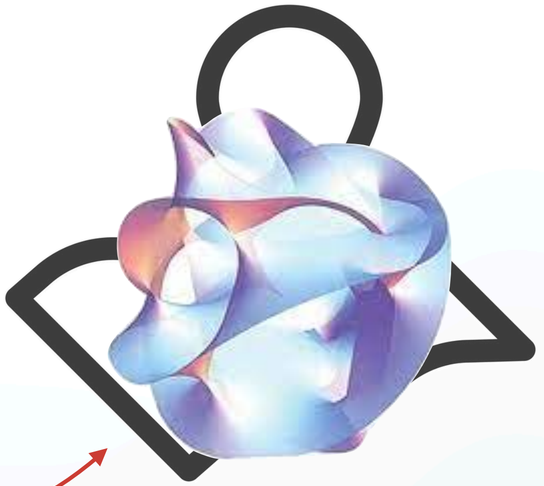


position =



windspeed =



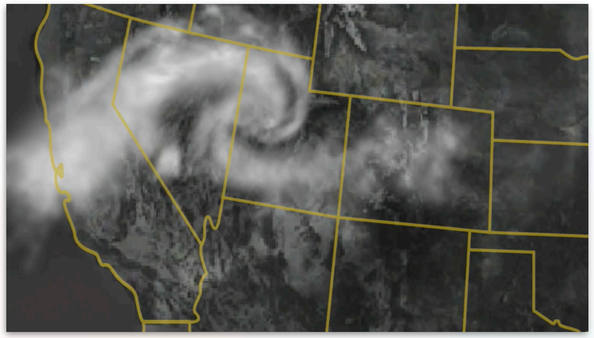


Get some result???

Probabilistic programming

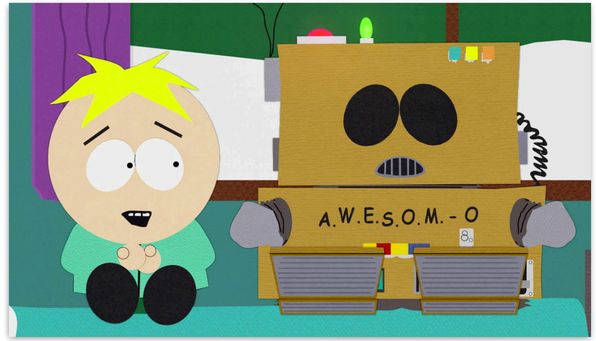
Do we really need it?

Pharmacology



Climate modelling

Logistics



Robotics

Auto-pilots

Finances

Audio

And so on...

Probabilistic programming

Bayesian Inference

Likelihood

how "likely" are the states, given the data?

Posterior

prior knowledge: how probable are the hidden states before observing any data

$$p(\boldsymbol{s}|\hat{\boldsymbol{y}}) = \frac{p(\hat{\boldsymbol{y}}|\boldsymbol{s})p(\boldsymbol{s})}{p(\hat{\boldsymbol{y}})}$$

Posterior

updated knowledge: how probable are hidden states given the observed data

Evidence

how probably are these particular observations given our assumptions

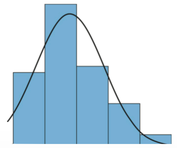
Why is it so challenging?

Probabilistic programming

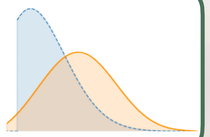
Why is it so challenging?

Account for the uncertainty in data 🎲

velocity =



position =



windspeed =

$$y = 2x$$

$$x = 1$$

Solve simple math 😎

Probabilistic programming

Why is it so challenging?

Add more components

velocity

position

windspeed

actions

angle

rotation

audio

video

$$Z_\mu^0 Z_\nu^0 W_\nu^+ W_\nu^-) + g^2 s_w^2 (A_\mu W_\mu^+ A_\nu W_\nu^- - A_\mu A_\mu W_\nu^+ W_\nu^-) + g^2 s_w c_w (A_\mu Z_\nu^0 (W_\mu^+ W_\nu^- - W_\nu^+ W_\mu^-) - 2 A_\mu Z_\mu^0 W_\nu^+ W_\nu^-) - \tfrac{1}{2}\partial_\mu H \partial_\mu H - 2M^2 \alpha_h H^2 - \partial_\mu \phi^+ \partial_\mu \phi^- - \tfrac{1}{2}\partial_\mu \phi^0 \partial_\mu \phi^0 - \beta_h \left(\tfrac{2M^2}{g^2} + \tfrac{2M}{g} H + \tfrac{1}{2}(H^2 + \phi^0 \phi^0 + 2\phi^+ \phi^-) \right) + \tfrac{2M^4}{g^2} \alpha_h - g \alpha_h M (H^3 + H \phi^0 \phi^0 + 2 H \phi^+ \phi^-) - \tfrac{1}{8} g^2 \alpha_h (H^4 + (\phi^0)^4 + 4(\phi^+ \phi^-)^2 + 4(\phi^0)^2 \phi^+ \phi^- + 4 H^2 \phi^+ \phi^- + 2(\phi^0)^2 H^2) - g M W_\mu^+ W_\mu^- H - \tfrac{1}{2} g \tfrac{M}{c_w^2} Z_\mu^0 Z_\mu^0 H - \tfrac{1}{2} i g \left(W_\mu^+ (\phi^0 \partial_\mu \phi^- - \phi^- \partial_\mu \phi^0) - W_\mu^- (\phi^0 \partial_\mu \phi^+ - \phi^+ \partial_\mu \phi^0) \right) + \tfrac{1}{2} g \left(W_\mu^+ (H \partial_\mu \phi^- - \phi^- \partial_\mu H) + W_\mu^- (H \partial_\mu \phi^+ - \phi^+ \partial_\mu H) \right) + \tfrac{1}{2} g \tfrac{1}{c_w} (Z_\mu^0 (H \partial_\mu \phi^0 - \phi^0 \partial_\mu H) + M \left(\tfrac{1}{c_w} Z_\mu^0 \partial_\mu \phi^0 + W_\mu^+ \partial_\mu \phi^- + W_\mu^- \partial_\mu \phi^+ \right) - i g \tfrac{s_w^2}{c_w} M Z_\mu^0 (W_\mu^+ \phi^- - W_\mu^- \phi^+) + i g s_w M A_\mu (W_\mu^+ \phi^- - W_\mu^- \phi^+) - i g \tfrac{1 - 2c_w^2}{2 c_w} Z_\mu^0 (\phi^+ \partial_\mu \phi^- - \phi^- \partial_\mu \phi^+) + i g s_w A_\mu (\phi^+ \partial_\mu \phi^- - \phi^- \partial_\mu \phi^+) - \tfrac{1}{4} g^2 W_\mu^+ W_\mu^- (H^2 + (\phi^0)^2 + 2\phi^+ \phi^-) - \tfrac{1}{8} g^2 \tfrac{1}{c_w^2} Z_\mu^0 Z_\mu^0 (H^2 + (\phi^0)^2 + 2(2 s_w^2 - 1)^2 \phi^+ \phi^-) - \tfrac{1}{2} g^2 \tfrac{s_w^2}{c_w} Z_\mu^0 \phi^0 (W_\mu^+ \phi^- + W_\mu^- \phi^+) - \tfrac{1}{2} i g^2 \tfrac{s_w^2}{c_w} Z_\mu^0 H (W_\mu^+ \phi^- - W_\mu^- \phi^+) + \tfrac{1}{2} g^2 s_w A_\mu \phi^0 (W_\mu^+ \phi^- + W_\mu^- \phi^+) + \tfrac{1}{2} i g^2 s_w A_\mu H (W_\mu^+ \phi^- - W_\mu^- \phi^+) - g^2 \tfrac{s_w}{c_w} (2 c_w^2 - 1) Z_\mu^0 A_\mu \phi^+ \phi^- - g^2 s_w^2 A_\mu A_\mu \phi^+ \phi^- + \tfrac{1}{2} i g_s \lambda_{ij}^a (\bar{q}_i^\sigma \gamma^\mu q_j^\sigma) g_\mu^a - \bar{e}^\lambda (\gamma \partial + m_e^\lambda) e^\lambda - \bar{\nu}^\lambda (\gamma \partial + m_\nu^\lambda) \nu^\lambda - \bar{u}_j^\lambda (\gamma \partial + m_u^\lambda) u_j^\lambda - \bar{d}_j^\lambda (\gamma \partial + m_d^\lambda) d_j^\lambda + i g s_w A_\mu \left(-(\bar{e}^\lambda \gamma^\mu e^\lambda) + \tfrac{2}{3} (\bar{u}_j^\lambda \gamma^\mu u_j^\lambda) - \tfrac{1}{3} (\bar{d}_j^\lambda \gamma^\mu d_j^\lambda) \right) + \tfrac{ig}{4 c_w} Z_\mu^0 \{ (\bar{\nu}^\lambda \gamma^\mu (1 + \gamma^5) \nu^\lambda) + (\bar{e}^\lambda \gamma^\mu (4 s_w^2 - 1 - \gamma^5) e^\lambda) + (\bar{d}_j^\lambda \gamma^\mu (\tfrac{4}{3} s_w^2 - 1 - \gamma^5) d_j^\lambda) + (\bar{u}_j^\lambda \gamma^\mu (1 - \tfrac{8}{3} s_w^2 + \gamma^5) u_j^\lambda) \} + \tfrac{ig}{2\sqrt{2}} W_\mu^+ \left((\bar{\nu}^\lambda \gamma^\mu (1 + \gamma^5) U^{lep}{}_{\lambda\kappa} e^\kappa) + (\bar{u}_j^\lambda \gamma^\mu (1 + \gamma^5) C_{\lambda\kappa} d_j^\kappa) \right) + \tfrac{ig}{2\sqrt{2}} W_\mu^- \left((\bar{e}^\kappa U^{lep\dagger}{}_{\kappa\lambda} \gamma^\mu (1 + \gamma^5) \nu^\lambda) + (\bar{d}_j^\kappa C_{\kappa\lambda}^\dagger \gamma^\mu (1 + \gamma^5) u_j^\lambda) \right) + \tfrac{ig}{2M\sqrt{2}} \phi^+ \left(-m_e^\kappa (\bar{\nu}^\lambda U^{lep}{}_{\lambda\kappa} (1 - \gamma^5) e^\kappa) + m_\nu^\lambda (\bar{\nu}^\lambda U^{lep}{}_{\lambda\kappa} (1 + \gamma^5) e^\kappa) \right) + \tfrac{ig}{2M\sqrt{2}} \phi^- \left(m_e^\lambda (\bar{e}^\lambda U^{lep\dagger}{}_{\lambda\kappa} (1 + \gamma^5) \nu^\kappa) - m_\nu^\kappa (\bar{e}^\lambda U^{lep\dagger}{}_{\lambda\kappa} (1 - \gamma^5) \nu^\kappa) \right) - \tfrac{g}{2} \tfrac{m_\nu^\lambda}{M} H (\bar{\nu}^\lambda \nu^\lambda) - \tfrac{g}{2} \tfrac{m_e^\lambda}{M} H (\bar{e}^\lambda e^\lambda) + \tfrac{ig}{2} \tfrac{m_\nu^\lambda}{M} \phi^0 (\bar{\nu}^\lambda \gamma^5 \nu^\lambda) - \tfrac{ig}{2} \tfrac{m_e^\lambda}{M} \phi^0 (\bar{e}^\lambda \gamma^5 e^\lambda) - \tfrac{1}{4} \bar{\nu}_\lambda M_{\lambda\kappa}^R (1 - \gamma_5) \hat{\nu}_\kappa -$$

Solve ~~simple~~ math 😳

Probabilistic programming

Why is it so challenging?

