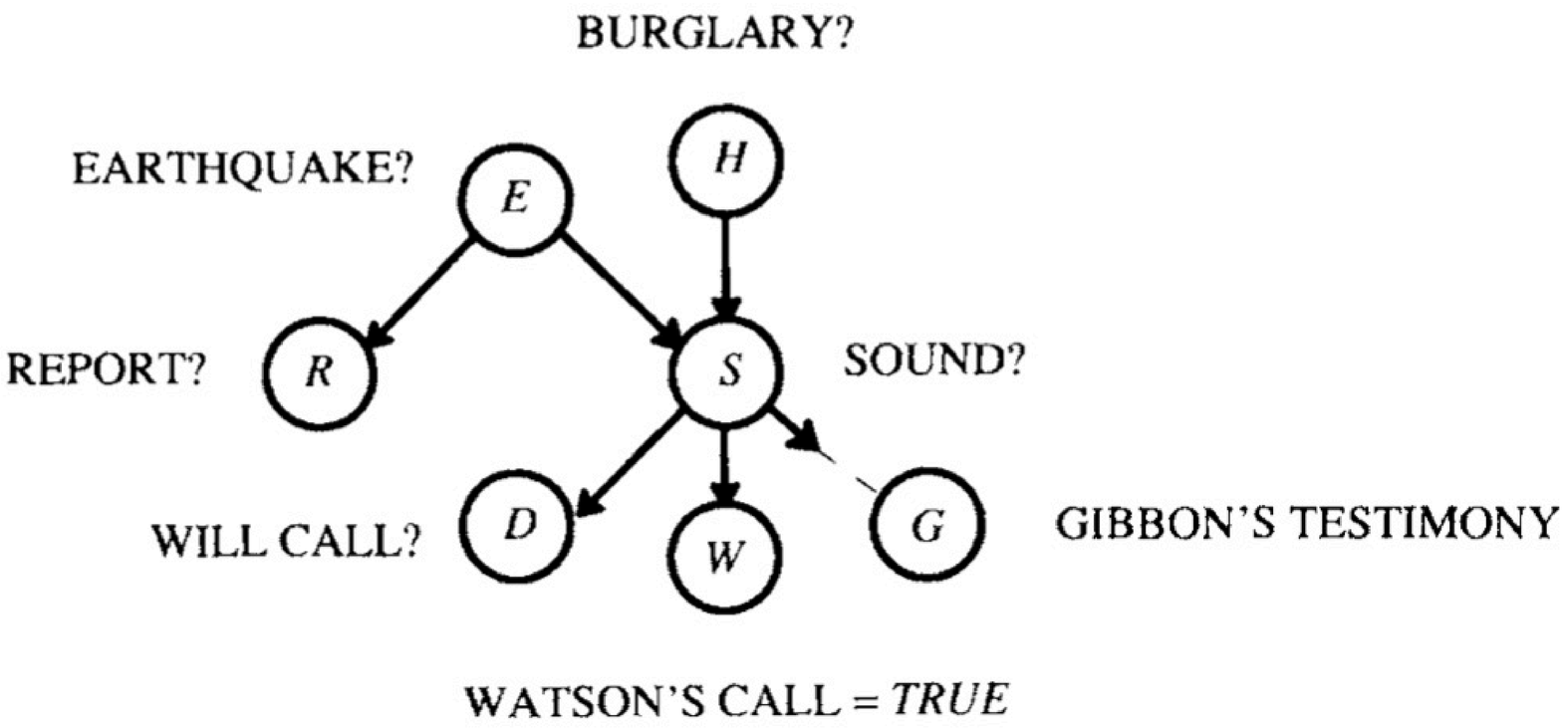
Graphical Models and Simulation-Based Inference

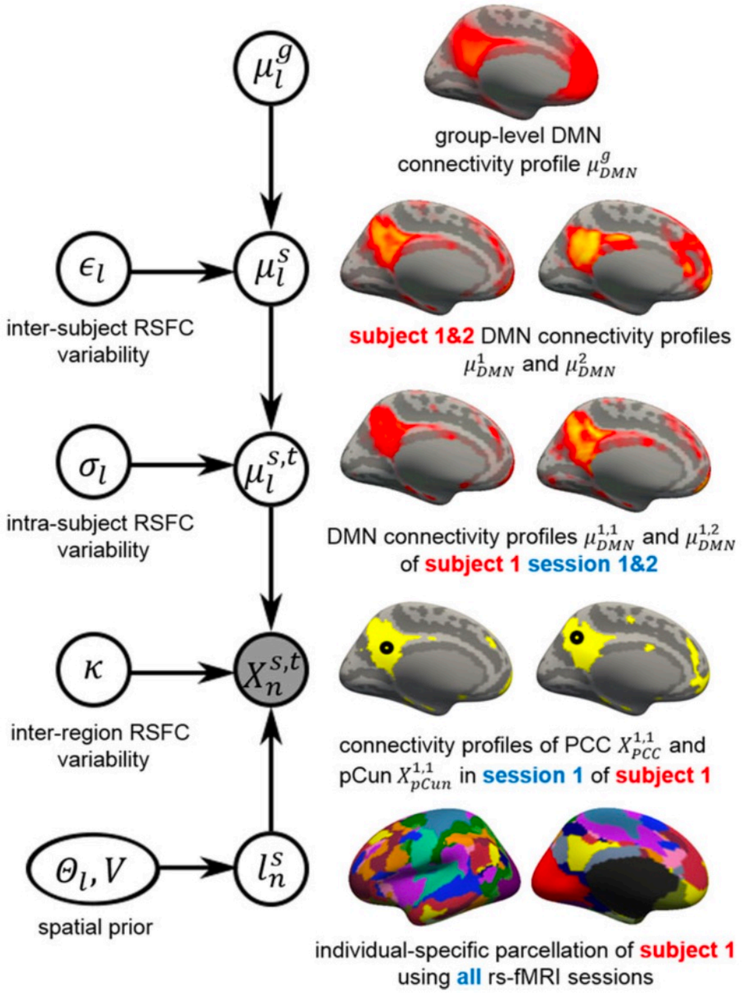
Graphical Models: Discrete Inference and Learning

Demian Wassermann 2025

Introduction to DAG and their relationship with Probability Functions (Pearl)

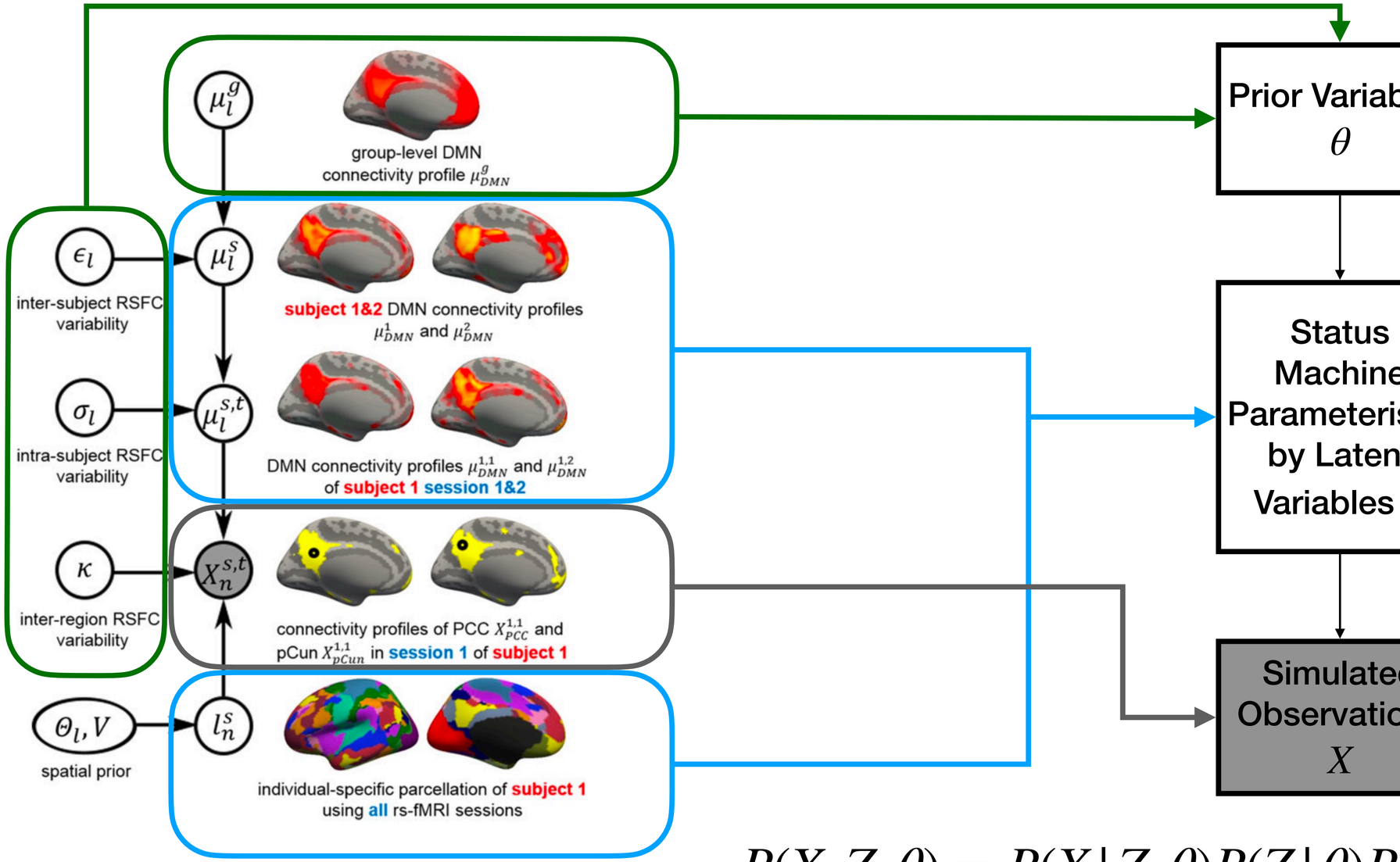


[Pearl 1987]



[Kong et al 2019]

Graphical Models and Simulation Systems



Prior Variables θ

$\theta \sim P(\theta)$

Status Machine Parameterised by Latent Variables Z

$Z \sim P(Z|\theta)$

Simulated Observations X

$X \sim P(X|Z,\theta)$

$$P(X,Z,\theta) = P(X|Z,\theta)P(Z|\theta)P(\theta) = P(X|Z)P(Z|\theta)P(Z)$$

General Inference Notation

θ: parameters X: observations

Likelihood Prior

$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}$$

Posterior

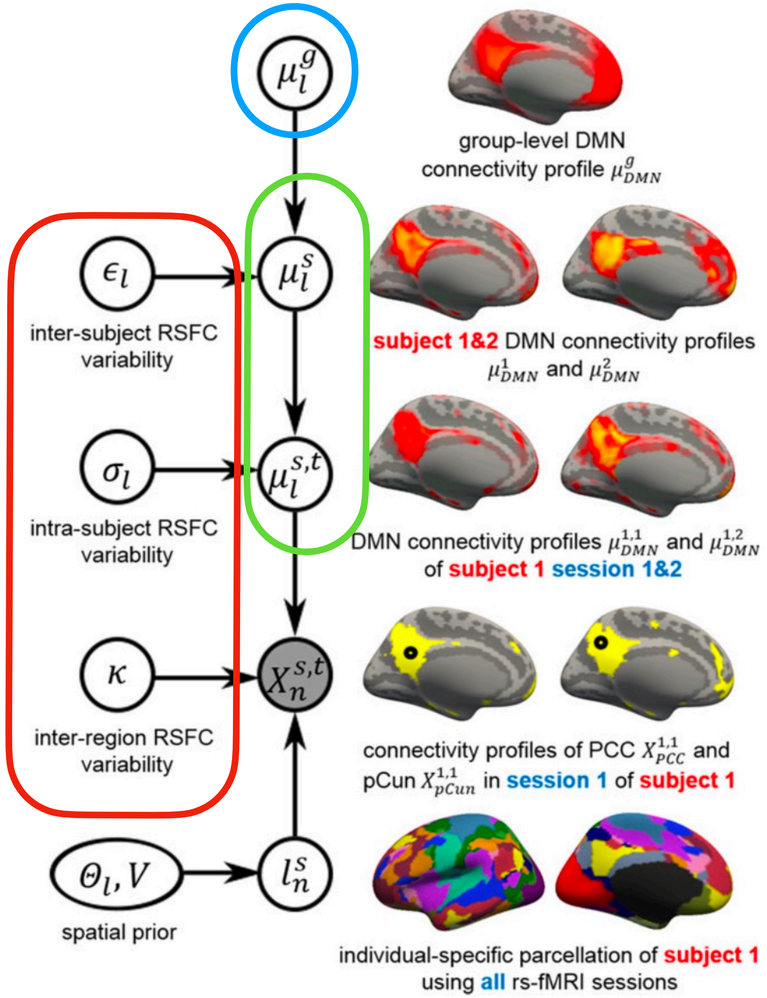
Evidence

Z: latent random variables

$$P(\theta \mid X) = \frac{\mathbb{E}_Z[P(X \mid Z, \theta)]P(\theta)}{P(X)}$$

η: nuisance random variables

$$P(\theta \mid X) = \frac{\mathbb{E}_\eta[P(X \mid \theta, \eta)]P(\theta)}{P(X)}$$



