Pyro Meets SBI: Unlocking Hierarchical Bayesian Inference for Complex Simulators

Bridging probabilistic programming and simulation-based inference

EuroScipy 2025, Kraków, Poland

Jan Teusen (né Boelts)

Slides available at EuroSciPy website

About Me

- Maintainer of the `sbi` package

Background

- PhD in Machine Learning and Neuroscience in Tübingen, Germany
- Focus on "AI for Science"

Current Role

- Researcher @ TransferLab, appliedAI Institute for Europe
- OSS, AI research, AI education

A Journey Through Many Concepts

Hierarchical

Simulation-Based

Bayesian Inference

with **Pyro**

We'll build these concepts step by step using a simple example

Our Example: The Cookie Factory Problem 🍪

The Scenario:

- Cookie factory with 5 locations producing chocolate chip cookies
- Same global recipe, but each location might vary
- Data: Number of chocolate chips in 30 cookies for each location

The Goal:

- Understand the differences between locations
- Estimate typical chip count per location

The Challenge:

- local and global patterns, limited data

The Data

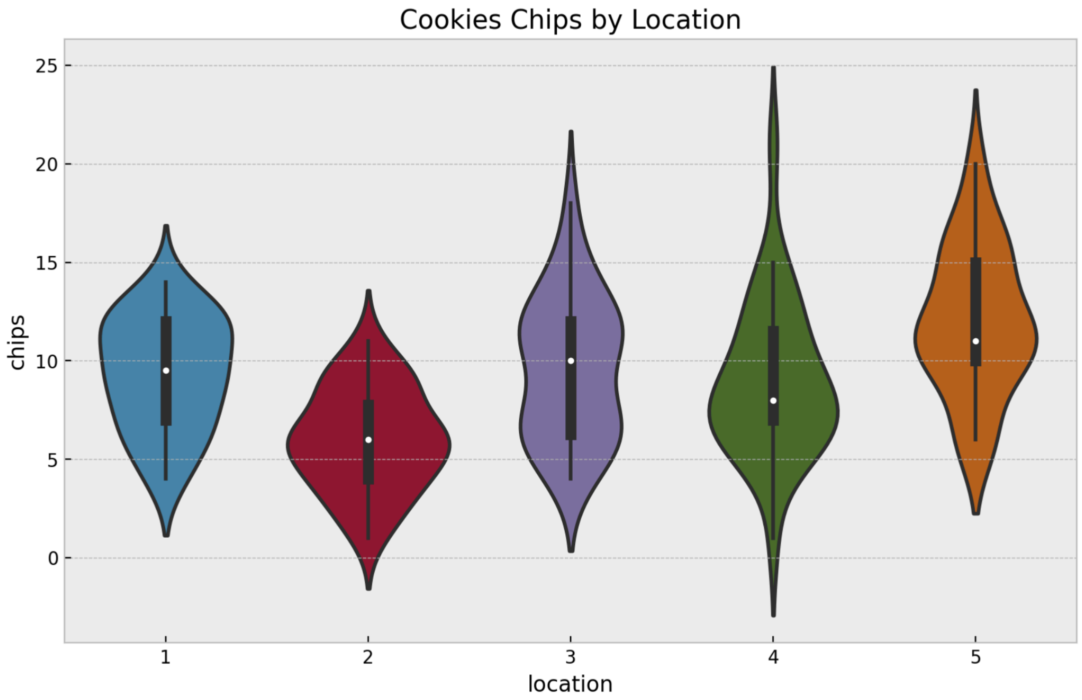
```
# Observed data: raw chip counts
location: [1, 1, 1, ..., 5, 5, 5]
chips:    [12, 12, 6, ..., 20, 11, 14]
```

Observations:

- Different means across locations
- 150 total observations
- 30 cookies per location

Key insight:

There might be a global and a local recipe



Chocolate chips across 5 factory locations

How Do We Model This?

Step 1: Choose a Probabilistic Model

- **Likelihood**: How data is generated given parameters
  - `chips ~ Poisson(rate)`
- **Prior**: Our beliefs before seeing data
  - `rate ~ Gamma(α, β)`

Why probabilistic?

- Captures uncertainty naturally
- Principled way to combine prior knowledge with data
- Enables hierarchical inference

The Goal: Bayesian Inference

What we want: Estimate rate λ for each location

Maximum Likelihood Estimate (point estimate)

```
λ_ML = argmax P(data|λ)  # Single number
# Location 1: λ = 9.3
```

Bayesian Inference (full distribution)

```
P(λ|data) ∝ P(data|λ) × P(λ)  # Distribution
# Location 1: λ ~ Gamma(9.3, 0.5 | data)
```

The Power: Uncertainty quantification!