Add even more components

velocity

position

angle

rotation

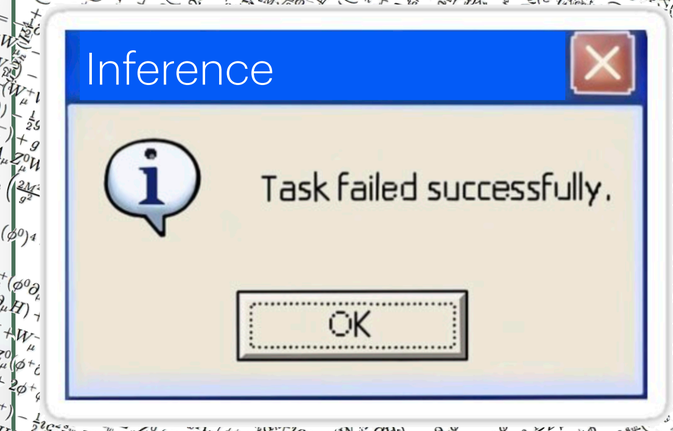
audio

video

physical constraints

engine power constraints

environment constraints



Probabilistic programming

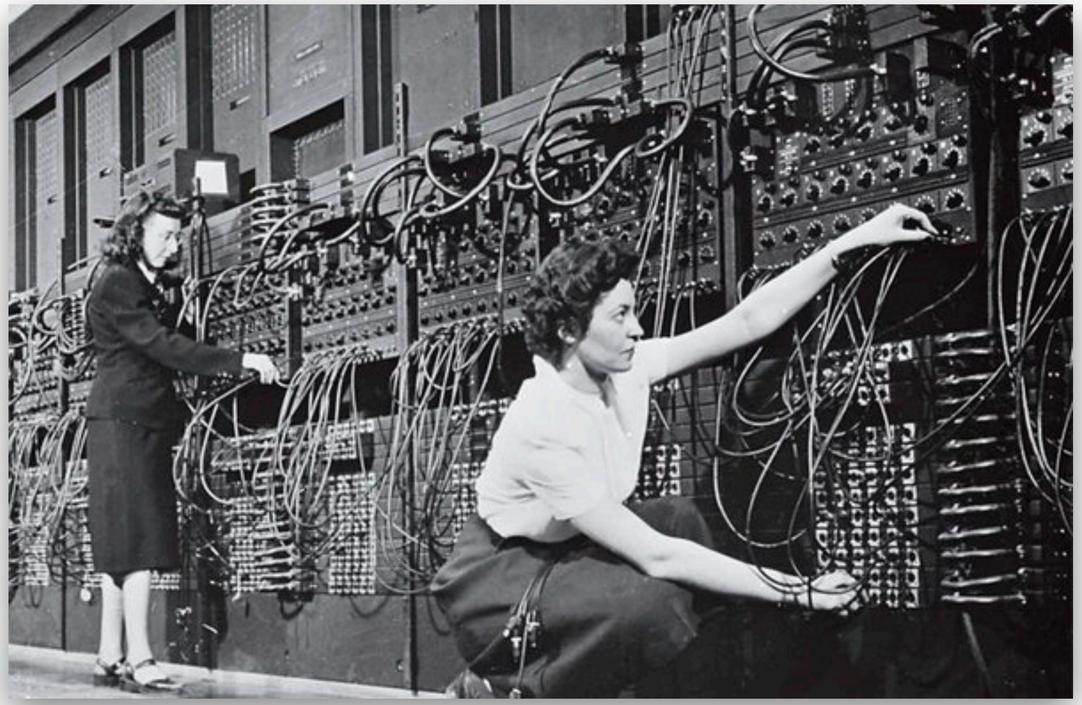
Why is it so challenging?

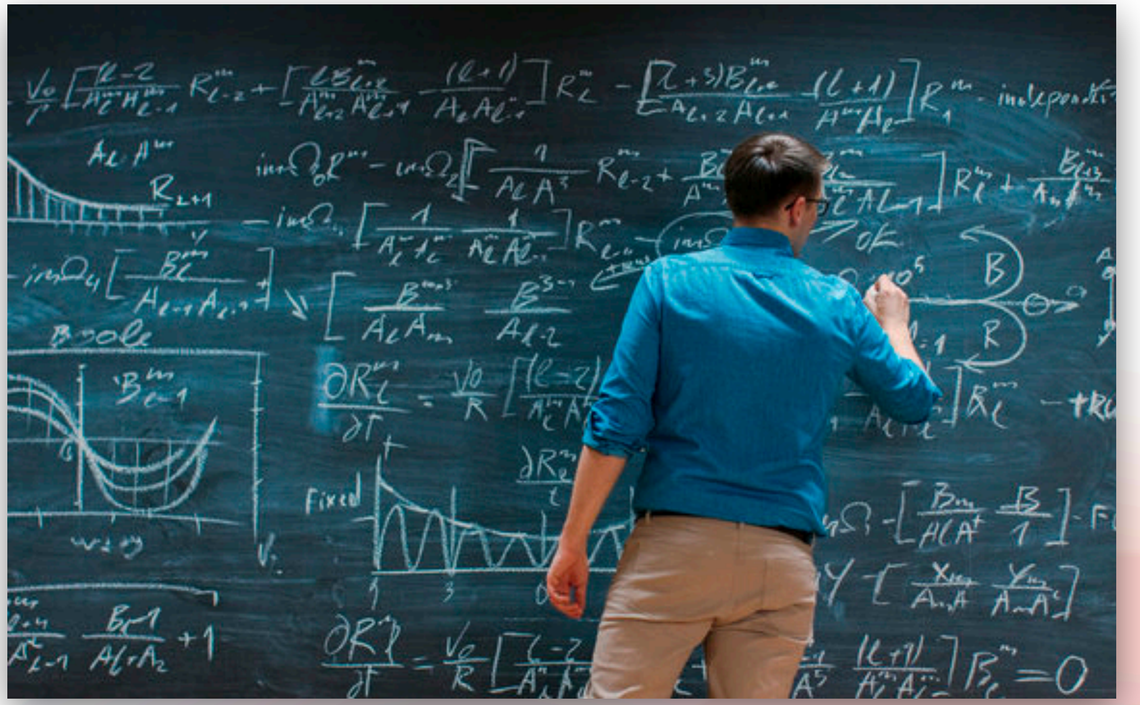
Add as many components as you want

compile

User-friendliness

Traditional programming and Probabilistic programming





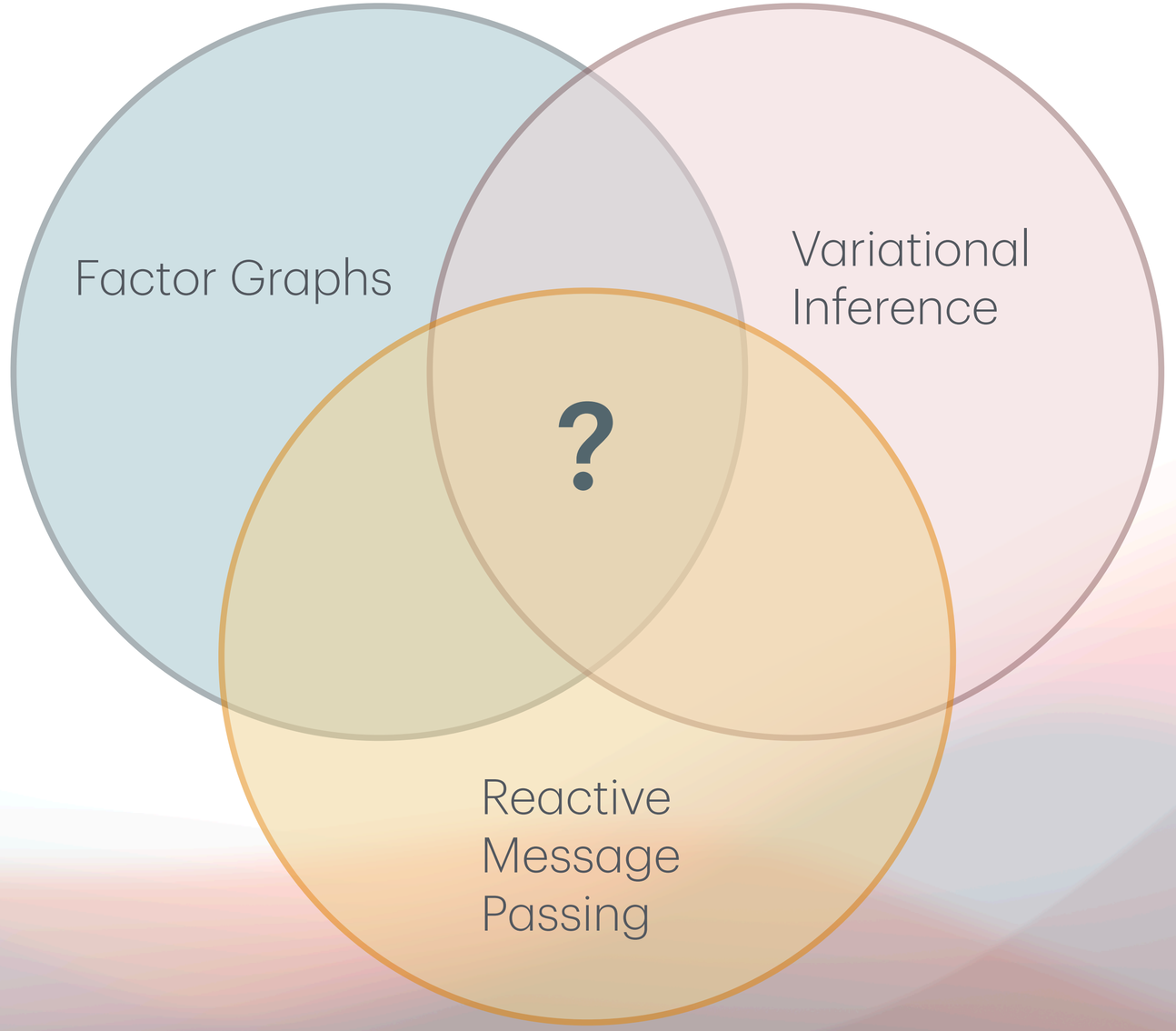
Probabilistic programming

Why is it so challenging?

How do we approach it?

How do we approach it?

Scalable Bayesian Inference architecture for autonomous systems



How do we approach it?