General Inference Notation

θ: parameters X: observations

Likelihood Prior

$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}$$

Posterior

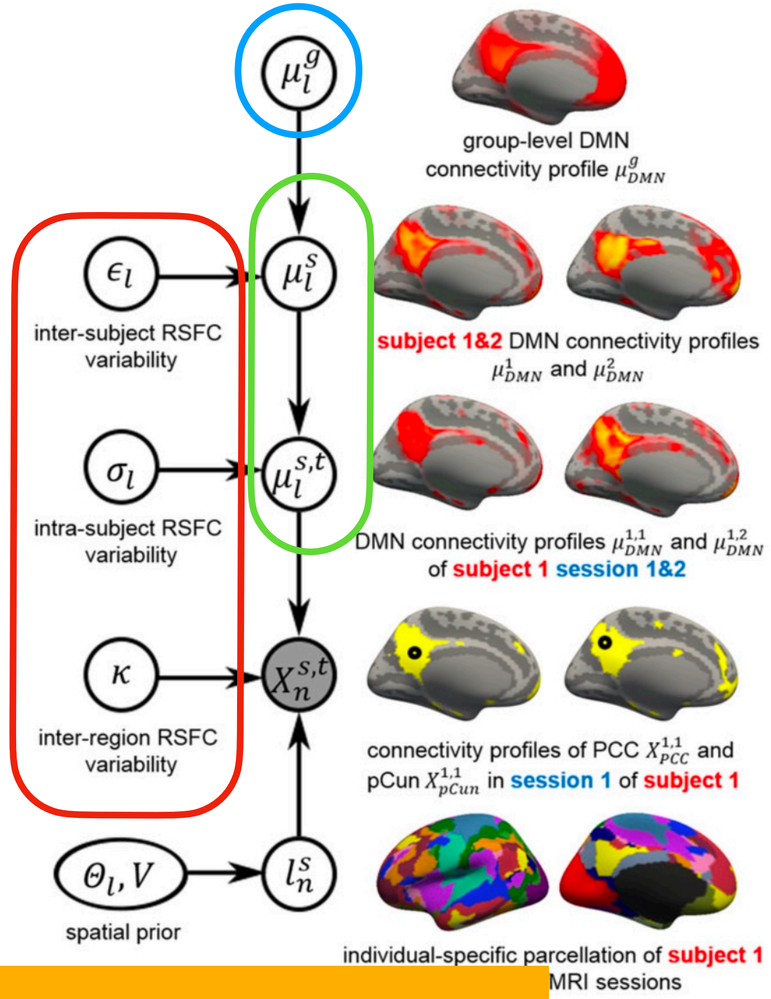
Evidence

Z: latent random variables

$$P(\theta \mid X) = \frac{\mathbb{E}_Z[P(X \mid Z, \theta)]P(\theta)}{P(X)}$$

η: nuisance random variables

$$P(\theta \mid X) = \frac{\mathbb{E}_\eta[P(X \mid \theta, \eta)]P(\theta)}{P(X)}$$



Intractable in general:
full likelihood impossible to evaluated or computation cost is extremely high

Likelihood computation is hard: Enter Mechanistic, Example Models Galton Board

θ: parameters X: observations

Likelihood Prior

$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}$$

Posterior

Evidence

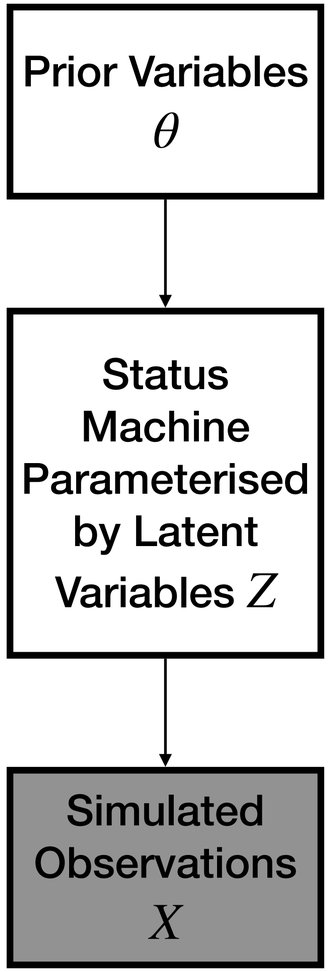
Z: latent random variables

$$P(\theta \mid X) = \frac{\mathbb{E}_Z[P(X \mid Z, \theta)]P(\theta)}{P(X)}$$

η: nuisance random variables

$$P(\theta \mid X) = \frac{\mathbb{E}_\eta[P(X \mid \theta, \eta)]P(\theta)}{P(X)}$$

Simulation-Based Inference



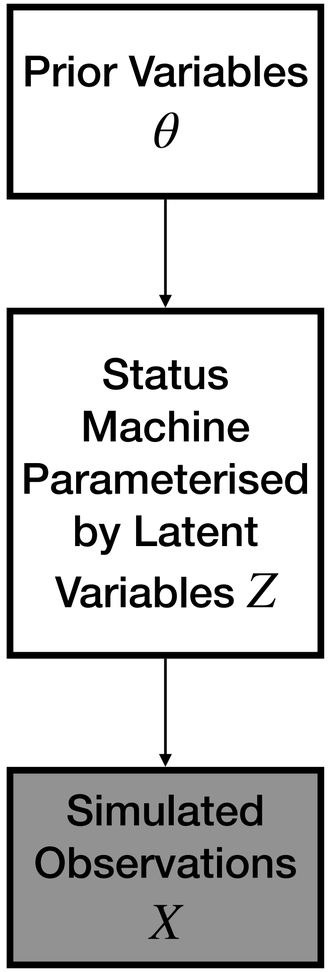
$\theta \sim P(\theta)$

$Z \sim P(Z \mid \theta)$

$X \sim P(X \mid Z, \theta)$

- Inference is defined as finding the θ that could be at the origin of an observation X. Specifically computing $P(\theta \mid X) = \mathbb{E}_Z[P(\theta, Z \mid X)]$
- For this, we use Bayes $P(\theta, Z \mid X) = \frac{P(X \mid Z, \theta)P(Z, \theta)}{P(X)}$, nonetheless the likelihood $P(X \mid Z, \theta)$ is often unknown or intractable.
- Hence simulation-based inference either approximates or eliminates the need for an explicit likelihood by simulating observations.

Simulation-Based Inference: Neural Network Approximations



$\theta \sim P(\theta)$

$Z \sim P(Z \mid \theta)$

$X \sim P(X \mid Z, \theta)$

θ: parameters X: observations

Likelihood Prior

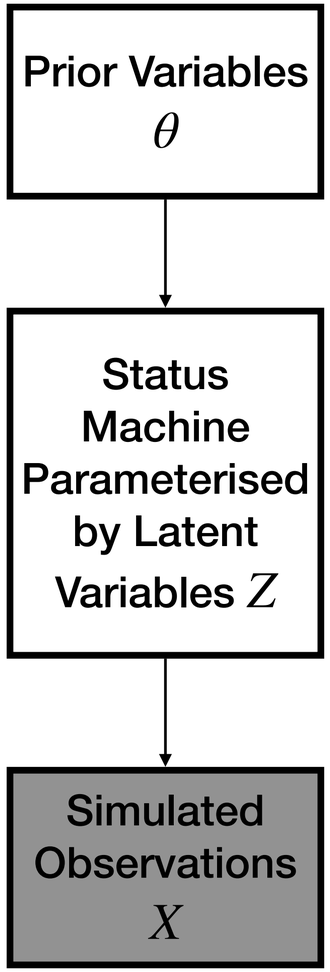
$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}$$

Posterior

Evidence

- $P(\theta \mid X)$ approximated through “Neural Posterior” estimators
- $P(X \mid \theta)$ approximated through “Neural Likelihood” estimators
- $\frac{P(X \mid \theta)}{P(X)}$ approximated through the “Neural ratio” estimators

Simulation-Based Inference: Why now it works (Cranmer et al 2019)



$$\theta \sim P(\theta)$$

$$Z \sim P(Z \mid \theta)$$

$$X \sim P(X \mid Z, \theta)$$

Likelihood Prior

θ: parameters X: observations

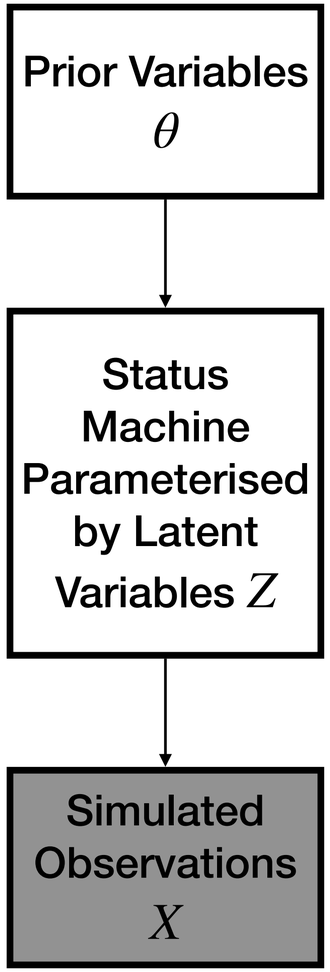
$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}$$

Posterior

Evidence

- Novel ML-based approaches allow us to massively generate simulated observations
- Autodifferentiation and neural network approaches are great non-linear function estimators
- Active learning can help improving sampling efficiency much better than Markov Chains

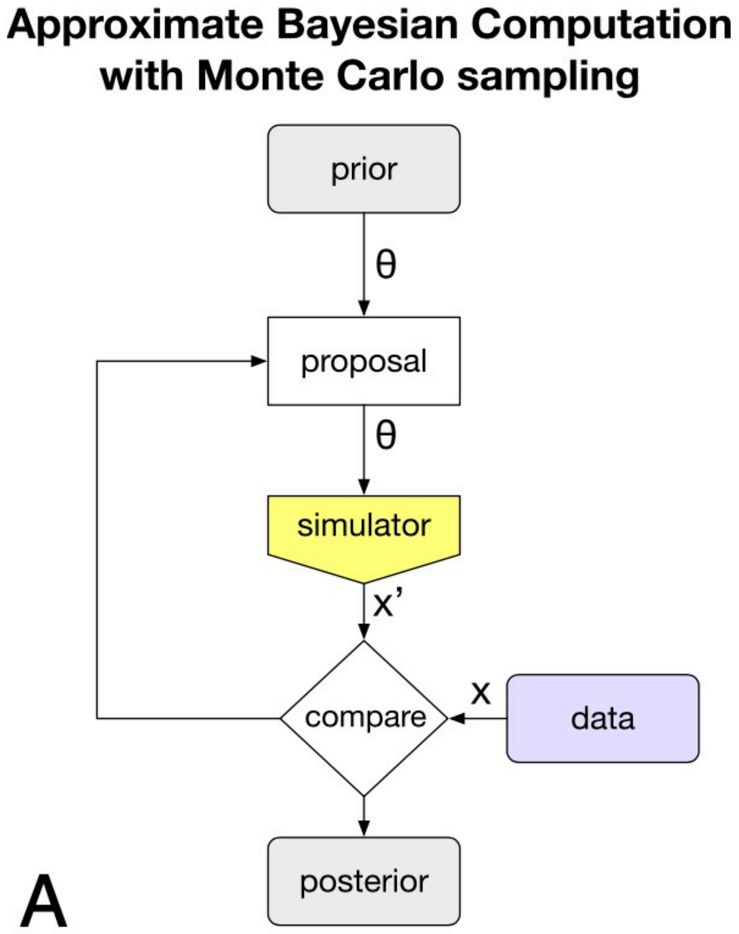
Simulation-Based Inference: Approximate Bayesian MC



$$\theta \sim P(\theta)$$

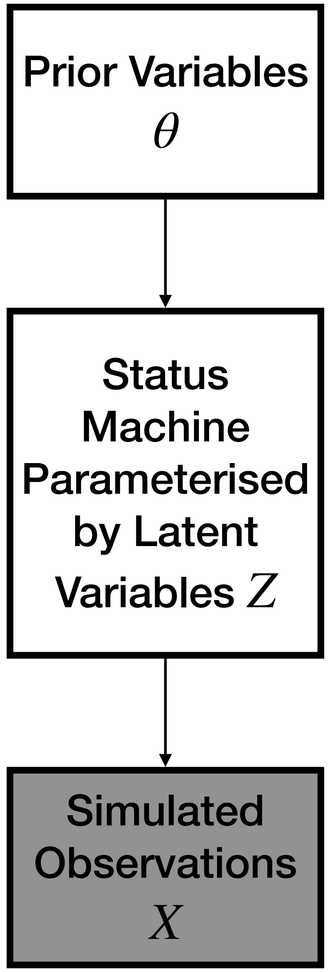
$$Z \sim P(Z \mid \theta)$$

$$X \sim P(X \mid Z, \theta)$$



(Cranmer et al 2019)