- "I'm 95% confident λ is between 8.3 and 10.3"

Probabilistic Programming with Pyro

How do we implement these models efficiently?

Probabilistic Programming Languages (PPLs) let us write models as code:

```
def cookie_model(chips=None):
    # Prior
    lam = pyro.sample("lam", dist.Gamma(2, 0.2))
    # Likelihood
    pyro.sample("obs", dist.Poisson(lam), obs=chips)

nuts_kernel = NUTS(cookie_model)
mcmc = MCMC(nuts_kernel)
```

- direct access to MCMC or VI algos
- `pyro.plate` exploits conditional independence
- Easy switch between different models
- Also available: PyMC, Stan, NumPyro

Approach 1: Pooled Model

Assumption: All locations have the same rate

```python
def pooled_model(locations, chips=None):
    # One rate for all locations
    lam = pyro.sample("lam", dist.Gamma(2, 0.2))

    # Likelihood
    with pyro.plate("data", len(locations)):
        pyro.sample("obs", dist.Poisson(lam), obs=chips)
```

$$\text{chips} \sim \text{Poisson}(\lambda)$$

$$\lambda \sim \text{Gamma}(2, 0.2)$$

Problem: Ignores location differences!

Approach 2: Unpooled Model

Assumption: Each location is completely independent

```
def unpooled_model(locations, chips=None):
    n_locations = 5
    with pyro.plate("location", n_locations):
        # Independent rate per location
        lam = pyro.sample("lam", dist.Gamma(2, 0.2))

    # Likelihood
    rate = lam[locations]
    with pyro.plate("data", len(locations)):
        pyro.sample("obs", dist.Poisson(rate), obs=chips)
```

$$\text{chips}_\ell \sim \text{Poisson}(\lambda_\ell)$$

$$\lambda_\ell \sim \text{Gamma}(2, 0.2) \quad \forall \ell$$

Problem: No information sharing between locations!

Approach 3: Hierarchical Model ✨

Key Insight: Locations are different but related

```python
def hierarchical_model(locations, chips=None):
    # Hyperpriors - global parameters
    mu = pyro.sample("mu", dist.Gamma(2, 0.2))
    sigma = pyro.sample("sigma", dist.Exponential(1))

    # Location-specific rates (drawn from shared distribution)
    with pyro.plate("location", len(locations)):
        lam = pyro.sample("lam", dist.Gamma(mu**2/sigma**2, mu/sigma**2))

    # Likelihood
    with pyro.plate("data", len(locations)):
        pyro.sample("obs", dist.Poisson(lam[locations]), obs=chips)
```

Benefits: Partial pooling, shrinkage, better predictions!

The Power of Hierarchical Models

Shrinkage Effect

```
Unpooled:     [9.3, 6.0, 9.6, 8.9, 12.0]
                ↓    ↓    ↓    ↓     ↓
Hierarchical: [9.2, 6.5, 9.5, 9.0, 11.5]

Global mean: ———————— 9.15 ————————
```

Extreme estimates pulled toward global mean

Why this matters:

- More robust estimates, less overfitting
- Borrow strength across groups
- Balance of pooling and independence
- Predict new locations with fewer data

But What If... 🤔

Your model is a complex simulator!

```python
def complex_simulator(params):
    # Drift-diffusion model for decision making
    # Neural network simulation
    # Climate model
    # Agent-based economic model
    # ... 1000s of lines of code ...
    return simulated_data

# Problem: No analytical likelihood!
# P(data|params) = ???
```

Traditional PPLs: 😢 "I need an explicit likelihood formula!"

This is where most science happens!

Enter: Simulation-Based Inference (SBI)

The Problem:

- Complex simulators
- No analytical likelihood
- P(data|params) = ???