Scalable Bayesian Inference architecture for autonomous systems

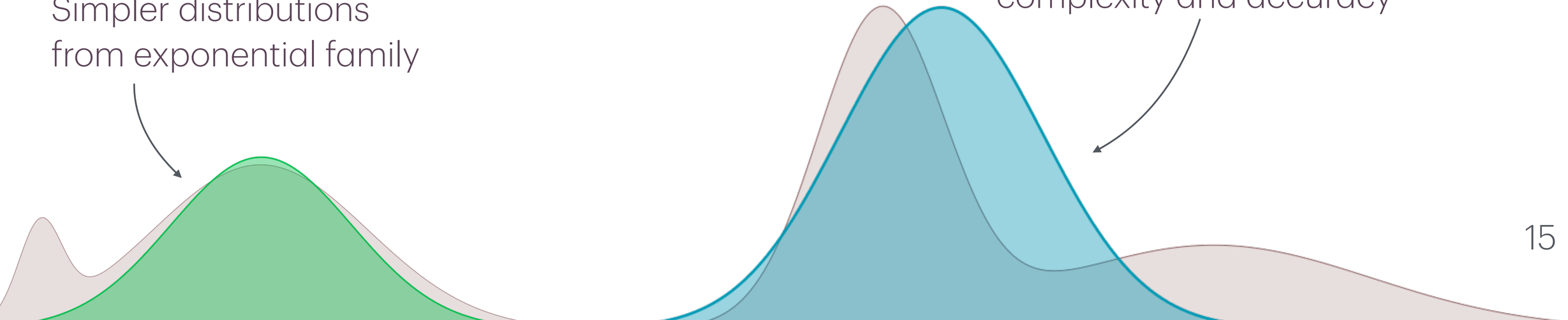
Variational inference

- Scalable and efficient
- Complexity vs accuracy trade-off

$$q^* = \arg\min_{q \in \mathcal{Q}} \int q(\boldsymbol{s}) \log \frac{q(\boldsymbol{s})}{p(\hat{\boldsymbol{y}}, \boldsymbol{s})} \mathrm{d}\boldsymbol{s}$$

Simpler distributions
from exponential family

Scalable trade-off between
complexity and accuracy



How do we approach it?

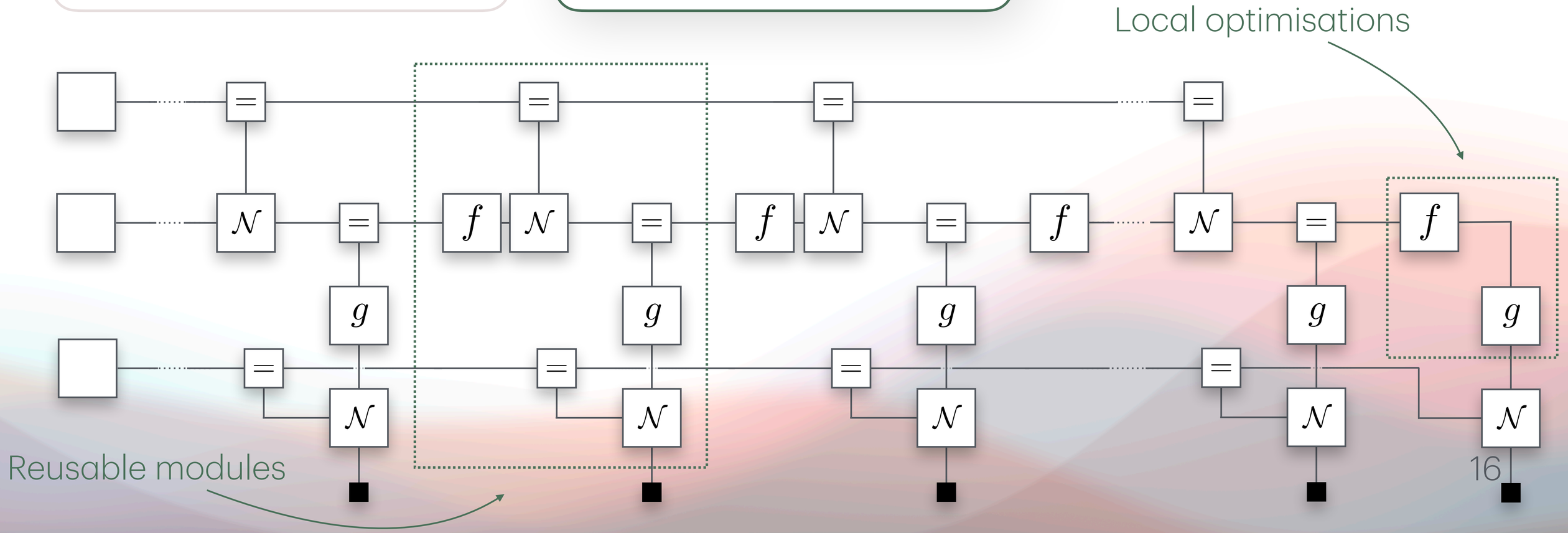
Scalable Bayesian Inference architecture for autonomous systems

Variational inference

- Scalable and efficient
- Complexity vs accuracy trade-off

Factor graphs

- Explainable and not a black-box
- Modular and local



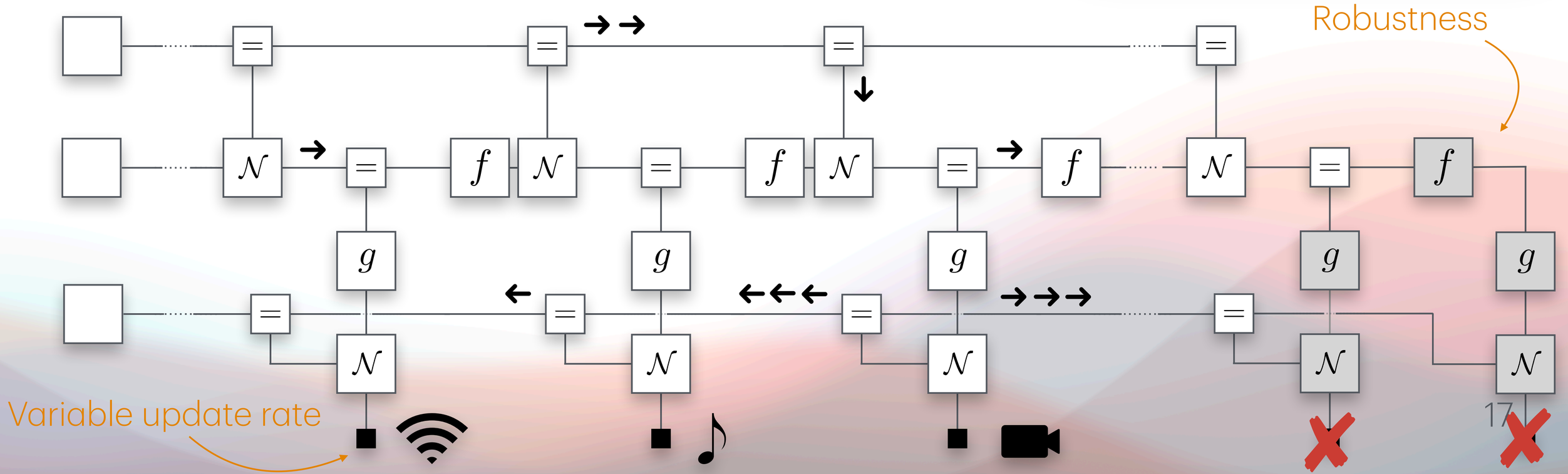
How do we approach it?

Scalable Bayesian Inference architecture for autonomous systems

Variational inference
- Scalable and efficient
- Complexity vs accuracy trade-off

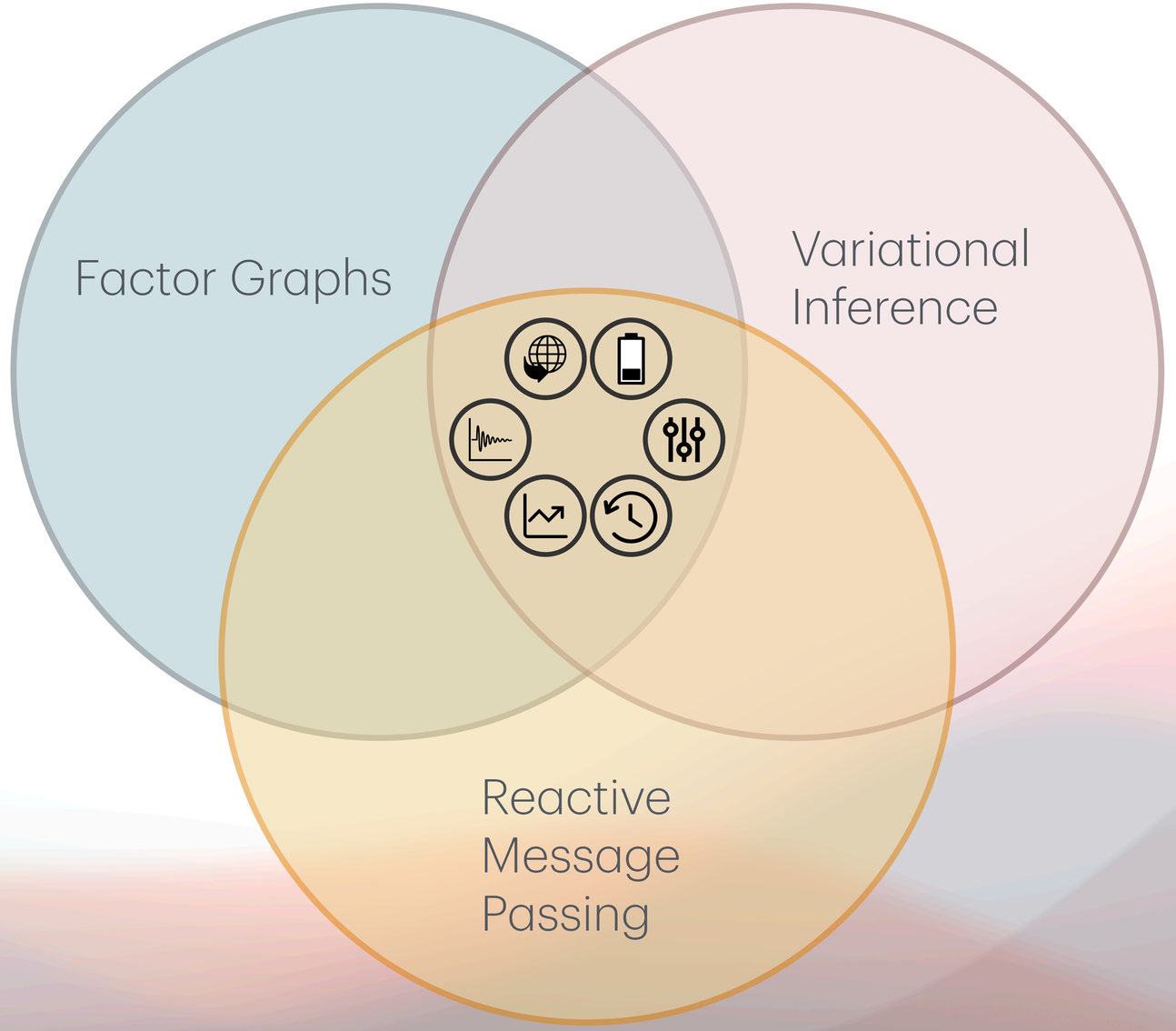
Factor graphs
- Explainable and not a black-box
- Modular and local

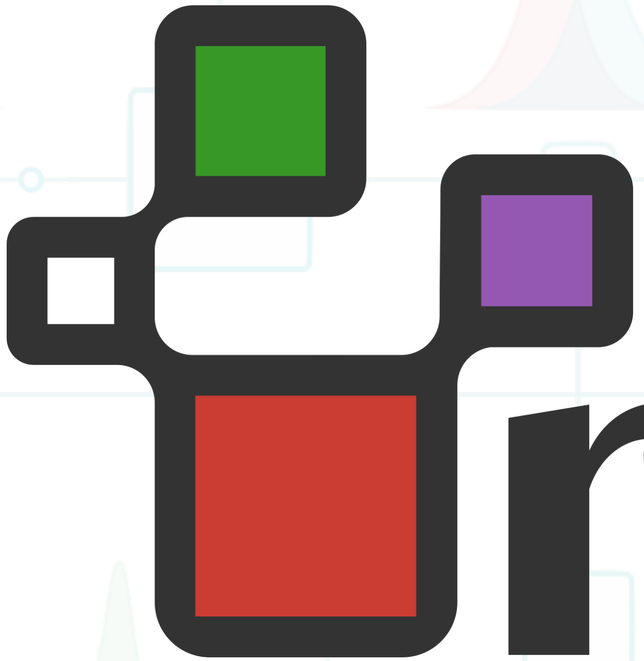
Reactive Message Passing
- Adaptive and robust
- Parallel and lazy



How do we approach it?