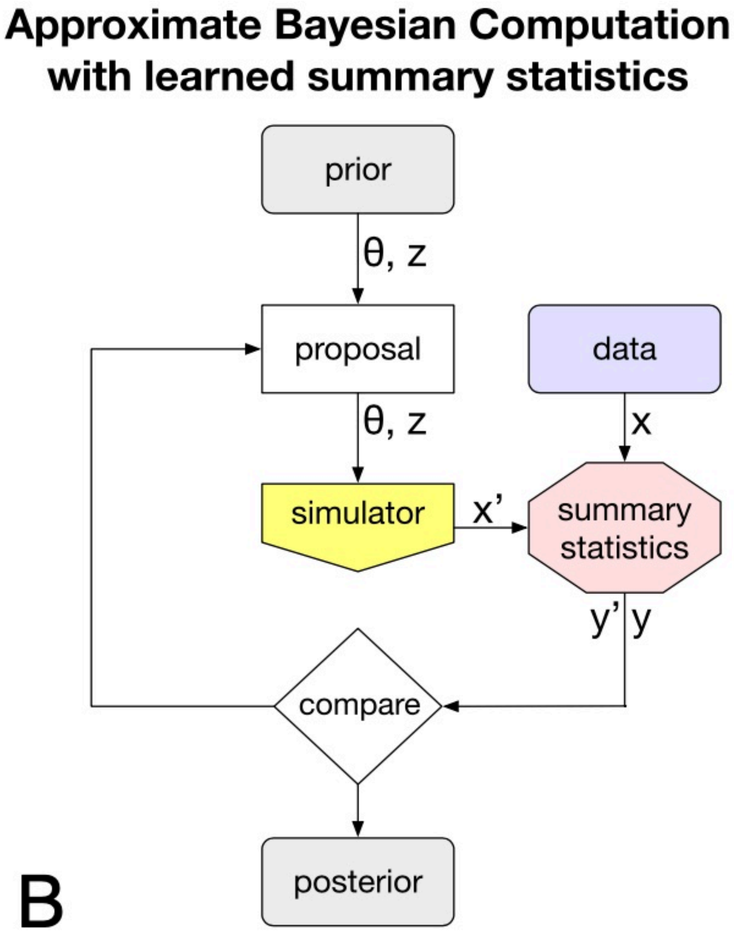
Simulation-Based Inference: Approximate Bayesian MC



$$\theta \sim P(\theta)$$

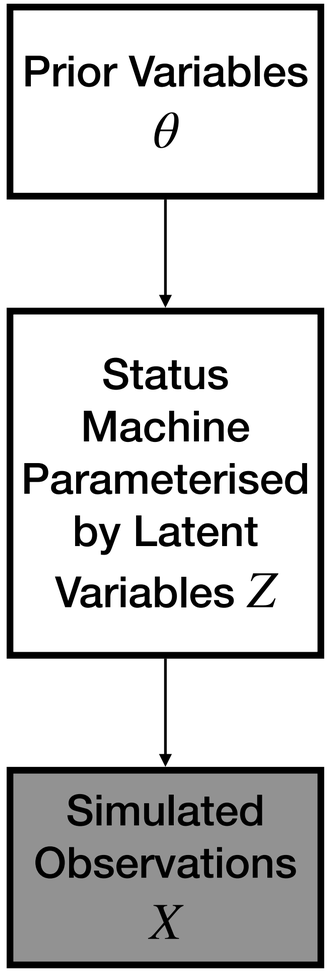
$$Z \sim P(Z \mid \theta)$$

$$X \sim P(X \mid Z, \theta)$$



(Cranmer et al 2019)

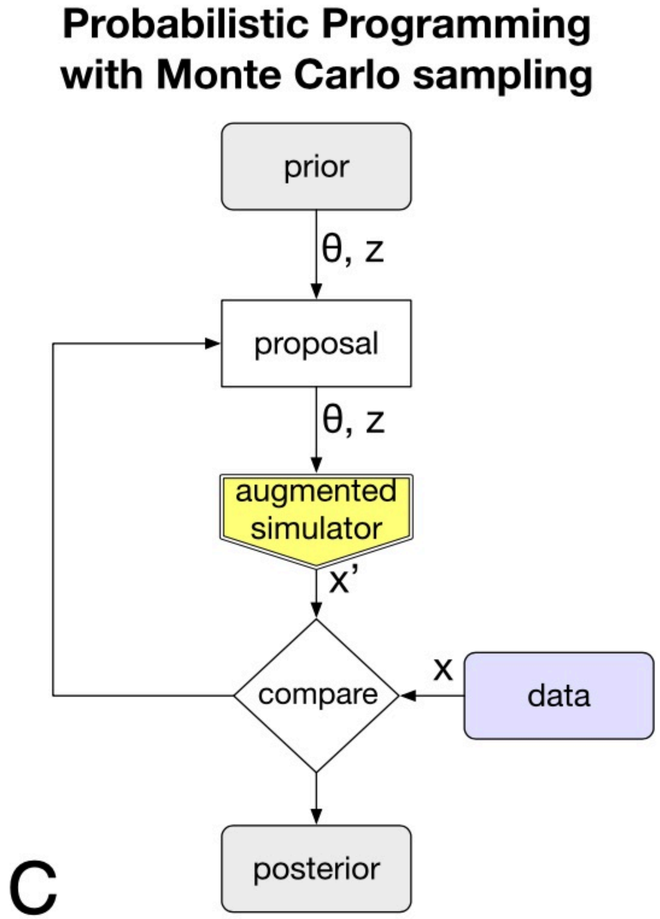
Simulation-Based Inference: Approximate Bayesian MC



$$\theta \sim P(\theta)$$

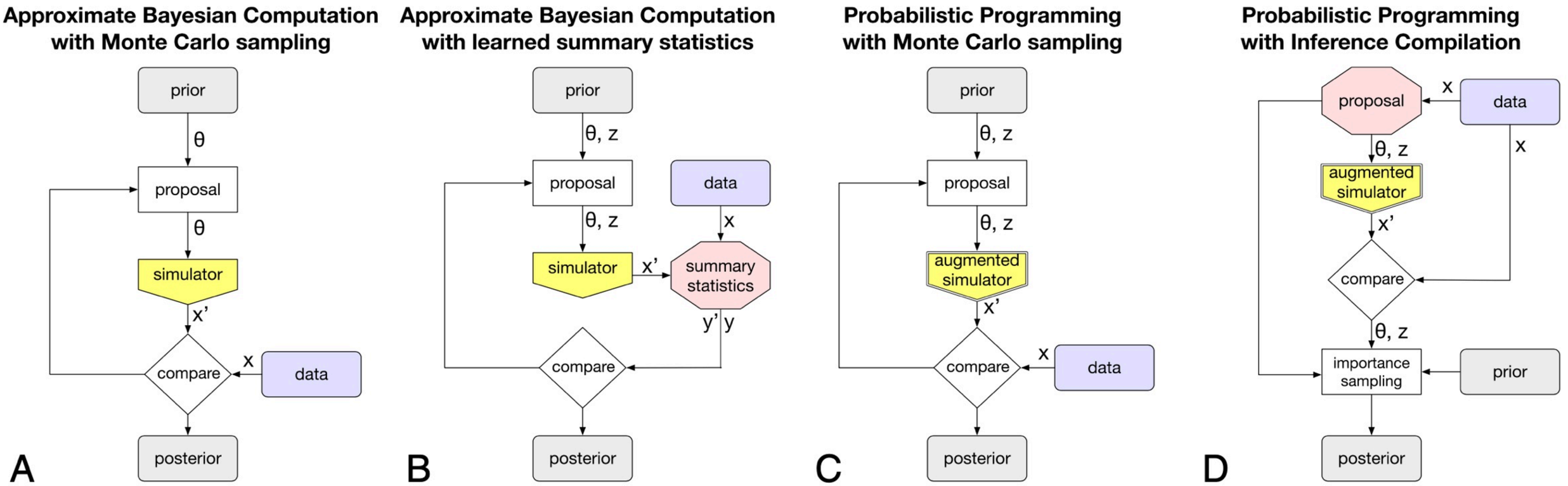
$$Z \sim P(Z \mid \theta)$$

$$X \sim P(X \mid Z, \theta)$$



(Cranmer et al 2019)

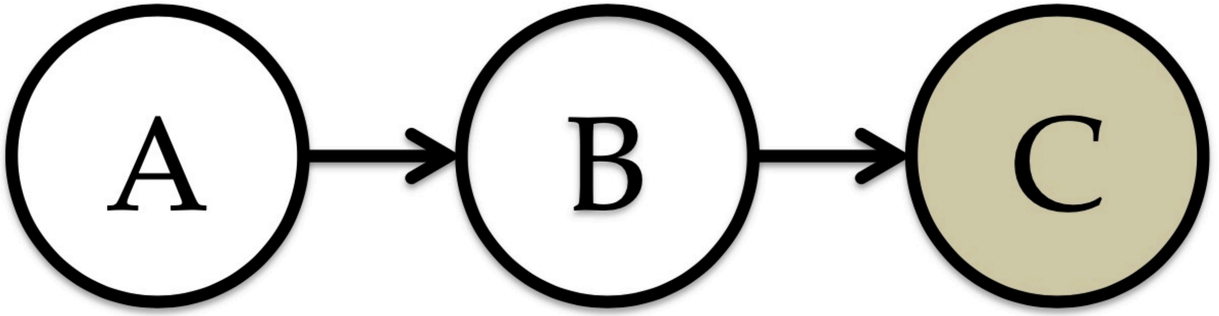
Simulation-Based Inference: Approximate Bayesian MC



(Cranmer et al 2019)