The SBI Solution:

Neural Likelihood Estiamtion (NLE):

1. Simulate (params, data) pairs
2. Train neural network
3. Use learned likelihood

```python
# Traditional: Need formula
def likelihood(data, params):
    # tractable statistical models
    return model(data, params)

# SBI: Only simulations!
def simulator(params):
    # Complex simulation
    return data

# Simulate
θ = sample_prior()
x = simulator(θ)

# Learn likelihood: normalizing flows
neural_likelihood = train(dataset)
```

The Magic: Wrapping NLE in Pyro 🎯

Step 1: Train NLE

```
# Generate training data
theta = prior.sample((1000,))
x = simulator(theta)

# Train neural likelihood
from sbi.inference import NLE
nle = NLE().append_simulations(theta, x)
estimator = nle.train()
```

Step 2: Use in Pyro

```
def sbi_pyro_model(x_o=None):
    # Prior
    theta = pyro.sample("theta", prior)

    # Use neural likelihood
    with pyro.plate("trials", n):
        dist = SBItoPyro(estimator, theta,)
        x = pyro.sample("x", dist, obs=x_o)
```

- We wrap `sbi` NLE object into a `pyro` distribution
- `SBItoPyro` wrapper: Class with 150 lines, mostly shape handling for `pyro`

Comparison: Standard Pyro vs SBI-Pyro

Standard Pyro

```python
def cookie_model(locations, chips):
    # Hyperpriors
    mu = pyro.sample("mu",
                    dist.Gamma(2, 0.2))
    sigma = pyro.sample("sigma",
                       dist.Exponential(1))

    # Location rates
    with pyro.plate("location", 5):
        lam = pyro.sample("lam",
            dist.Gamma(mu**2/sigma**2,
                      mu/sigma**2))

    # Explicit statistical model
    # ↓ Need to know this!
    with pyro.plate("data", len(chips)):
        pyro.sample("obs",
                   dist.Poisson(lam[locations]),
                   obs=chips)
```

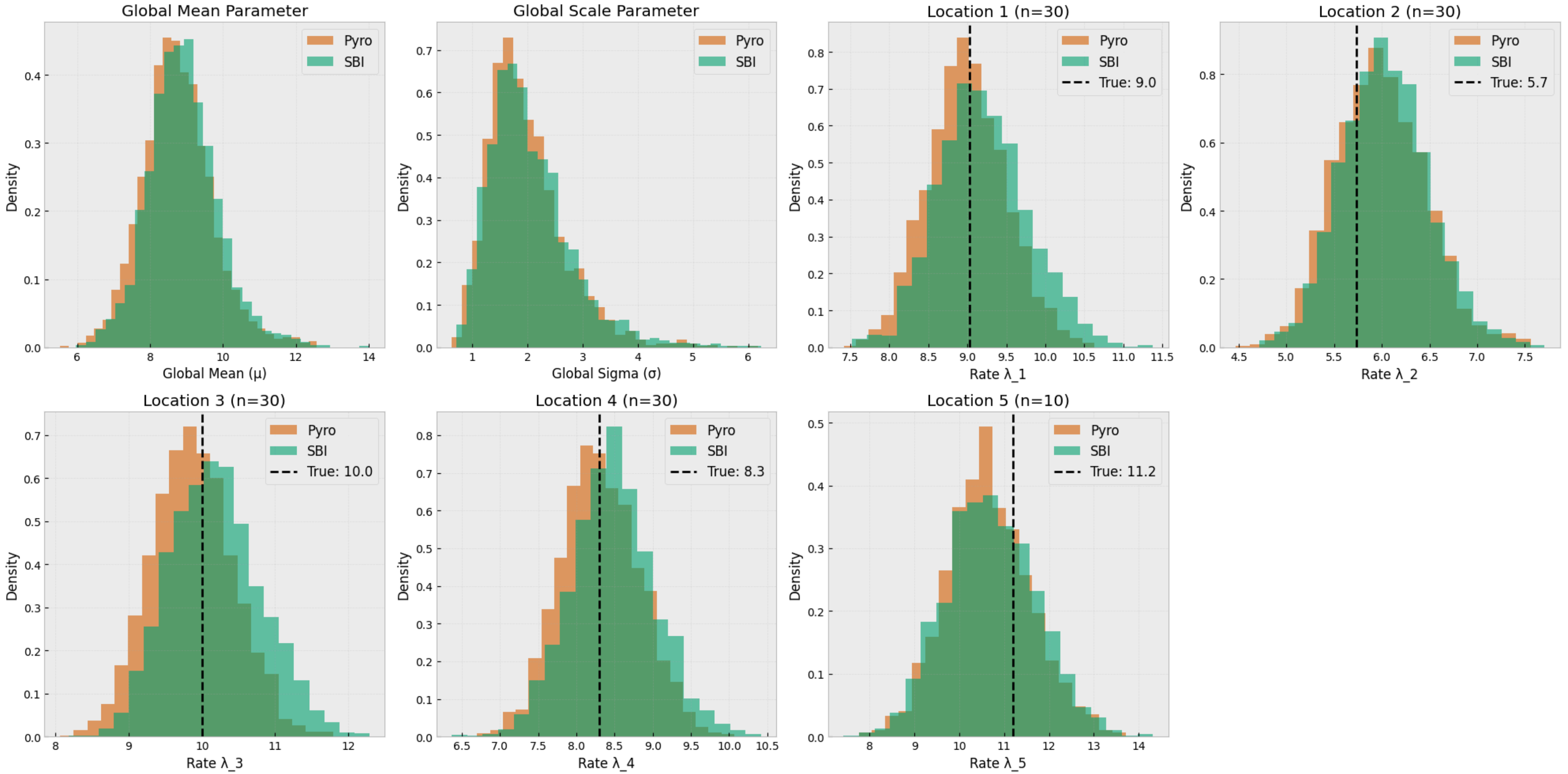
SBI + Pyro

```python
def sbi_cookie_model(locations, chips):
    # Hyperpriors (same!)
    mu = pyro.sample("mu",
                    dist.Gamma(2, 0.2))
    sigma = pyro.sample("sigma",
                       dist.Exponential(1))

    # Location rates (same!)
    with pyro.plate("location", 5):
        lam = pyro.sample("lam",
            dist.Gamma(mu**2/sigma**2,
                      mu/sigma**2))

    # Black-box neural likelihood
    with pyro.plate("data", len(chips)):
        pyro.sample("obs",
                   SBItoPyro(lam[locations]),
                    obs=chips)
```