Scalable Bayesian Inference architecture for autonomous systems





rxinfer

Automatic Bayesian Inference through **Reactive Message Passing**

Reactive Bayesian Inferenc

Automatic Bayesian Inference through Reactive Message Passing

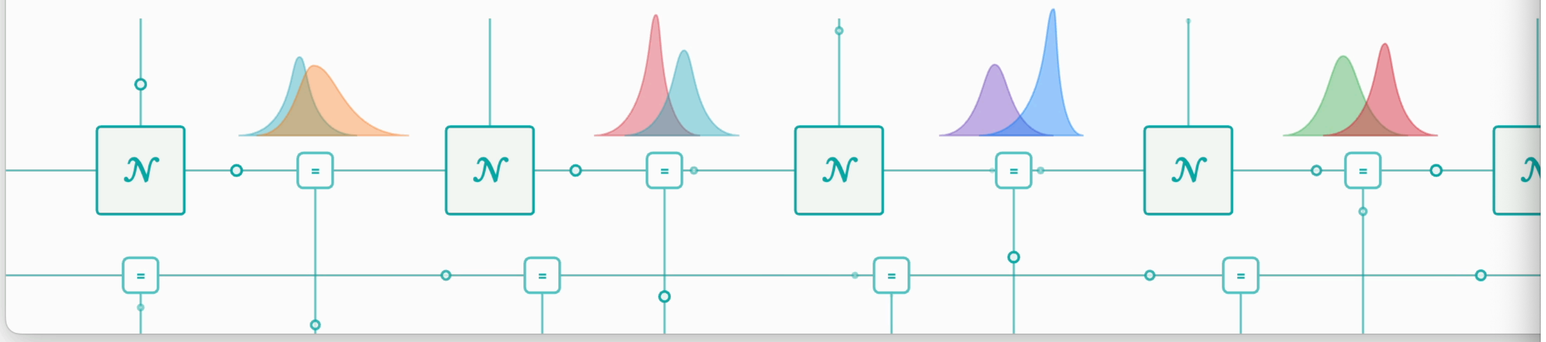
RxInfer.jl

rxinfer.ml

rxinfer Get Started Documentation Examples Discussions Team Contact GitHub

rxinfer

Automatic Bayesian Inference through **Reactive Message Passing**



biaslab/RxInfer.jl: Julia packag

github.com/biaslab/RxInfer.jl

biaslab / RxInfer.jl

Type / to search

Code Issues 18 Pull requests 2 Discussions Actions Projects 1 Wiki Security Insights Settings

RxInfer.jl Public

Edit Pins Watch 8 Fork 14

main 19 branches 37 tags Go to file Add file Code

bvdmitri update: Bump version to 2.15.0 2 792c6e3 5 days ago 731 commits

.github/workflows	Merge branch 'main' into dev-exp-family	last month
benchmarks	wip restructure	5 months ago
docs	Update index.md	last week
examples	Merge pull request #189 from biaslab/dev-fix-imports	2 weeks ago
paper	paper: fix doi for the Julia paper	9 months ago
scripts	fix examples	2 months ago
src	fix import warnings	2 weeks ago
test	use ReTestItems	5 days ago
.JuliaFormatter.toml	apply new formatting rules	last year
.gitignore	preliminary fixes, not complete	5 months ago
CITATION.cff	add the CITATION.cff and a reference to the JOSS paper	8 months ago
CODE_OF_CONDUCT.md	Update CoC	10 months ago
LICENSE	chore: update README & LICENSE	last year
Makefile	adjust Makefile	last week
Project.toml	update: Bump version to 2.15.0	5 days ago
README.md	Update README.md with href to MIT license	2 months ago
codemeta.json	update: Bump version to 2.15.0	5 days ago

README.md

rxinfer

official page RxInfer | docs stable | docs dev | examples RxInfer | CI passing | codecov 80%

Journal of Open Source Software 10.21105/joss.05161 | Zenodo 10.5281/zenodo.7774921

Overview

`RxInfer.jl` is a Julia package for automatic Bayesian inference on a factor graph with reactive message passing.

Given a probabilistic model, RxInfer allows for an efficient message-passing based Bayesian inference. It uses the model structure to generate an algorithm that consists of a sequence of local computations on a Forney-style factor graph (FFG) representation of the model.

About

Julia package for au inference on a facto message passing

Readme

MIT license

Code of conduct

Cite this repositor

Activity

175 stars

8 watching

14 forks

Report repository

Releases 33

v2.15.0 Latest

5 days ago

+ 32 releases

Packages

No packages published

Publish your first package

Contributors 16

+ 2 contributors

Deployments 213

github-pages 1 ho

+ 212 deployments

Languages

Jupyter Notebook 6

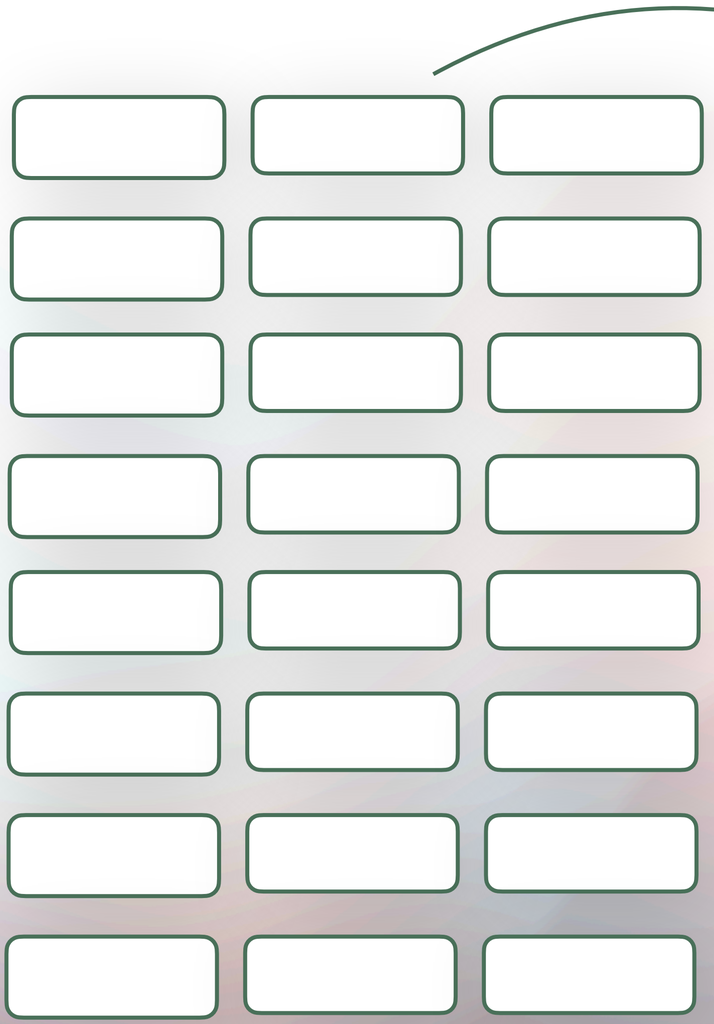
Julia 36.0%

Other 0.4%

Reactive Bayesian Inference

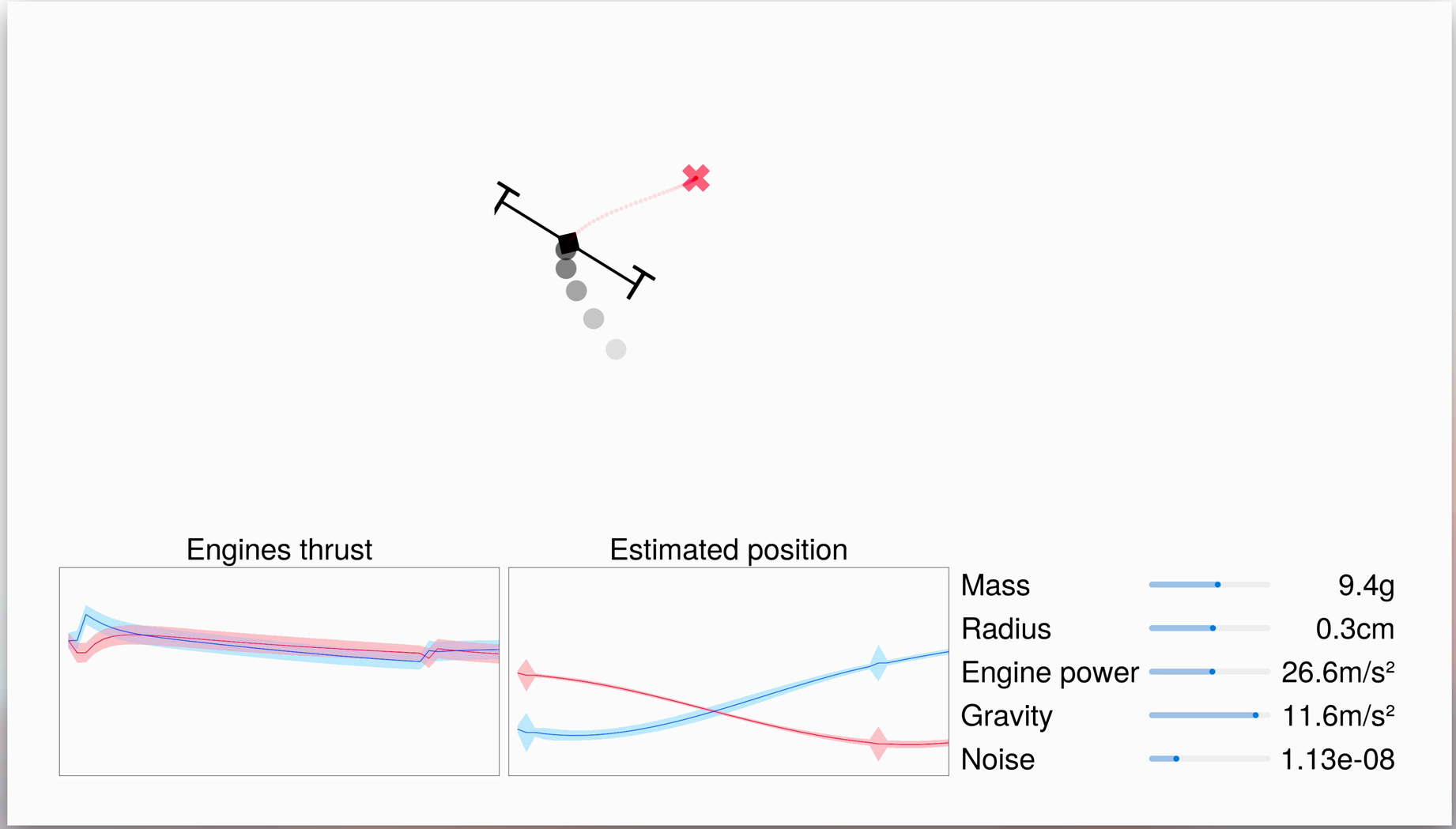
Automatic Bayesian Inference through Reactive Message Passing