Simulation-Based Inference: Amortization

Population Sample Observation

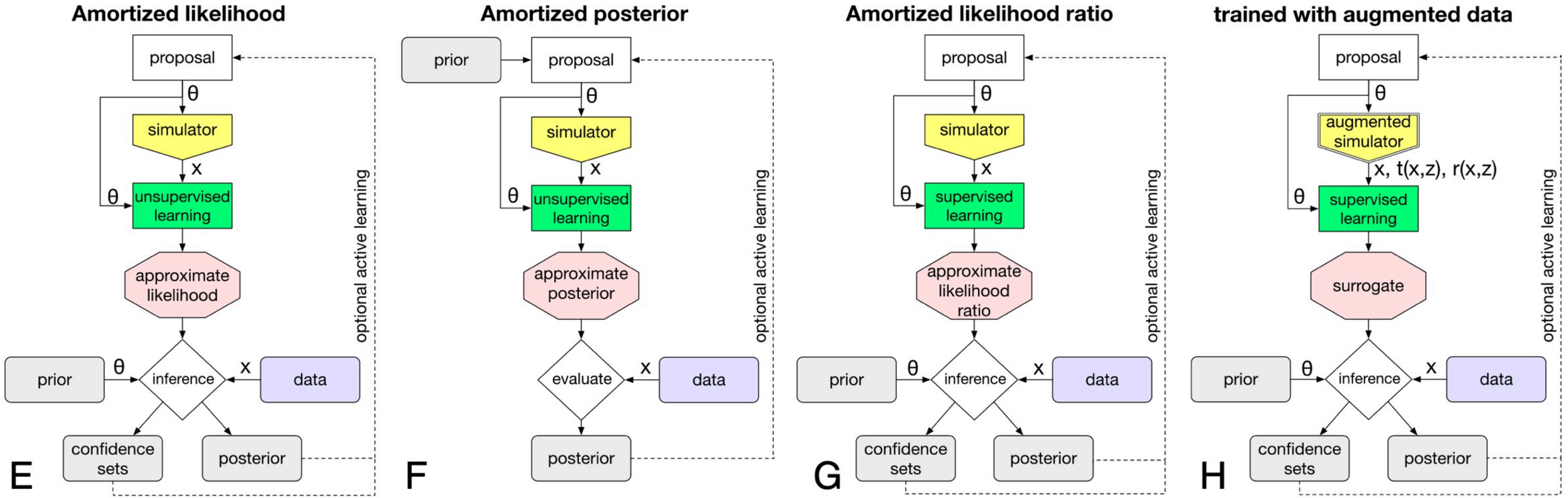


Query 1: $P(B|C) = P(C|B)P(B)/P(C)$

Query 2: $P(A|C) = \sum_B P(A|B)P(B|C)$

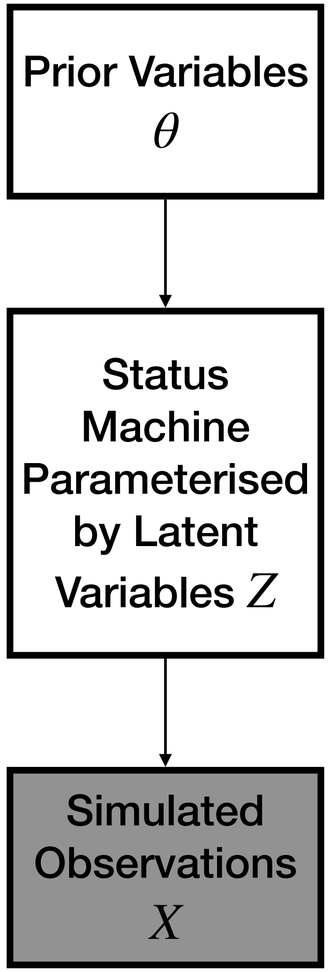
(Gershman et al 2014

Simulation-Based Inference: Amortisation Techniques



(Cranmer et al 2019)

Simulation-Based Inference: Neural Network Approximations



$\theta \sim P(\theta)$

$Z \sim P(Z \mid \theta)$

$X \sim P(X \mid Z, \theta)$

θ: parameters X: observations

Likelihood Prior

$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}$$

Posterior

Evidence

- $P(\theta \mid X)$ approximated through “Neural Posterior” estimators
- $P(X \mid \theta)$ approximated through “Neural Likelihood” estimators
- $\frac{P(X \mid \theta)}{P(X)}$ approximated through the “Neural ratio” estimators

Simulation-Based Inference: Neural Network Approximations Through Stochastic Flows

θ: parameters X: observations

Likelihood Prior

$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}$$

Posterior

Evidence

$$f(X, \theta) = N_{\mu,\Sigma}(\phi(X, \theta)) \, | \, J_{\phi}(X, \theta) \, |$$

f the Neural estimator and ϕ the stochastic flow

- $P(\theta \mid X)$ approximated through “Neural Posterior” estimators
- $P(X \mid \theta)$ approximated through “Neural Likelihood” estimators
- $\frac{P(X \mid \theta)}{P(X)}$ approximated through the “Neural ratio” estimators

Simulation-Based Inference: Automatic Posterior Transformation (Greenberg et al 2019)

θ: parameters X: observations

Likelihood Prior

$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}$$

Posterior

Evidence

- $P(\theta \mid X)$ approximated through "Neural posterior" by a flow $Q_{F(x_0,\phi)}(\theta)$
- Loss function:

$$\tilde{q}_{x,\phi}(\theta) = q_{F(x,\phi)}(\theta)\frac{\tilde{p}(\theta)}{p(\theta)}\frac{1}{Z(x,\phi)}, \qquad (2)$$

- Where a proposal posterior is

$$\tilde{p}(\theta|x) = p(\theta|x)\frac{\tilde{p}(\theta)\ p(x)}{p(\theta)\ \tilde{p}(x)}$$

Algorithm 1 APT with per-round proposal updates

Input: simulator with (implicit) density $p(x|\theta)$, data x_o, prior $p(\theta)$, density family q_ψ, neural network $F(x, \phi)$, simulations per round N, number of rounds R.

$\tilde{p}_1(\theta) := p(\theta)$
for $r = 1$ **to** R **do**
  for $j = 1$ **to** N **do**
    Sample $\theta_{r,j} \sim \tilde{p}_r(\theta)$
    Simulate $x_{r,j} \sim p(x|\theta_{r,j})$
  end for
  $\phi \leftarrow \underset{\phi}{\text{argmin}} \sum_{i=1}^{r}\sum_{j=1}^{N} -\log \tilde{q}_{x_{i,j},\phi}(\theta_{i,j})$ using (2)
  $\tilde{p}_{r+1}(\theta) := q_{F(x_o,\phi)}(\theta)$
end for
return $q_{F(x_o,\phi)}(\theta)$

Simulation-Based Inference: Sequential Neural Likelihood (Papamakarios et al 2019)

θ: parameters X: observations

Likelihood Prior

$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}$$

Posterior

Evidence

- $P(X \mid \theta)$ approximated through "Neural likelihood" by a flow $Q_\phi(X \mid \theta)$

Algorithm 1: Sequential Neural Likelihood (SNL)

Input : observed data $\mathbf{x}_o$, estimator $q_{\boldsymbol{\phi}}(\mathbf{x} \mid \boldsymbol{\theta})$, number of rounds R, simulations per round N

Output: approximate posterior $\hat{p}(\boldsymbol{\theta} \mid \mathbf{x}_o)$

set $\hat{p}_0(\boldsymbol{\theta} \mid \mathbf{x}_o) = p(\boldsymbol{\theta})$ and $\mathcal{D} = \{\}$
for $r = 1 : R$ **do**
  for $n = 1 : N$ **do**
    sample $\boldsymbol{\theta}_n \sim \hat{p}_{r-1}(\boldsymbol{\theta} \mid \mathbf{x}_o)$ with MCMC
    simulate $\mathbf{x}_n \sim p(\mathbf{x} \mid \boldsymbol{\theta}_n)$
    add $(\boldsymbol{\theta}_n, \mathbf{x}_n)$ into $\mathcal{D}$
  (re-)train $q_{\boldsymbol{\phi}}(\mathbf{x} \mid \boldsymbol{\theta})$ on $\mathcal{D}$ and set
  $\hat{p}_r(\boldsymbol{\theta} \mid \mathbf{x}_o) \propto q_{\boldsymbol{\phi}}(\mathbf{x}_o \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})$
return $\hat{p}_R(\boldsymbol{\theta} \mid \mathbf{x}_o)$

Simulation-Based Inference: Neural Ratio (Hermans et al 2020)

θ: parameters X: observations

Likelihood Prior

$$P(\theta \mid X) = \frac{P(X \mid \theta)P(\theta)}{P(X)}$$

Posterior

Evidence

- $P(X \mid \theta)/P(X)$ approximated through "Neural ratio" by a flow $d_\phi(X \mid \theta)$

Algorithm 1 Optimization of $\mathbf{d}_\phi(\mathbf{x}, \boldsymbol{\theta})$.

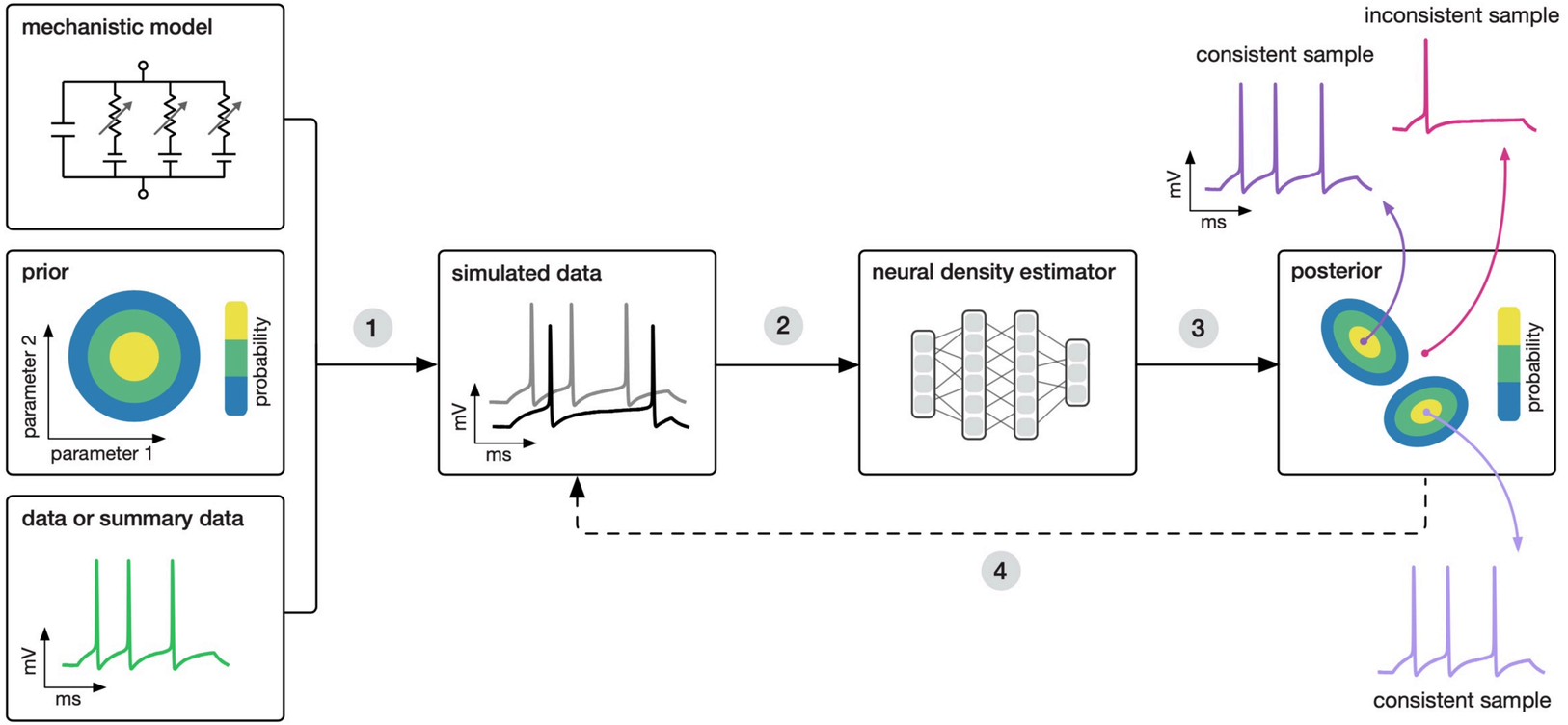
Inputs:	Criterion ℓ (e.g., BCE)
	Implicit generative model $p(\mathbf{x} \mid \boldsymbol{\theta})$
	Prior $p(\boldsymbol{\theta})$
Outputs:	Parameterized classifier $\mathbf{d}_\phi(\mathbf{x}, \boldsymbol{\theta})$
Hyperparameters:	Batch-size M

while not converged do
  Sample $\boldsymbol{\theta} \leftarrow \{\boldsymbol{\theta}_m \sim p(\boldsymbol{\theta})\}_{m=1}^{M}$
  Sample $\boldsymbol{\theta}' \leftarrow \{\boldsymbol{\theta}'_m \sim p(\boldsymbol{\theta})\}_{m=1}^{M}$
  Simulate $\mathbf{x} \leftarrow \{\mathbf{x}_m \sim p(\mathbf{x} \mid \boldsymbol{\theta}_m)\}_{m=1}^{M}$
  $\mathcal{L} \leftarrow \ell(\mathbf{d}_\phi(\mathbf{x}, \boldsymbol{\theta}),\ 1) + \ell(\mathbf{d}_\phi(\mathbf{x}, \boldsymbol{\theta}'),\ 0)$
  $\phi \leftarrow \textbf{OPTIMIZER}(\phi,\ \nabla_\phi \mathcal{L})$
end while
return $\mathbf{d}_\phi$