Comparison: Pyro vs SBI posteriors for the Hierarchical Model



Real Example: Drift Diffusion Model (DDM)

DDM Equations

Evidence accumulation:

$$dx = v \cdot dt + s \cdot dW$$

- v: drift rate
- a: threshold
- z: bias
- t_0: non-decision time

Decision when $|x(t)| \geq a$

No closed-form likelihood!

Hierarchical DDM in Pyro

```python
def hierarchical_ddm(data):
    # Population level
    v_mu = pyro.sample("v_mu",
                       dist.Normal(0, 2))
    v_sigma = pyro.sample("v_sigma",
                          dist.HalfNormal(1))

    # Subject level
    with pyro.plate("subjects", n_subj):
        v = pyro.sample("v",
            dist.Normal(v_mu, v_sigma))

        # Trial level (SBI likelihood)
        with pyro.plate("trials", n_trials):
            pyro.sample("obs",
                DDMLikelihood(nle, v),
                obs=data)
```

Practical Considerations

When to use this approach:

✅ Complex simulators without tractable likelihoods

✅ Hierarchical/grouped data structure

✅ Multiple experimental conditions

Challenges to consider:

⚠️ Simulation budget (10K-100K simulations)

⚠️ Neural network training time

⚠️ Validation and diagnostics crucial

Rule of thumb: If you can write the likelihood, use standard Pyro. If not, add SBI!

Applications Across Domains

Cognitive Science

- Decision-making models (DDM, race models)
- Attention and memory models

Epidemiology

- Agent-based disease spread models
- Network effects and interventions

Economics

- Agent-based market models
- Behavioral economics experiments

Business

- Marketing models
- A-B testing
- Demand forecasting

Key insight 💡: We can now apply `pyro` as before, but with intractable models!

Key Takeaways

1. **Probabilistic programming** makes Bayesian inference accessible
    - Write model as code, get inference for free
2. **Hierarchical models** capture structure in grouped data
    - Partial pooling, shrinkage, robustness
3. **SBI** enables inference when likelihoods are intractable
    - Treat simulator as black-box, learn from simulations
4. **Pyro + SBI** combines both strengths:
    - Pyro's elegant hierarchical modeling
    - SBI's ability to handle any simulator

Acknowledgments

Cookie Example:

- Juan Camilo Orduz: juanitorduz.github.io/cookies_example_numpyro/

Pyro-SBI Bridge Implementation:

- Seth Axen 🪓 - Implementation during SBI Hackathon 2025

Communities:

- `sbi` community and contributors
- `pyro` community and developers
- EuroScipy 2025 conference organizers

TransferLab

Job Offering:

AI Research Engineer

We are looking for an AI Research Engineer to design, develop, and deploy software and GenAI applications, test and benchmark algorithms, create training materials, contribute to open-source projects, participate in hackathons, and provide practical solutions across AI/ML projects.

- We have a booth outside in the hall
- There will be a Python Quiz
- Win a **mechanical keyboard**! 🤗