Add as many components as you want



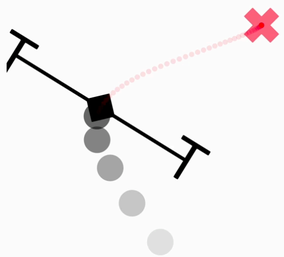
Reactive Bayesian Inference

Automatic Bayesian Inference through Reactive Message Passing

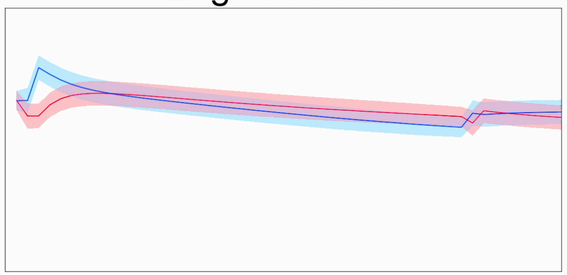


Reactive Bayesian Inference

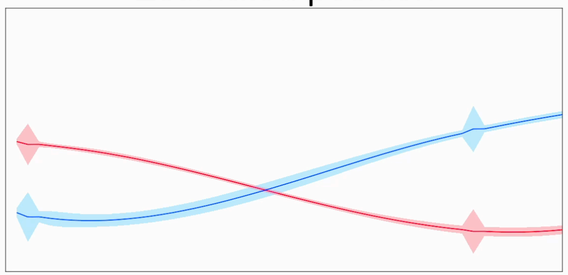
Automatic Bayesian Inference through Reactive Message Passing



Engines thrust



Estimated position

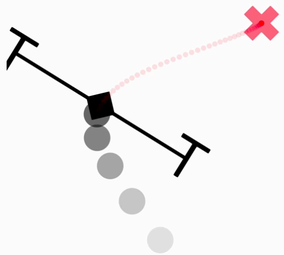


Mass	
Radius	0.3cm
Engine power	26.6m/s²
Gravity	11.6m/s²
Noise	1.13e-08

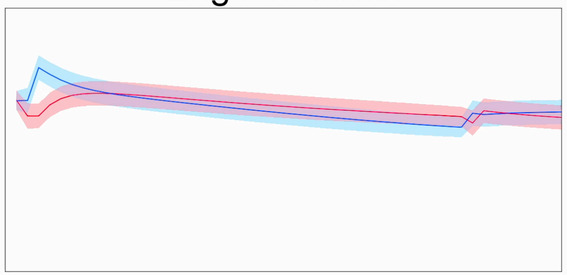
```julia
function state_transition(state, action, drone, environment)
    dt, g, _      = environment
    _, r, m, l    = drone
    Fₗ, Fᵣ        = action
    x, y, ẋ, ẏ, θ = state

    dẋ = (Fₗ + Fᵣ)/m * sin(θ)
    dẏ = (Fₗ + Fᵣ)/m * cos(θ) - g/m
    dθ = (Fₗ - Fᵣ) * r
    ẋ  = ẋ + dt * dẋ
    ẏ  = ẏ + dt * dẏ

    return [ x + dt * ẋ, y + dt * ẏ, ẋ, ẏ, rem(θ + dt * dθ, 2π) ]
end

@model function drone_dynamics(drone, environment, priors)

    # Goal may change
    goalM = datavar(Vector{Float64}, N)
    goalV = datavar(Matrix{Float64}, N)

    # The states and the actions are unknown components
    # The state is (x, y, ẋ, ẏ, θ)
    s  = randomvar(N)
    F  = randomvar(N)

    # We know the current position only up to a certain degree
    current_s ~ MvNormal(
        mean = priors[:current],
        covariance = 1e-6 * diageye(5)
    )
    previous_s = current_s

    # Predict the future up to N time steps
    N = drone.horizon
    for i in 1:N
        # Prior on the engine force
        F[i] ~ MvNormal(mean = priors[:actions][i], covariance = diageye(2))
        # State transition function
        s[i] ~ MvNormal(
            mean = state_transition(previous_s, F[i], drone, environment),
            covariance = environment.noise * diageye(5)
        )
        # Set the desired goal
        s[i] ~ MvNormal(mean = goalM[i], covariance = goalV[i])
        previous_s = s[i]
    end

end

@meta function drone_dynamics_approximations()
    # Use unscented transform to approximate the non-linearity
    state_transition() -> Unscented()
end
```

Reactive Bayesian Inference

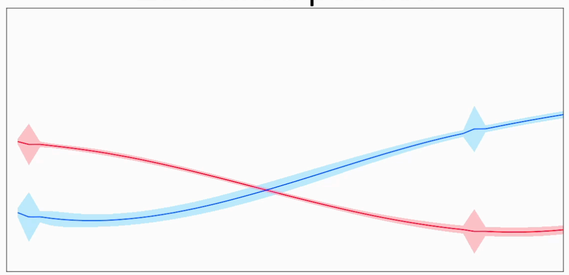
Automatic Bayesian Inference through Reactive Message Passing



Engines thrust



Estimated position



Mass

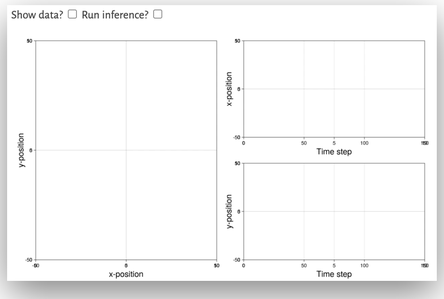
Radius

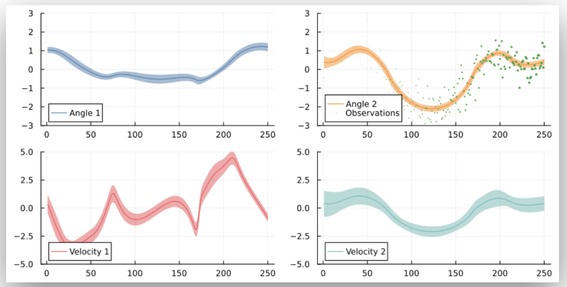
Engine p

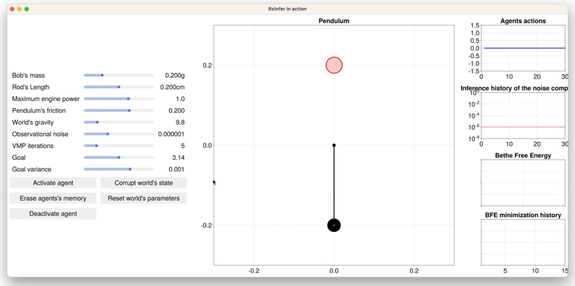
Gravity 11.6m/s²

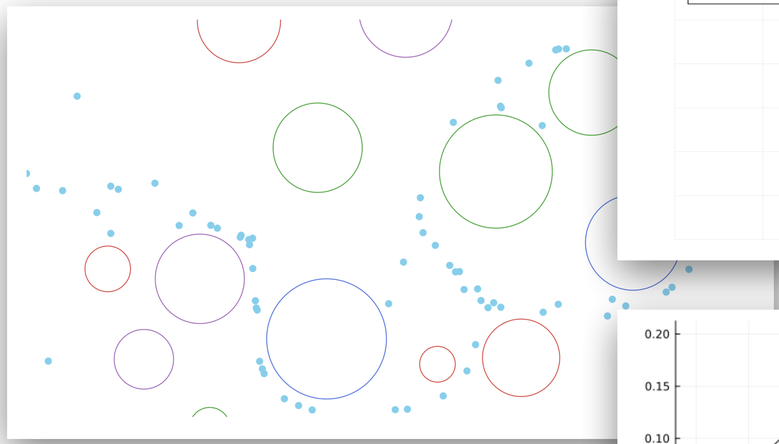
Noise 1.13e-08

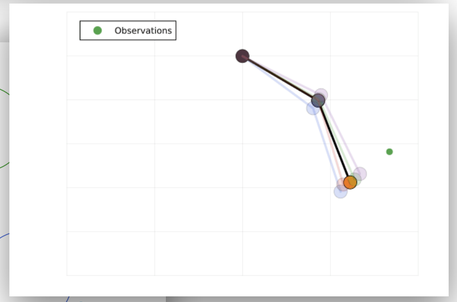
```
function state_transition(state, action, drone, envi
    dt, g, _       = environment
    _, r, m, l     = drone
    Fₗ, Fᵣ         = action
    x, y, ẋ, ẏ, θ = state

    dẋ = (Fₗ + Fᵣ)/m * sin(θ)
    dẏ = (Fₗ + Fᵣ)/m * cos(θ) - g/m
    dθ = (Fₗ - Fᵣ) * r
    ẋ  = ẋ + dt * dẋ
    ẏ  = ẏ + dt * dẏ

    return [ x + dt * ẋ, y + dt * ẏ, ẋ, ẏ, rem(θ + dt
end

@model function drone_dynamics(drone, environment, p

    # Goal may change
    goalM = datavar(Vector{Float64}, N)
    goalV = datavar(Matrix{Float64}, N)

    # The states and the actions are unknown compone
    # The state is (x, y, ẋ, ẏ, θ)
    s  = randomvar(N)
    F  = randomvar(N)

    # We know the current position only up to a certa
    current_s ~ MvNormal(
        mean = priors[:current],
        covariance = 1e-6 * diageye(5)
    )
    previous_s = current_s

    # Predict the future up to N time steps
    N = drone.horizon
    for i in 1:N
        # Prior on the engine force
        F[i] ~ MvNormal(mean = priors[:actions][i],
        # State transition function
        s[i] ~ MvNormal(
            mean = state_transition(previous_s, F[i]
            covariance = environment.noise * diageye
        )
        # Set the desired goal
        s[i] ~ MvNormal(mean = goalM[i], covariance =
        previous_s = s[i]
    end

end

@meta function drone_dynamics_approximations()
    # Use unscented transform to approximate the non
    state_transition() -> Unscented()
end
```