RESEARCH ARTICLE

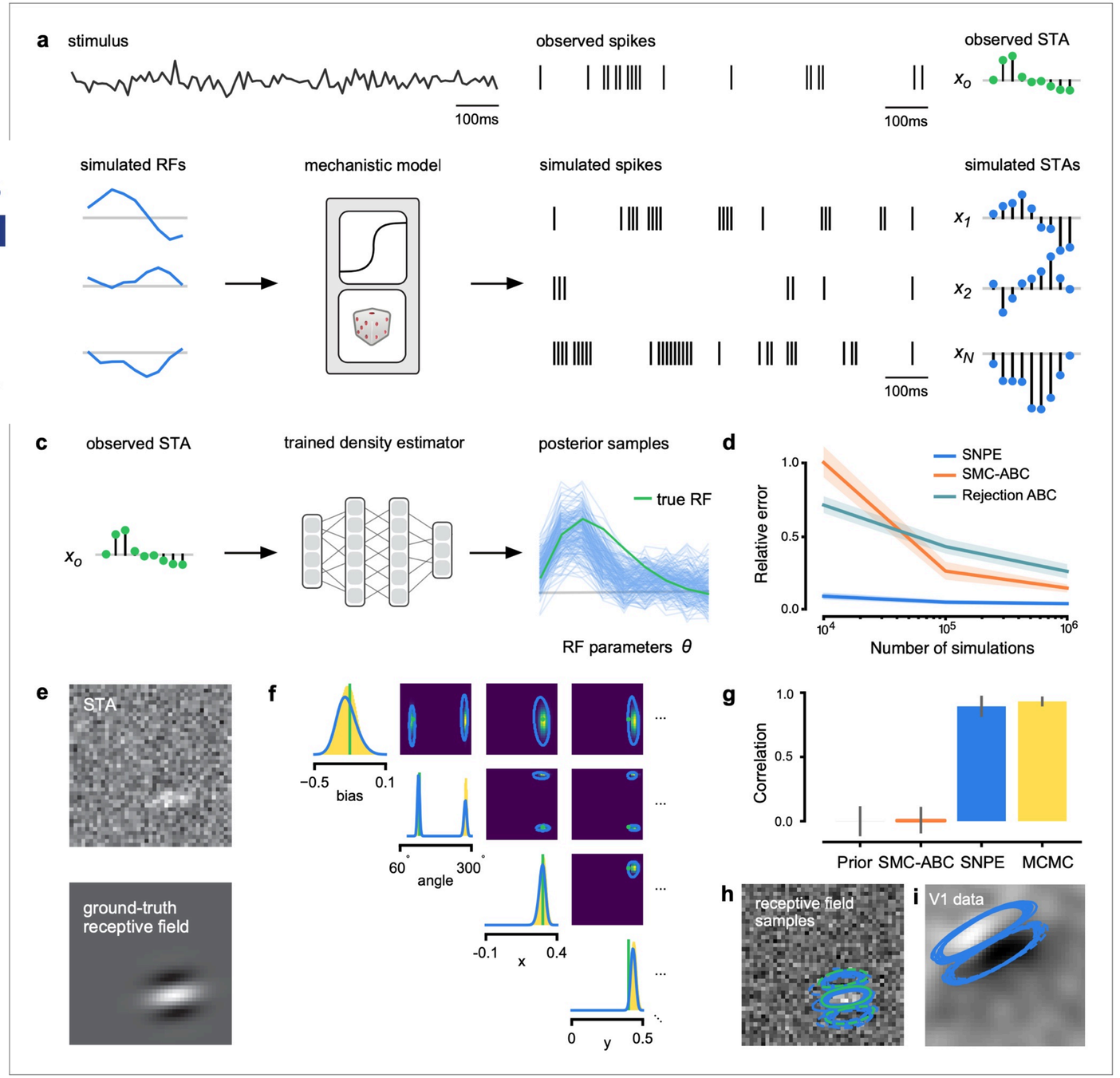
Training deep neural density estimators to identify mechanistic models of neural dynamics

Pedro J Gonçalves[1,2†*], Jan-Matthis Lueckmann[1,2†*], Michael Deistler[1,3†*], Marcel Nonnenmacher[1,2,4], Kaan Öcal[2,5], Giacomo Bassetto[1,2], Chaitanya Chintaluri[6,7], William F Podlaski[6], Sara A Haddad[8], Tim P Vogels[6,7], David S Greenberg[1,4], Jakob H Macke[1,2,3,9*]



Training deep neural density estimators to identify mechanistic models of neural dynamics

Pedro J Gonçalves[1,2†*], Jan-Matthis Lueckmann[1,2†*], Michael Deistler[1,3†*], Marcel Nonnenmacher[1,2,4], Kaan Öcal[2,5], Giacomo Bassetto[1,2], Chaitanya Chintaluri[6,7], William F Podlaski[6], Sara A Haddad[8], Tim P Vogels[6,7], David S Greenberg[1,4], Jakob H Macke[1,2,3,9*]

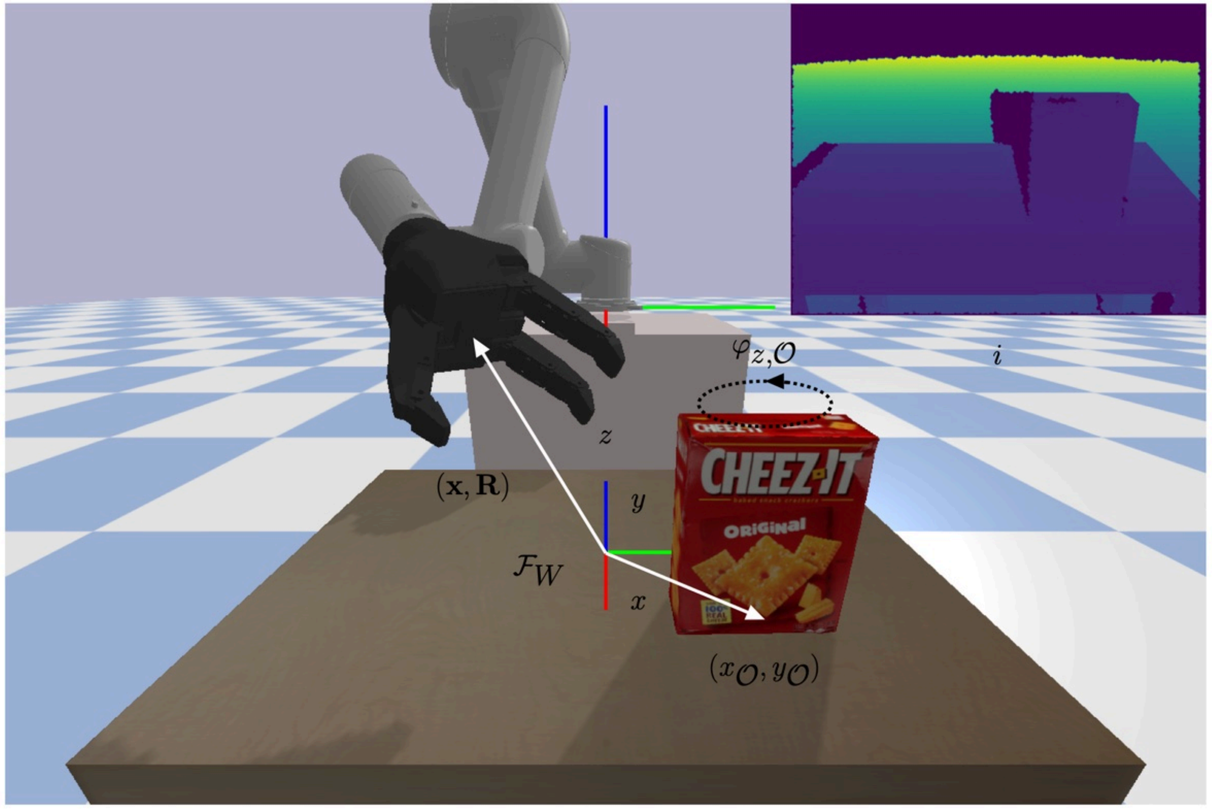


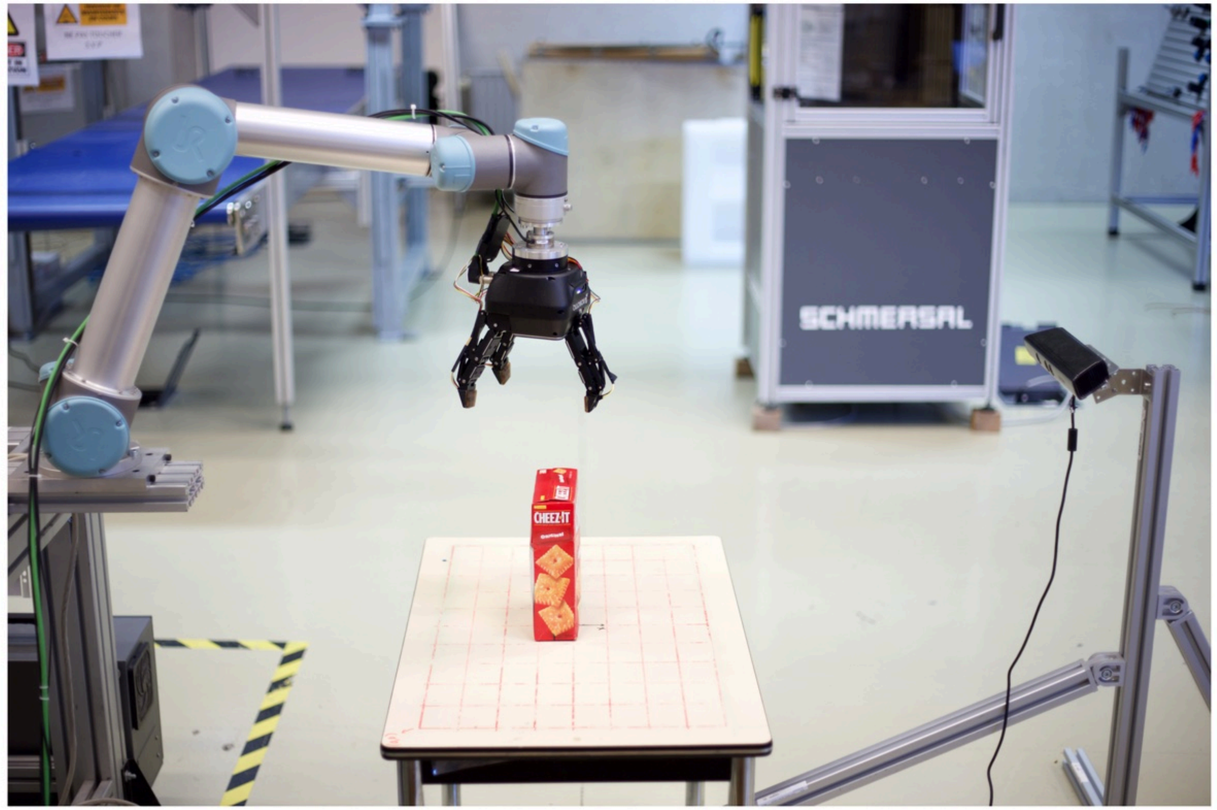
SIMULATION-BASED BAYESIAN INFERENCE FOR MULTI-FINGERED ROBOTIC GRASPING

Norman Marlier
University of Liège
norman.marlier@uliege.be

Olivier Brüls
University of Liège
o.bruls@uliege.be

Gilles Louppe
University of Liège
g.louppe@uliege.be



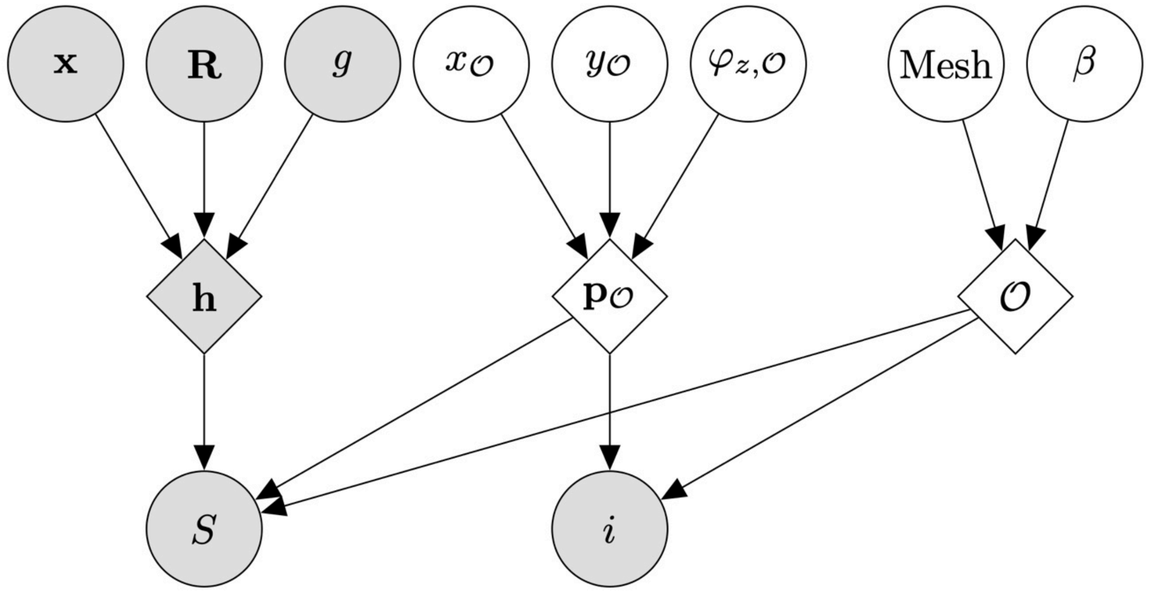


SIMULATION-BASED BAYESIAN INFERENCE FOR MULTI-FINGERED ROBOTIC GRASPING

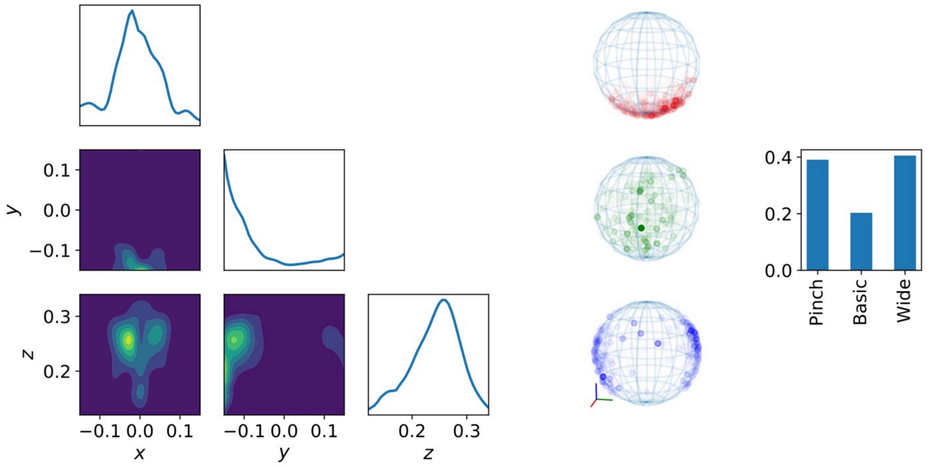
Norman Marlier
University of Liège
norman.marlier@uliege.be

Olivier Brüls
University of Liège
o.bruls@uliege.be

Gilles Louppe
University of Liège
g.louppe@uliege.be



(a)



Variable	Prior
x	uniform$(-0.15, 0.15)$
y	uniform$(-0.15, 0.15)$
z	uniform$(0.12, 0.34)$
$\mathbf{R}$	mixture of power spherical(μ_i, κ)
g	categorical$(\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\})$
$x_\mathcal{O}$	uniform$(-0.05, 0.05)$
$y_\mathcal{O}$	uniform$(-0.05, 0.05)$
$\varphi_{z,\mathcal{O}}$	uniform$(-\pi, \pi)$
Mesh	uniform in the set of objects
β	uniform(0.9, 1.1)

(b)

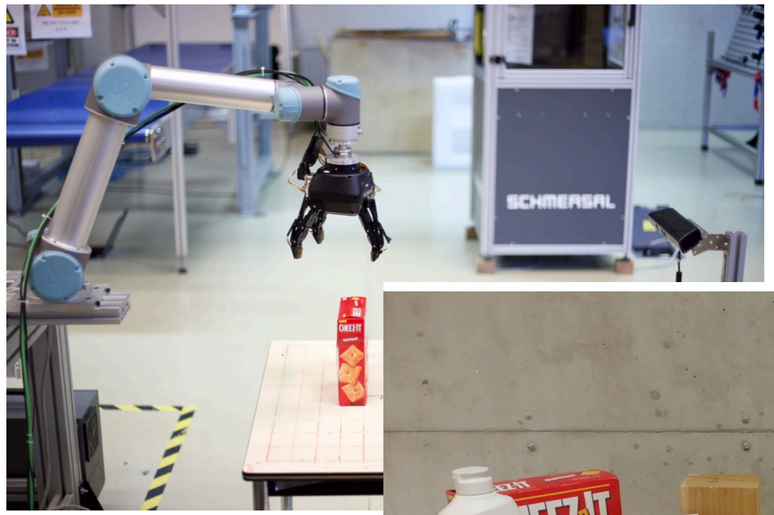
Figure 2: (a) Probabilistic graphical model of the environment. Gray nodes correspond to observed variables and white nodes to unobserved variables. (b) Prior distributions.

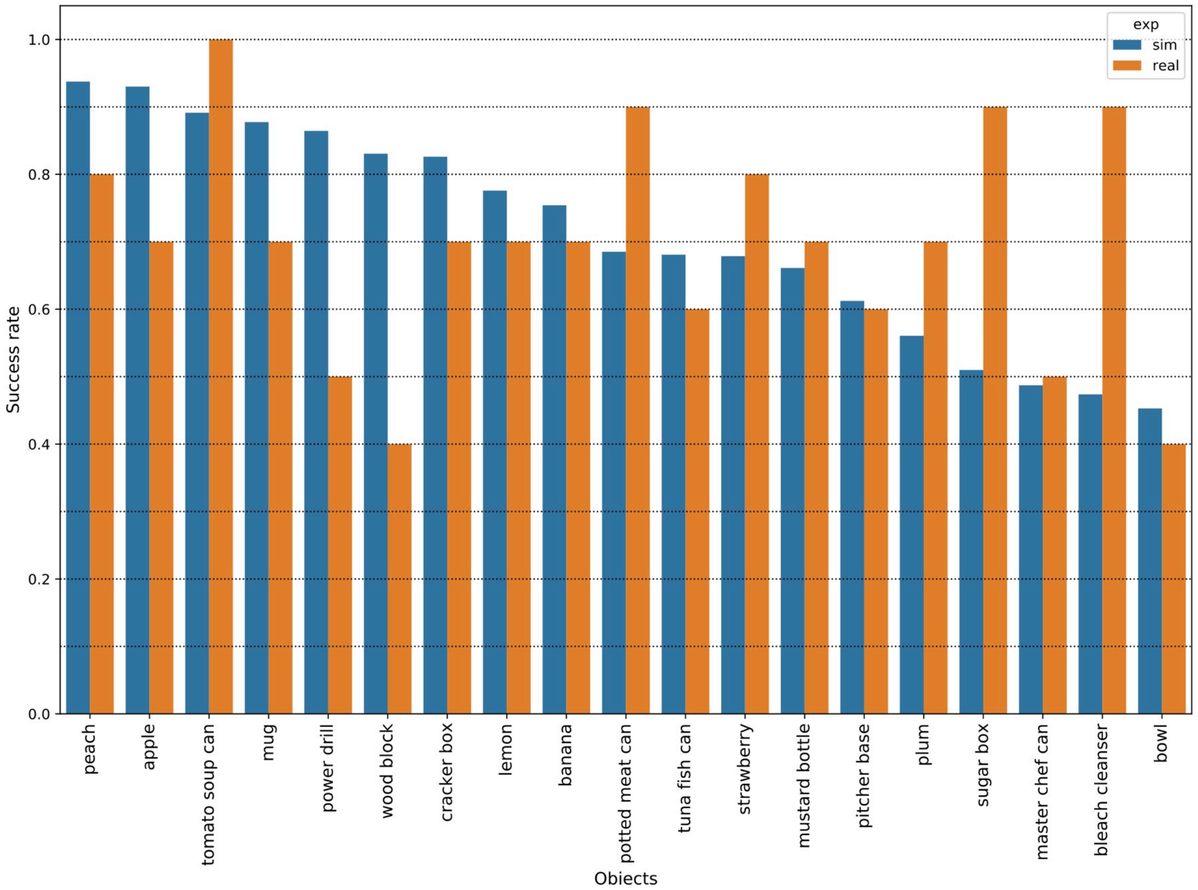
SIMULATION-BASED BAYESIAN INFERENCE FOR MULTI-FINGERED ROBOTIC GRASPING

Norman Marlier
University of Liège
norman.marlier@uliege.be

Olivier Brüls
University of Liège
o.bruls@uliege.be

Gilles Louppe
University of Liège
g.louppe@uliege.be





Current Uses of Simulation-Based Inference Tabular Foundation Models

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › articles › article

Article | Open access | Published: 08 January 2025

Accurate predictions on small data with a tabular foundation model

Noah Hollmann ✉, Samuel Müller ✉, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister & Frank Hutter ✉

Nature **637**, 319–326 (2025) | Cite this article

441k Accesses | **475** Citations | **518** Altmetric | Metrics

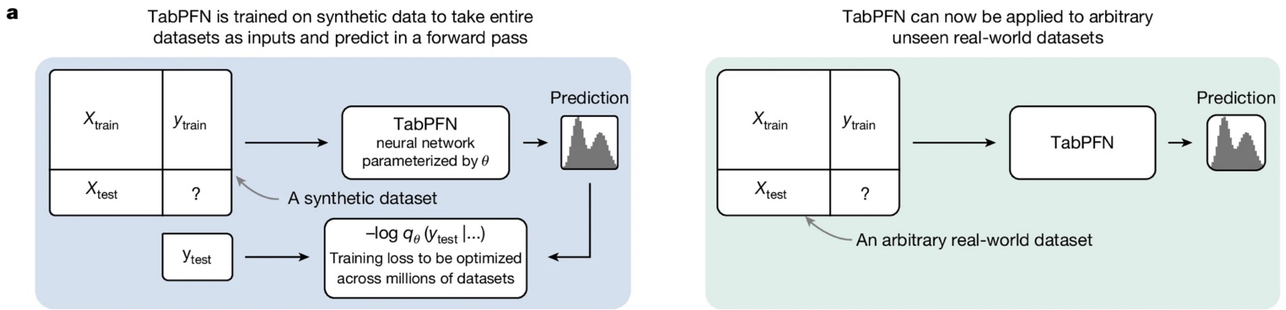
arXiv > cs > arXiv:2207.01848

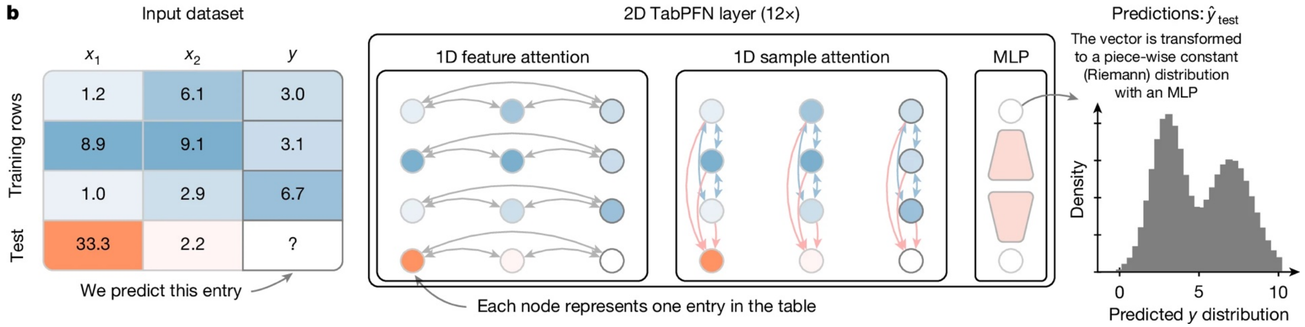
Computer Science > Machine Learning

[Submitted on 5 Jul 2022 (v1), last revised 16 Sep 2023 (this version, v6)]

TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second

Noah Hollmann, Samuel Müller, Katharina Eggensperger, Frank Hutter





Current Uses of Simulation-Based Inference Tabular Foundation Models

- The problem:
  Why tabular classification is hard; fitting a new model per dataset is expensive and brittle.