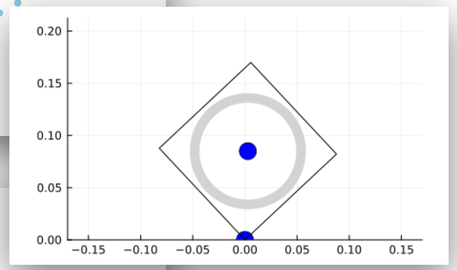


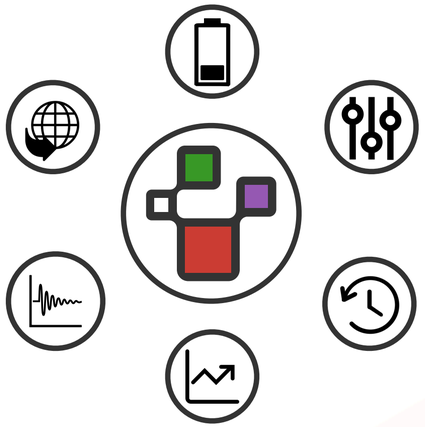


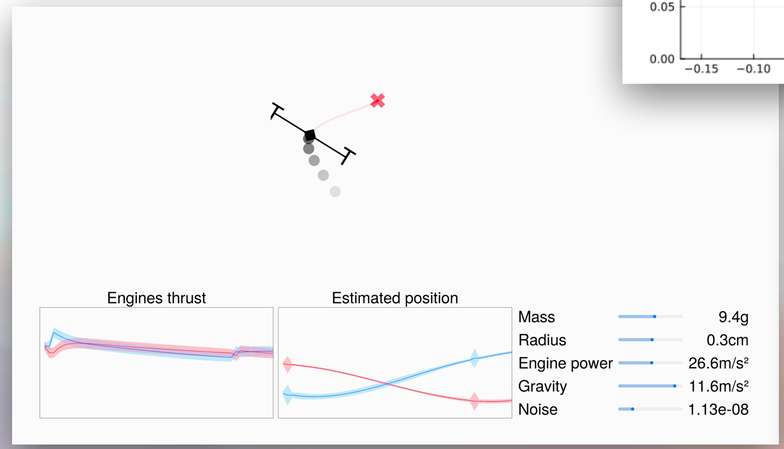


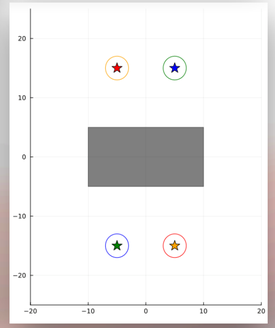


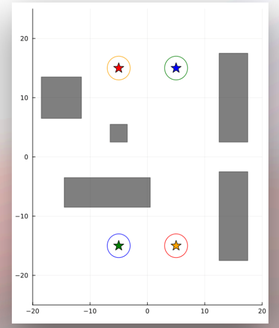


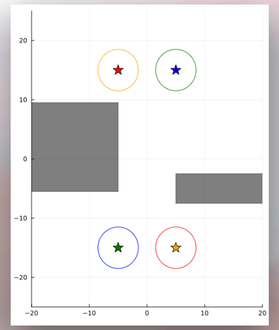












RxInfer.jl

Search docs

Overview

Coin toss model (Beta-Bernoulli)

Bayesian Linear Regression

Active Inference Mountain car

Chance-Constrained Active Inference

Assessing People's Skills

Gaussian Linear Dynamical System

Ensemble Learning of a Hidden Markov Model

Autoregressive Model

Hierarchical Gaussian Filter

Bayesian ARMA model

Infinite Data Stream

System Identification Problem

Univariate Gaussian Mixture Model

Multivariate Gaussian Mixture Model

Gamma Mixture Model

Universal Mixtures

Global Parameter Optimisation

Invertible neural networks: a tutorial

Conjugate-Computational Variational Message Passing (CVI)

Solve GP regression by SDE

Nonlinear Smoothing: Noisy Pendulum

Nonlinear Smoothing: Rabbit Population

Nonlinear Virus Spread

Nonlinear Sensor Fusion

Kalman filter with LSTM network driven dynamic

Handling Missing Data

Custom Nonlinear Node

Probit Model (EP)

Version v2.11.1

Home Get Started Documentation Examples Pape

All examples have been pre-generated automatically from the `example`

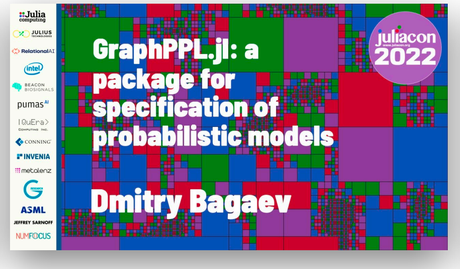
- Coin toss model (Beta-Bernoulli): An example of Bayesian inference observations.
- Bayesian Linear Regression: An example of Bayesian linear regressio
- Active Inference Mountain car: This notebooks covers RxInfer usage simple mountain car problem.
- Chance-Constrained Active Inference: This notebook applies reactiv the context of chance-constraints.
- Assessing People's Skills: The demo is inspired by the example from Machine Learning book. We are going to perform an exact inference results of the test.
- Gaussian Linear Dynamical System: An example of inference proced with multivariate noisy observations using Belief Propagation (Sum Bayesian Filtering and Smoothing.
- Ensemble Learning of a Hidden Markov Model: An example of struc Hidden Markov Model with unknown transition and observational
- Autoregressive Model: An example of variational Bayesian Inferenc Reference: Albert Podusenko, Message Passing-Based Inference for
- Hierarchical Gaussian Filter: An example of online inference proced univariate noisy observations using Variational Message Passing alg Message Passing-based Inference in the Hierarchical Gaussian Filter
- Bayesian ARMA model: This notebook shows how Bayesian ARMA be implemeted in RxInfer.jl
- Infinite Data Stream: This example shows RxInfer capabilities of run
- System Identification Problem: This example attempts to identify an
- Univariate Gaussian Mixture Model: This example implements varia Gaussian mixture model with mean-field assumption.
- Multivariate Gaussian Mixture Model: This example implements var Gaussian mixture model with mean-field assumption.
- Gamma Mixture Model: This example implements one of the Gamm https://biaslab.github.io/publication/mp-based-inference-in-gmm/ .
- Universal Mixtures: Universal mixture modeling.
- Global Parameter Optimisation: This example shows how to use Rx optimisation packages such as Optim.jl.
- Invertible neural networks: a tutorial: An example of variational Bay networks. Reference: Bart van Erp, Hybrid Inference with Invertible
- Conjugate-Computational Variational Message Passing (CVI): This e the non-conjugate message-passing based inference by exploiting t
- Solve GP regression by SDE: In this notebook, we solve a GP regres Differential Equation' (SDE). This method is well described in the di methods for spatio-temporal Gaussian process regression.' by Arno Temporal Gaussian Process Models' by Jouni Hartikainen.
- Nonlinear Smoothing: Noisy Pendulum: In this demo, we will look at state transitions: tracking a noisy single pendulum. We translate a d form to a probabilistic model.
- Nonlinear Smoothing: Rabbit Population: In this demo, we will look transitions. We will start with a one-dimensional problem; the numb seems overly simple, but it is a good way to demonstrate the basic p
- Nonlinear Virus Spread: In this demo we consider a model for the sp population. We are interested in estimating the reproduction rate fr infected individuals.
- Nonlinear Sensor Fusion: Nonlinear object position identification us
- Kalman filter with LSTM network driven dynamic: In this demo, we in Nonlinear State-Space Model using the LSTM.
- Handling Missing Data: This example shows how to extend the set

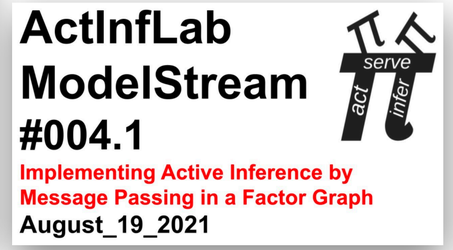
Reactive Probabilistic Programming for Scalable Bayesian Inference