- PFN Framework
  Prior-Data Fitted Networks: approximating the Bayesian Posterior Predictive Distribution offline.

- Graphical Model Prior
  Structural Causal Models (SCMs) and BNNs as data generators — the backbone of TabPFN's world model.

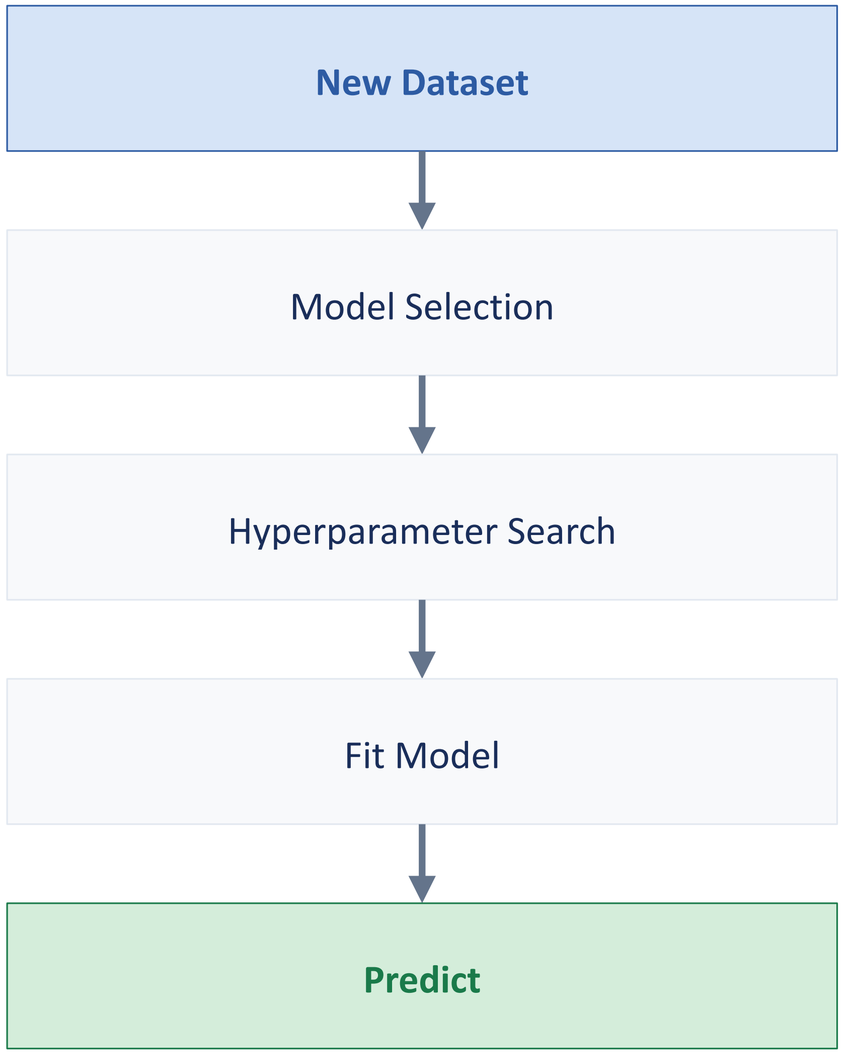
- Simulation-Based Inference
  Generating millions of synthetic datasets from the prior; training the transformer to invert the simulator.

- In-Context Learning
  How a single transformer forward pass replaces dataset-specific fitting at inference time.

- Results & Insights
  What the causal/SCM prior buys us; performance vs. boosted trees and AutoML.

The Problem to be Solved

Each new dataset requires fitting from scratch — expensive and hard to tune



COSTLY

⏱ Time
AutoML can take hours per dataset for hyperparameter tuning

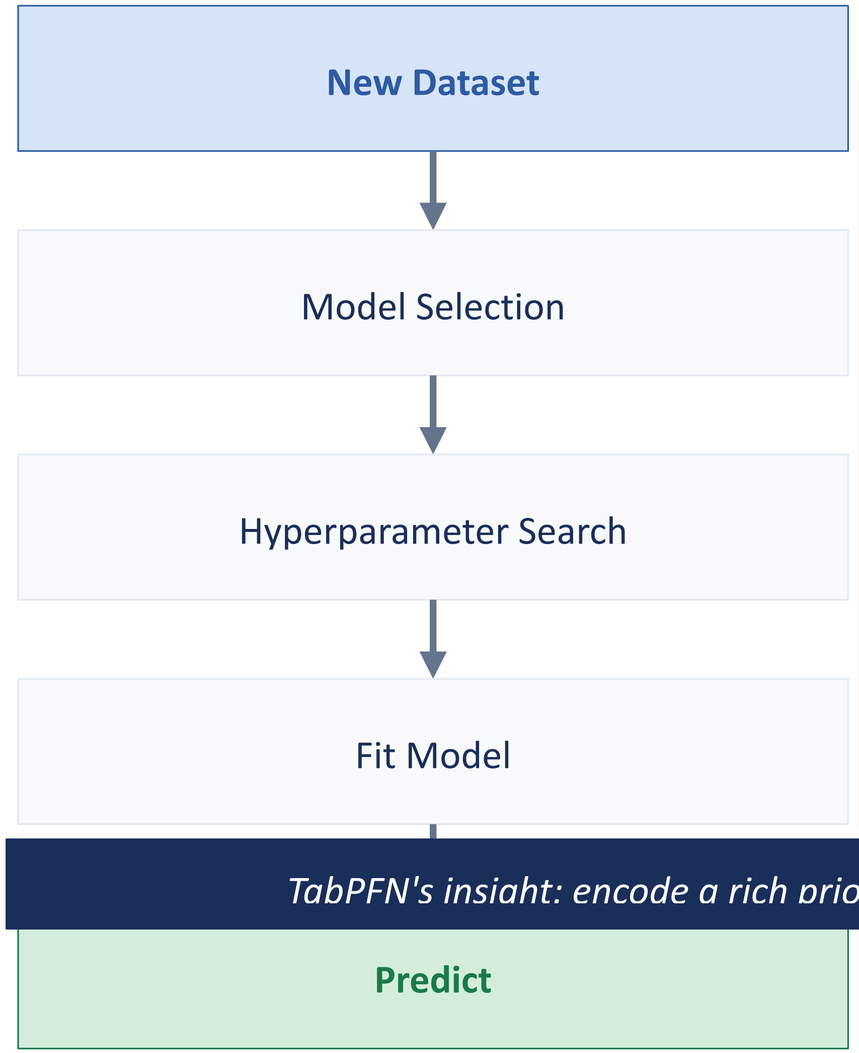
🔧 Effort
Model selection and preprocessing decisions require expertise

📉 Small Data
DL methods struggle when N < 1000; gradient boosting dominates

🔁 Repetition
The same fitting procedure repeats for every new dataset

The Problem to be Solved

Each new dataset requires fitting from scratch — expensive and hard to tune



COSTLY

⏱ **Time**
AutoML can take hours per dataset for hyperparameter tuning

🔧 **Effort**
Model selection and preprocessing decisions require expertise

📉 **Small Data**
DL methods struggle when N < 1000; gradient boosting dominates

🔁 **Repetition**
The same fitting procedure repeats for every new dataset

TabPFN's insight: encode a rich prior offline, then replace fitting with a neural network forward pass

Prior-Data Fitted Networks: Amortized Bayesian Inference

Learn to approximate the Posterior Predictive Distribution (PPD) across a whole prior

$$p(y^* \mid x^*, D_{train}) = \int p(y^* \mid x^*, \phi) \cdot p(\phi \mid D_{train})\, d\phi$$

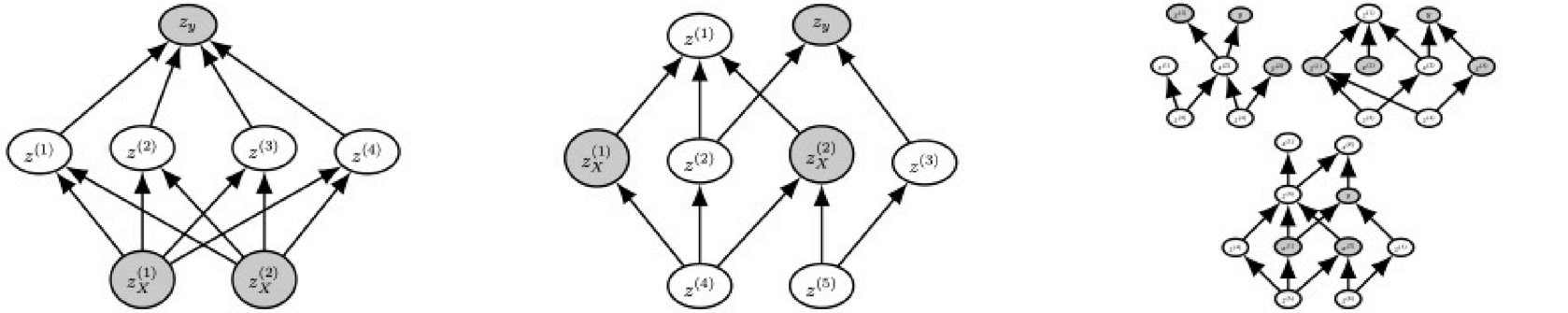
PPD: target posterior *likelihood* *parameter posterior* *marginalise over φ*

Classical Bayes	PFN (TabPFN) Approach
1. Fix prior p(ϕ)	1. Design prior p(ϕ) as a simulator
2. Observe D_train	2. Sample datasets D ~ p(D|ϕ), ϕ ~ p(ϕ)
3. Compute posterior p(ϕ|D_train)	3. Train transformer: (D_train, x*) → ŷ
4. Marginalize analytically / MCMC	4. Amortizes the integral over all ϕ
5. → prediction for x*	5. → prediction = one forward pass
Expensive per dataset	***Fast at inference; expensive only once offline***

VS

Key insight: the transformer weight vector ϑ ≡ an amortized representation of the full posterior over the prior's hypothesis space

Prior-Data Fitted Networks: Amortized Bayesian Inference

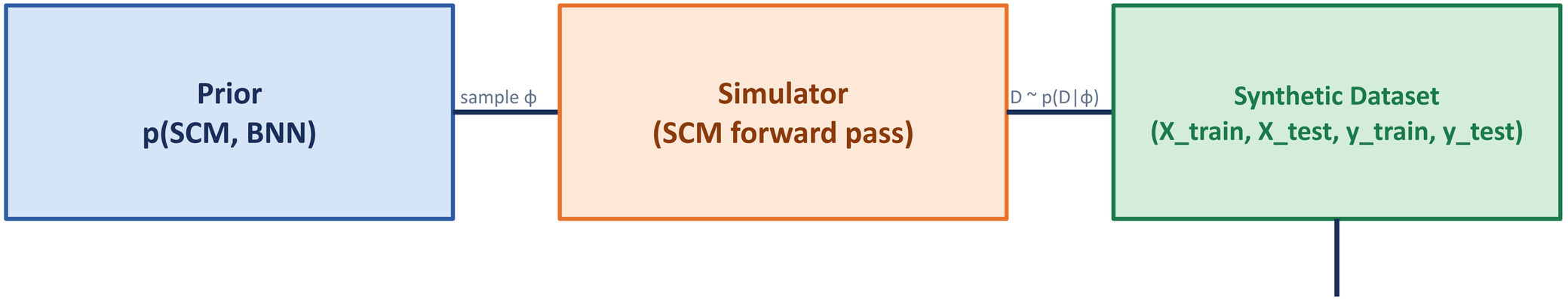


(a) A BNN (b) An SCM (c) SCMs sampled from the prior

Figure 2: Overview of graphs generating data in our prior. Inputs x are mapped to the output y through unobserved nodes z. Plots based on Müller et al. (2022).

Simulation-Based Inference (SBI): The Training Paradigm

TabPFN is trained like a neural posterior estimator — but applied to supervised prediction tasks



minimise CE loss
across all synthetic D

Transformer Training Loss

$$L(\theta) = E_{D\sim prior} [- \log q_\theta(y_{test} \mid X_{train}, y_{train}, X_{test})]$$

TabPFN vs. Classical SBI

	Classical SBI (e.g. NPE/SNPE)	TabPFN
Goal	Estimate p(θ|x_obs)	Estimate p(y*|x*, D_train)
Simulator	Scientific forward model	SCM/BNN generative prior
Observed data	Summary statistics of real experiment	Labeled training set D_train
Amortization	Over observations x_obs	Over all datasets in prior

Inside the Prior: SCMs + Bayesian Neural Networks

Two complementary mechanisms generate the diversity of synthetic datasets

Structural Causal Model (SCM) Path

- Sample random DAG G ~ Erdős–Rényi(N, P)
- Each edge: random nonlinear function $f_{i\ j}$

  (MLP, linear, multiplicative, step, …)
- Root nodes seeded with noise ε ~ Uniform/Gaussian/…
- Forward-propagate all N samples through G
- Designate random subset of nodes as features X
- Remaining node(s) become target Y
- Implicit causal structure: X ⊥⊥ Y | pa(Y)

+

Bayesian Neural Network (BNN) Path

- Sample a random MLP architecture

  (depth, width, activation drawn from priors)
- Sample weights W ~ N(0, σ^2) per layer
- Generate inputs X ~ some distribution
- Compute f_W(X) = regression output
- Complementary coverage: smooth function classes

  that may not arise naturally from SCMs
- Adds regularity and interpolation bias to prior

Hyperprior key detail: Almost all prior hyperparameters (graph size, edge density, activation, noise scale…) are themselves drawn from distributions rather than fixed — this creates enormous diversity and prevents overfitting to any single data-generating mechanism.

Forward Simulation in Detail: How One Synthetic Dataset is Generated

Four precise steps from raw hyperparameters to a labelled (X, y) dataset ready for transformer training

1 Sample Hyperparams → **2** Build SCM Graph → **3** Propagate Samples → **4** Designate & Extract

① Hyperparameter Sample

`N (nodes)`	~ Γ(α,β), small
`P (density)`	~ Beta, sparse
`n_samples`	[50, 1000]
`noise type`	Gauss/ Uniform/…
`edge fn`	per edge sample
`activations`	per-node nonlin.
`n_classes`	in [2, 10]

② SCM Graph Construction

Erdős–Rényi DAG:

```
for i < j in V:
add edge (i→j)
with prob P
```

Per edge, sample fn:

```
MLP / Linear /
Multiplicative /
Step / Tanh
```

Topo sort for

③ Sample Propagation

For s = 1 … N_samples:

```
ε_i ~ p_noise
z_root ← ε_root
For each j (topo):
  z_j = f_j(pa(j))
        + ε_j
```

Result: N × |V|

```
matrix of node
activations
```

④ Extract & Post-process

Node designation:

```
F ⊂ V → features X
T ∈ V → target Y
```

Post-processing:

```
Quantile-transform
X → N(0,1)/U(0,1)
Bin Y → C classes
equal-frequency
```

Optional noise:

Training batch: 512 synthetic datasets × 18 000 batches ≈ 9 M unique prior samples seen by the transformer during offline training

The Edge Function Zoo: Diversity in the Simulator

Each directed edge in the SCM is independently assigned one of several random nonlinear mapping types

MLP (Neural Network)

$f(x) = W_L\ \sigma(\dots\ \sigma(W_1\ x + b_1)\dots) + b_L$

depth ~ U{1,3}; width ~ U{4,64}; W ~ N(0,1); act ∈ {ReLU, Tanh, GELU}

→ Captures arbitrary smooth nonlinear relationships between parent and child nodes.

Linear / Affine

$f(x) = a \cdot x + b, \quad a, b \sim N(0, 1)$

Slope and intercept sampled independently per edge; ensures linear subspace covered.

→ Guarantees linear functions are well-represented — prevents prior from being purely nonlinear.

Multiplicative (Interaction)

$f(x_1, x_2) = x_1 \cdot x_2 \quad \text{(multi-parent nodes)}$

Applied when a node has ≥2 parents; creates non-additive feature interaction terms.

→ Generates interaction effects ($x_1 \cdot x_2$ type) that are common in real-world tabular datasets.

Step / Threshold Function

$f(x) = 1[x > \tau], \quad \tau \sim \text{Uniform}(x_{min}, x_{max})$

Threshold τ drawn from the data range; creates piecewise-constant decision boundaries.

→ Mimics decision-rule-like structure naturally arising in categorical / rule-based processes.

Smooth Nonlinear (Tanh / Sigmoid)

$f(x) = \tanh(a \cdot x + b) \quad \text{or} \quad \sigma(a \cdot x + b)$

Gain a ~ U(0.5, 5); bias b ~ N(0, 1); saturating output keeps activations bounded.

→ Saturating functions produce bounded outputs typical of physiological or economic signals.

Noise (Root Nodes — Exogenous)

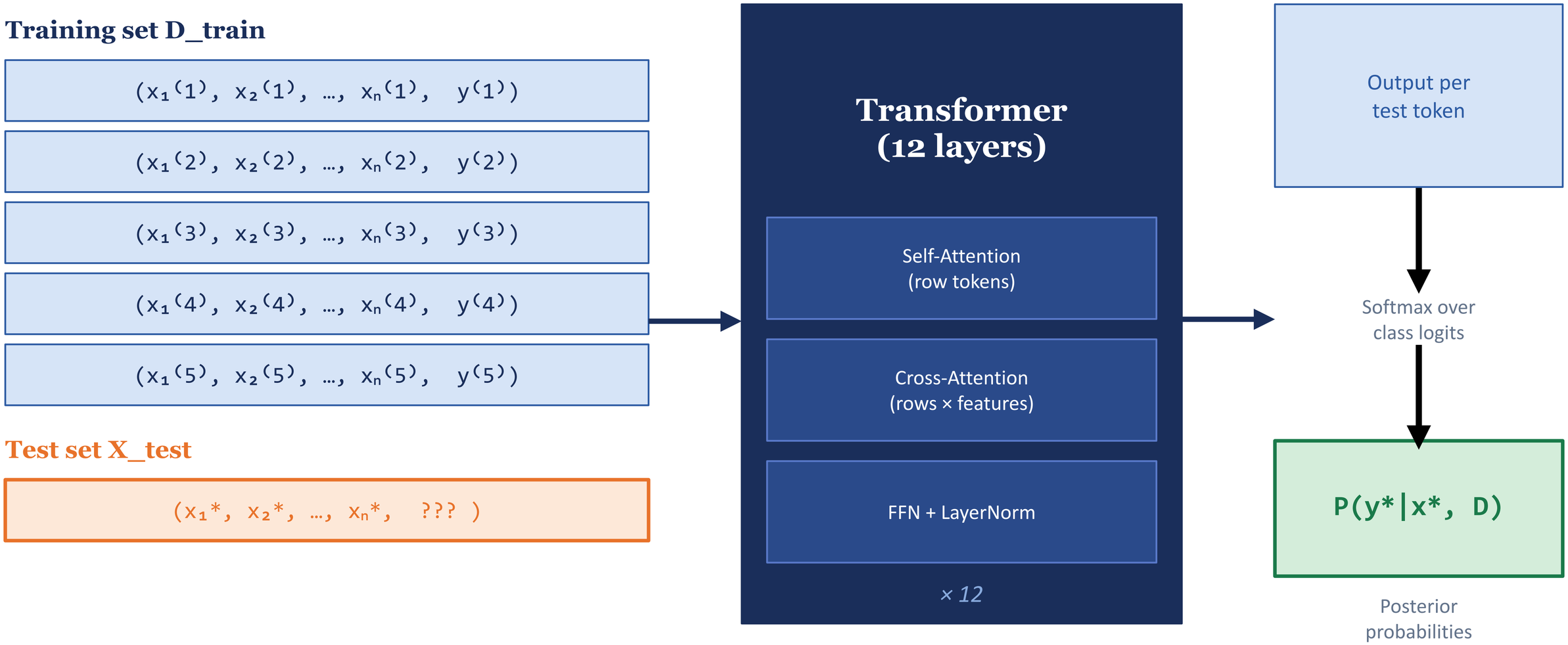
$z_{root} = \varepsilon, \quad \varepsilon \sim p_{noise}(\cdot)$

p_noise ∈ {N(0,1), Uniform[0,1], Beta(α,β), Laplace(0,b), …} sampled per node.

→ Varied marginal distributions give diverse feature marginals across synthetic datasets.

Architecture: In-Context Learning via Transformer

The network processes training + test samples jointly in one forward pass



Permutation invariant over training samples	All test predictions in one pass	No positional encoding (set not sequence)	Row & feature attention combined

Full Pipeline: From Causal Prior to Prediction

End-to-end view connecting the graphical model, simulation, training and inference

OFFLINE

1

Prior: p(SCM)

Random DAG + edge functions sampled

2

Simulate D

N_train + N_test samples generated

3

Train θ

Min CE loss over 18k batches × 512 D

ONLINE (inference)

1

Real D_train

User provides labelled training set

2

Append X_test

Test features concatenated as context

3

Forward pass

→ P(y*|x*,D) in < 1 second

Full Pipeline: From Causal Prior to Prediction

End-to-end view connecting the graphical model, simulation, training and inference

OFFLINE

1

Prior: p(SCM)

Random DAG + edge functions sampled

2

Simulate D

N_train + N_test samples generated

3

Train θ

Min CE loss over 18k batches × 512 D

ONLINE (inference)

1

Real D_train

2

Append X_test

3

Forward pass

The transformer weights θ are the amortized posterior — they implicitly encode p(ϕ|D) averaged over the entire SCM prior

Inductive Bias: Simplicity & Causality in the Prior

Two principled choices that help TabPFN generalise to real-world tabular data

Simplicity Bias (Occam's Razor)

- Prior places higher weight on simpler SCM structures: smaller DAGs, fewer hidden nodes, sparser edges.
- Motivated by Occam's razor and MDL principle: simpler explanations are preferred in the absence of evidence.
- Implemented by skewing hyperprior distributions toward smaller N (nodes) and sparser P (edges).
- Hypothesis: real-world tabular datasets arise from relatively simple underlying mechanisms.
- This prevents the posterior from collapsing to overly complex, overfit explanations.

Causal (SCM) Inductive Bias

- Using SCMs as data generators builds in an assumption that features and labels have a joint causal history.
- Pearl's Ladder of Causation: TabPFN operates at 'rung 1.5' — association-based but causally-structured.
- TabPFN does NOT perform causal discovery. It bypasses explicit graph recovery and predicts Y directly.
- Benefits: handles spurious correlations, distribution shifts from observational data more gracefully.
- Empirical check: TabPFN's predictions correlate with simple SCM-generated hypotheses (cf. Fig. 8 in paper).

Both biases are baked into the prior distribution — not into the transformer architecture itself, making them transparent and modifiable.

Results: TabPFN vs. GBDT and AutoML

OpenML-CC18 benchmark — 18 datasets, ≤1000 training samples, ≤100 numerical features

230×
speedup vs AutoML (CPU)

5700×
speedup vs AutoML (GPU)

< 1s
total inference time

#1
on 18 small datasets

Method	Relative Accuracy	Train + Predict Time	Notes
TabPFN	**Ref (SOTA)**	**< 1 s**	**No HPO needed**
XGBoost (tuned)	~-2% ROC	minutes	Hyperparameter search required
AutoML (AutoSklearn)	comparable	hours	Full pipeline search
Random Forest (def)	< TabPFN	seconds	No tuning but lower accuracy
KNN	< TabPFN	ms	Simple baseline

TabPFN also validates on 67 additional OpenML datasets — consistent advantage at small-data regime.