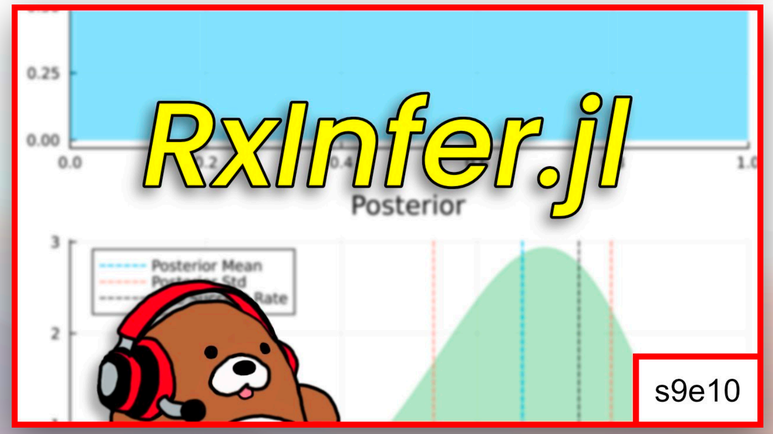
Dmitry Bagaev

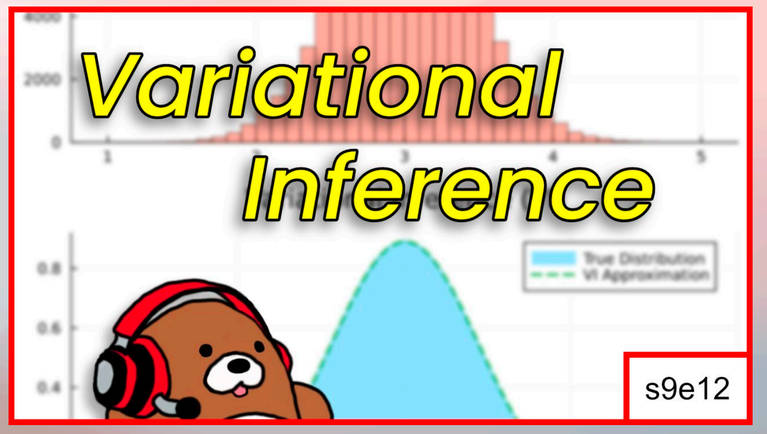
Recognition beyond BIASlab

Scalable Bayesian Inference architecture for autonomous systems





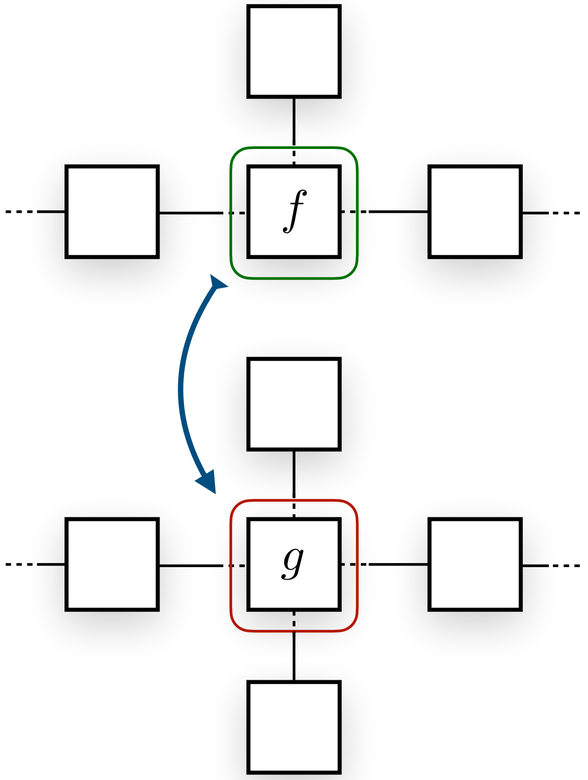




Future research ideas

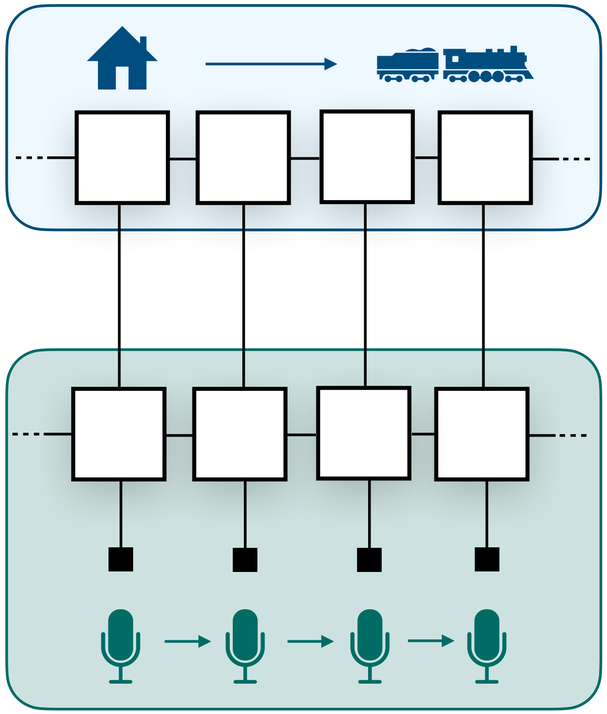
Structure and constraints adaptation in real time

Infer structure & constraints automatically



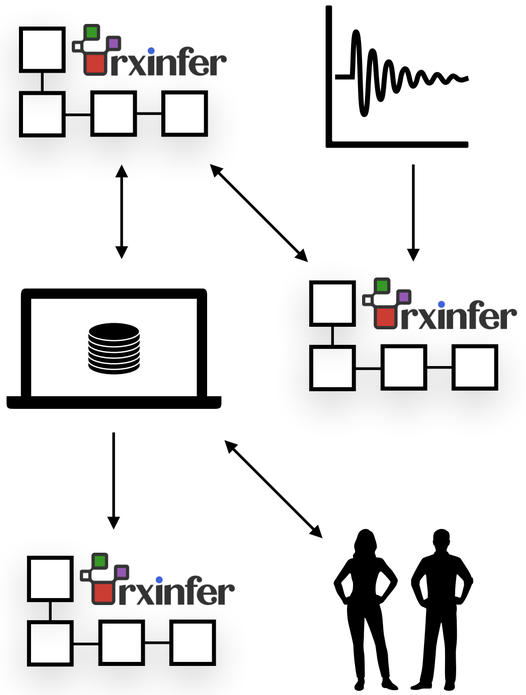
Tradeoff accuracy and computational complexity

Some states change their values fast, while others change their values over long period



Multi-agent interaction in changing environment

Reactivity is a natural way to support multi-agent interaction

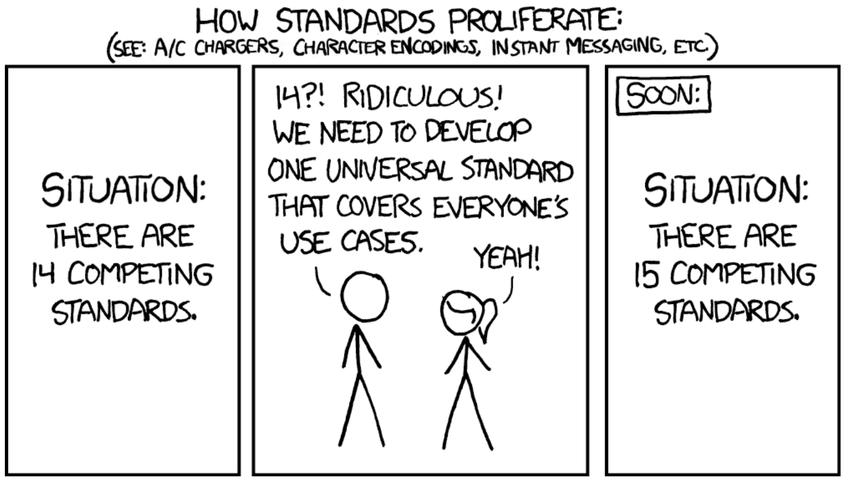


Many more

- Universality
- Dynamic constraints
- Documentation
- Usability
- EFE support
- Non-parametric
- Modular PPL
- Robustness
- Fixing

How do we approach it?

And why don't we use another solution?



Stan

https://mc-stan.org/

Pyro / NumPyro

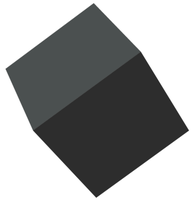
https://pyro.ai/

Turing

https://turinglang.org/

PyMC3

https://www.pymc.io/



infer.NET

https://dotnet.github.io/infer/

BAT.jl

https://bat.mpp.mpg.de/

JASP

https://jasp-stats.org/

BUGS

https://www.mrc-bsu.cam.ac.uk/software/bugs/

And others

How do we approach it?

And why don’t we use another solution?

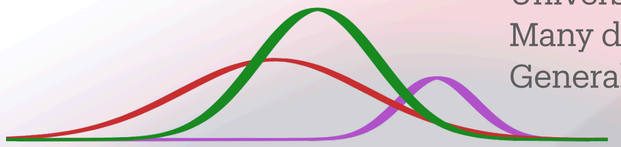
Efficiency

RxInfer

Fast
Scalable
Efficient
Reactive inference

Turing

Easy to use
Universal
Many different algorithms
General purpose



Universality

User-friendliness

The toolbox should be easy to use

```
@model function rotate_ssm(n, x0, A, B)

    x = randomvar(n)
    y = datavar(Vector{Float64}, n)

    x_prior ~ MvNormal(μ = mean(x0), cov(x0))
    x_prev = x_prior

    Q ~ InverseWishart(3, diageye(2))
    P ~ InverseWishart(3, diageye(2))

    for i in 1:n
        x[i] ~ MvNormal(μ = A * x_prev, Σ = Q)
        y[i] ~ MvNormal(μ = B * x[i], Σ = P)
        x_prev = x[i]
    end
end
```

How to specify?

Generative model $p(s, y)$

Variational distribution $q(s) \in \mathcal{Q}$

Compute posteriors $q^*(s) = \arg\min_{q \in \mathcal{Q}} F\left[\, q(s)\,,\, p(s, y)\,\right]$

$$
\begin{aligned}
p(Q) &= \mathcal{IW}(Q|\nu_Q, K_Q) \\
p(P) &= \mathcal{IW}(P|\nu_P, K_P) \\
p(x_k|x_{k-1}) &= \mathcal{N}(x_k|Ax_{k-1}, Q) \\
p(y_k|x_k) &= \mathcal{N}(y_k|Bx_k, P)
\end{aligned}
$$

How to specify?

```
@constraints begin
    q(x, Q, P) = q(x)q(Q)q(P)

    q(Q) :: PointMass
    q(P) :: PointMass
    q(x) :: Gaussian
end
```

$$\mathcal{Q} = \left\{\, q(x, Q, P) = q(x)q(Q)q(P)\ ,\ q(Q) = \delta(Q - \hat{Q}),\ q(P) = \delta(P - \hat{P})\ ,\ q(X) = \mathcal{N}(x) \right\}$$

How to minimise?

```
result = inference(
    model = rotate_ssm(n, s0, A, B),
    constraints = constraints,
    data = (y = dataset, )
    iterations = 10,
    free_energy = true
)
```

Focus on customizability

The toolbox as a playground

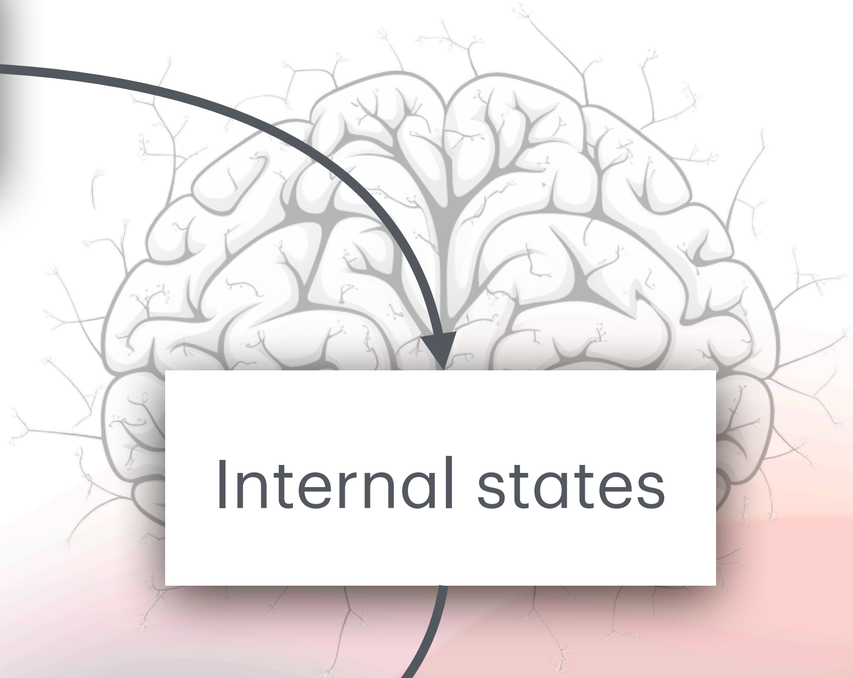
Advanced message passing playground

- Easy to add new custom factor nodes
- Easy to add new message update rules
  - Belief propagation
  - Mean-field/Structured VI
  - Expectation maximisation
  - Expectation propagation
- Easy to redefine product of messages
- Easy to add new functional form constraints
- Easy to change approximation methods
  - Unscented transform
  - Linearization
  - Conjugate-Computation VI
- Custom plugins for message passing itself
- A large part of the toolbox is an “extension”
- Debugging messages

How do we approach it?

Bayesian Brain hypothesis, Free Energy Principle and Active Inference

Sensation



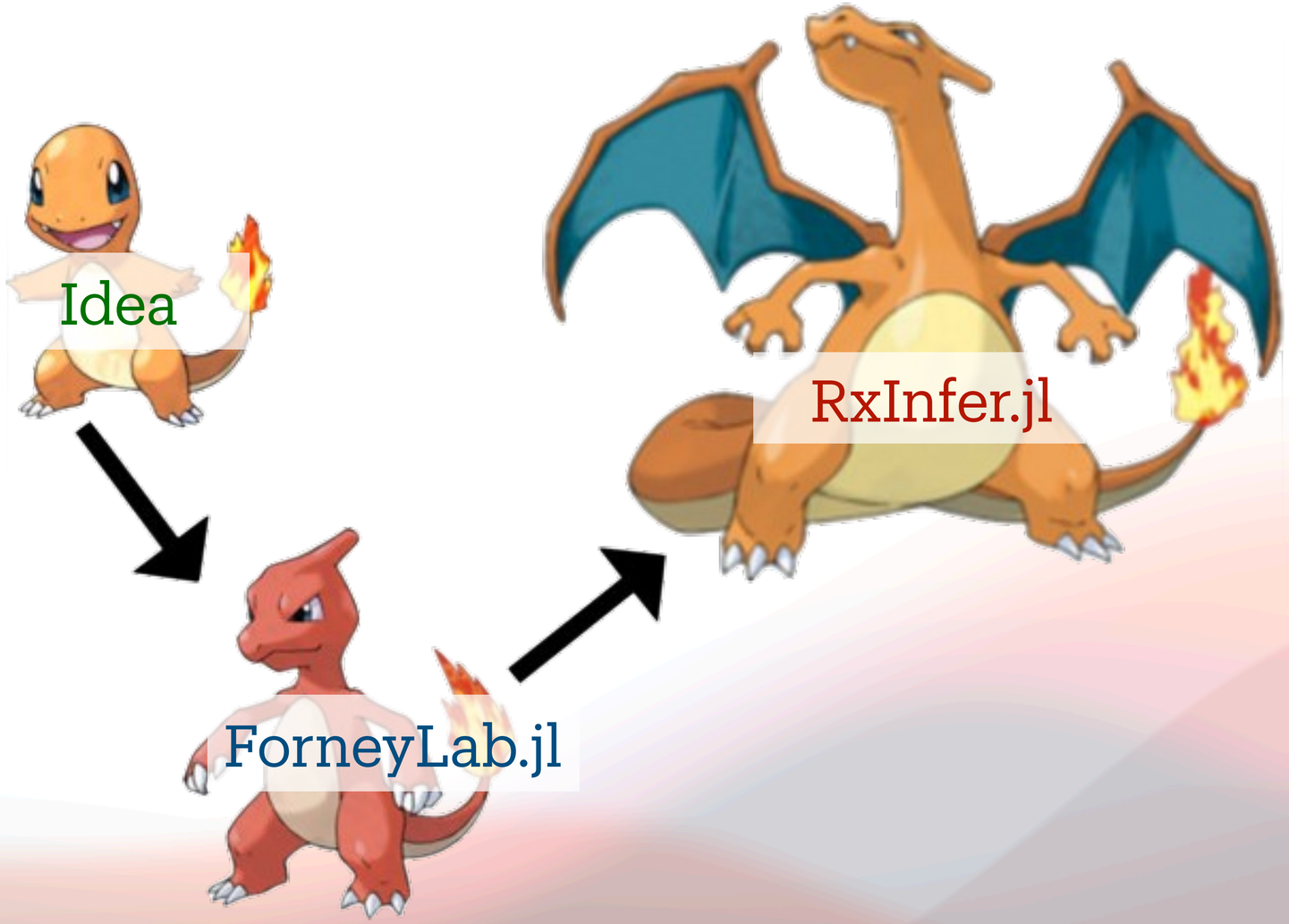
Internal states

Action

Hidden states

Comparison to ForneyLab

Evolution of ideas and implementations



Evolution of ideas and implementations

A lot of people contributed

The team

 bvdmitri

 albertpod

 bartvanerp

 ismailsenoz

The contributors

 wmkouw

 HoangMHNguyen

 Chengfeng-Jia

 Nimrais

 wouterwln

 Sepideh-Adamiat

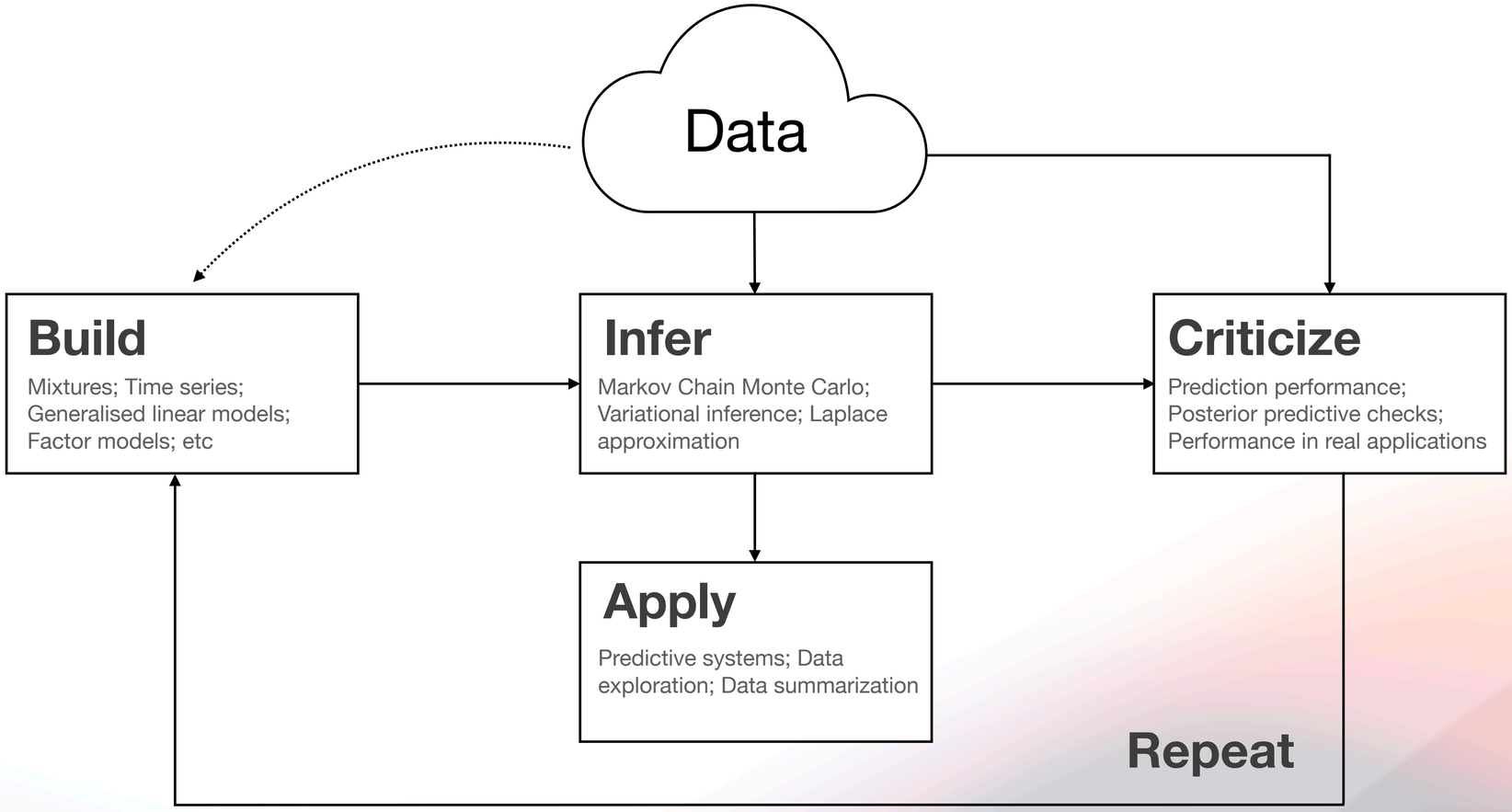
 ThijsvdLaar

 alstat

 and others…

Build, Infer, Apply, Criticize

until satisfactory or until your PhD contract runs out



Efficient Bayesian inference on the edge

What can you do with RxInfer?

We do Real-Time Bayesian Inference Audio Processing On These Little Beauties With Low Power And Performance

How do we specify prior knowledge?

Prior knowledge is an experience

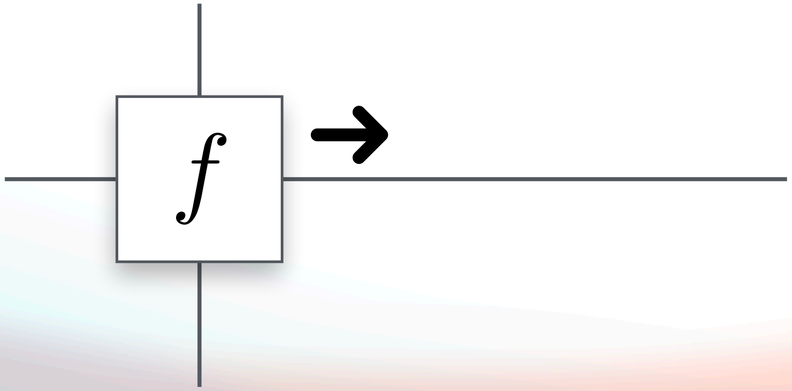
Message passing in a nutshell

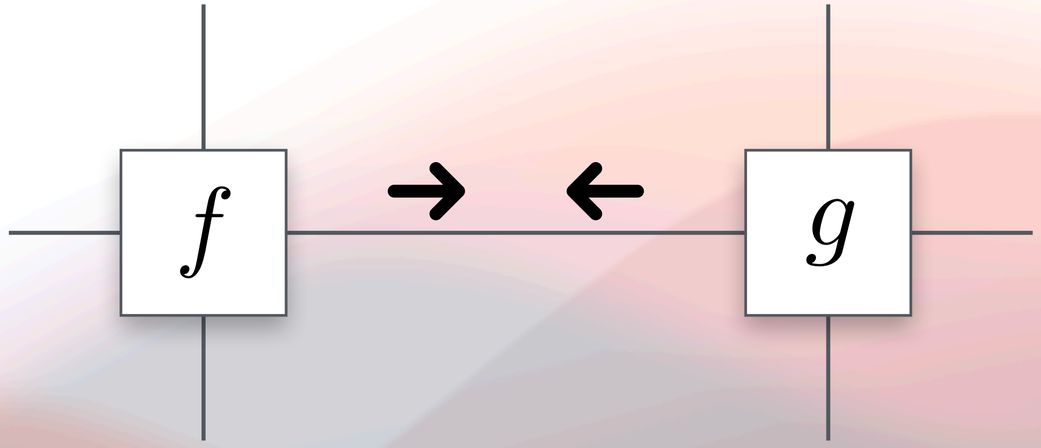
Variational Message Passing has two essential operations

$$f_a^n(\boldsymbol{s}_a^n) = \exp\left[\int \{\prod_{m\in\mathcal{C}(a),\ m\neq n} q_a^m(\boldsymbol{s}_a^m)\} \log f_a(\boldsymbol{s}_a)\mathrm{d}\boldsymbol{s}_{a\setminus n}\right]$$

$$\mu_{ib}(s_i) = \int f_a^n(\boldsymbol{s}_a^n) \prod_{j\in\mathcal{C}(a,i),\ j\neq i} \mu_{ja}(s_j)\mathrm{d}s_j$$

$$q_i(s_i) = \frac{\mu_{ia}(s_i)\mu_{ib}(s_i)}{\int \mu_{ia}(s_i)\mu_{ib}(s_i)\mathrm{d}s_i}$$





Blank slide hello

Blank slide hello