arXiv:2602.07098v2 [stat.CO] 23 Mar 2026

BayesFlow2：Python 中的多后端摊销贝叶斯推断

Lars Kühmichel 多特蒙德工业大学
Jerry M. Huang 伦斯勒理工学院
Valentin Pratz 海德堡大学
Jonas Arruda 波恩大学
Hans Olischläger 多特蒙德工业大学
Daniel Habermann 多特蒙德工业大学
Šimon Kucharský 多特蒙德工业大学
Lasse Elsemüller 独立科学家
Aayush Mishra 多特蒙德工业大学
Niels Bracher 伦斯勒理工学院
Svenja Jedhoff 多特蒙德工业大学
Marvin Schmitt 独立科学家
Paul-Christian Bürkner 多特蒙德工业大学
Stefan T. Radev 伦斯勒理工学院

摘要

现代贝叶斯推断涉及多种计算方法，用于在规范化工作流中对概率模型进行估计、验证并据此得出结论。许多贝叶斯方法的一个共同特点是速度相对较慢，因此在对大型数据集拟合复杂模型时，往往会成为难以克服的瓶颈。摊销贝叶斯推断（ABI）为解决贝叶斯方法中的计算难题提供了一条途径。ABI 在模型模拟数据上训练神经网络，使用户能够快速推断模型所隐含的任意量，例如点估计、似然或完整的后验分布。本文介绍用于通用摊销贝叶斯推断的 Python 库 BayesFlow 版本 2。除直接进行后验、似然和比率估计外，该软件还支持多种流行的深度学习后端，提供丰富的生成网络以用于采样和密度估计，并具备全面的定制能力和高级接口，以及超参数优化、设计优化和层次建模等新功能。通过一项关于动态系统参数估计的案例研究，并与类似软件进行比较，我们展示了这种精简、易用的工作流在推动广泛采用方面的巨大潜力。

*K*关键词：摊销贝叶斯推断、基于模拟的推断、深度学习 .

1. 引言

基于模拟的推断（SBI；Cranmer、Brehmer 和 Louppe 2020年；Deistler、Boelts、Steinbach、Moss、Moreau、Gloeckler、Rodrigues、Linhart、Lappalainen、Miller、Gonçalves、Lueckmann、Schröder 和 Macke 2025年）已成为计算科学中日益普遍的一部分（Lavin、Krakauer、Zenil、Gottschlich、Mattson、Brehmer、Anandkumar、Choudry、Rocki、Baydin 等 2021年）。SBI 旨在解决基于可模拟的概率模型 θ 和数据 $\mathcal{D}$ 估计未知参数 $p(\theta, \mathcal{D})$ 的问题。尽管 SBI 最初是为传统贝叶斯方法难以处理或无法实现的情境而提出的（Diggle 和 Gratton 1984年），但如今它正逐渐成为贝叶斯计算的一种通用方法（Zammit-Mangion、Sainsbury-Dale 和 Huser 2025年）。

在许多情况下，我们可能希望将模型拟合到多个数据集，无论是为了分析大数据（von Krause、Radev 和 Voss 2022），还是为了在计算机模拟中验证模型的关键属性（例如可识别性）（Bürkner、Schmitt 和 Radev 2025）。因此，除非我们拥有无限的计算时间和计算资源，否则理想的做法是汇集或“编译”计算，形成全局估计器，从而针对任意查询即时给出结果。作为一种高效概率推理和模型反演的通用框架（Gershman 和 Goodman 2014；Stuhlmüller、Taylor 和 Goodman 2013；Kingma 和 Welling 2013；Paige 和 Wood 2016；Le、Baydin 和 Wood 2017），摊销推断通过学习估计器，以尽量少的重复计算回答大量查询，从而实现这一目标。

作为 SBI 工作流的新兴标准，摊销贝叶斯推断（ABI）利用来自 $p(\theta, \mathcal{D})$ 的模拟数据训练神经网络。训练成本会因重复使用这些网络来快速推断模型蕴含的任意量（例如点估计、似然或完整的后验分布）而得到摊销。因此，ABI 为成熟的近似推断方法提供了替代方案，也可与之互补，例如马尔可夫链蒙特卡罗（MCMC；Brooks、Gelman、Jones 和 Meng 2011）、变分推断（Ranganath、Gerrish 和 Blei 2014）或拉普拉斯近似（Rue、Martino 和 Chopin 2009）。

版本 2 的 **BayesFlow** 进行了全面重新设计，在初始版本的基础上大幅扩展并改进（Radev、Schmitt、Schumacher、Elsemüller、Pratz、Schälte、Köthe 和 Bürkner 2023b）。它顺应了从孤立分析转向迭代式贝叶斯工作流的普遍趋势（Gelman、Vehtari、Simpson、Margossian、Carpenter、Yao、Kennedy、Gabry、Bürkner 和 Modrák 2020年；Li、Vehtari、Bürkner、Radev、Acerbi 和 Schmitt 2024年）。此外，它也契合了人们对人工智能辅助统计学日益增长的兴趣（Musslick、Bartlett、Chan- dramouli、Dubova、Gobet、Griffiths、Hullman、King、Kutz、Lucas等人2025年），在这一领域，神经代理模型可以替代经典工作流中的任何性能关键组件（Ye、Pandey、Bawden、Sumsuzzman、Rajput、Shoukat、Singer、Moghadas 和 Galvani 2025年）。

BayesFlow 为 **BayesFlow**提供精简的高级用户界面，以支持摊销贝叶斯推断（ABI）工作流，**Keras 3**生态系统（Chollet 等人，2025年）的一部分，**BayesFlow** 支持所有主流深度学习后端，并能顺畅融入 Python中的现代机器学习工作流。它具有很强的可移植性，也可通过**R**（R 核心团队，2025）访问，从而增强与 **R** 生态系统的互操作性。凭借稳健的默认设置，它降低了新用户的入门门槛。最重要的是，**BayesFlow**涵盖完整的贝叶斯工作流——从构建和拟合复杂模型（例如，无似然或基于似然、层级式或平坦式、动态或平稳，等等）到模型比较、模型批评和部署。

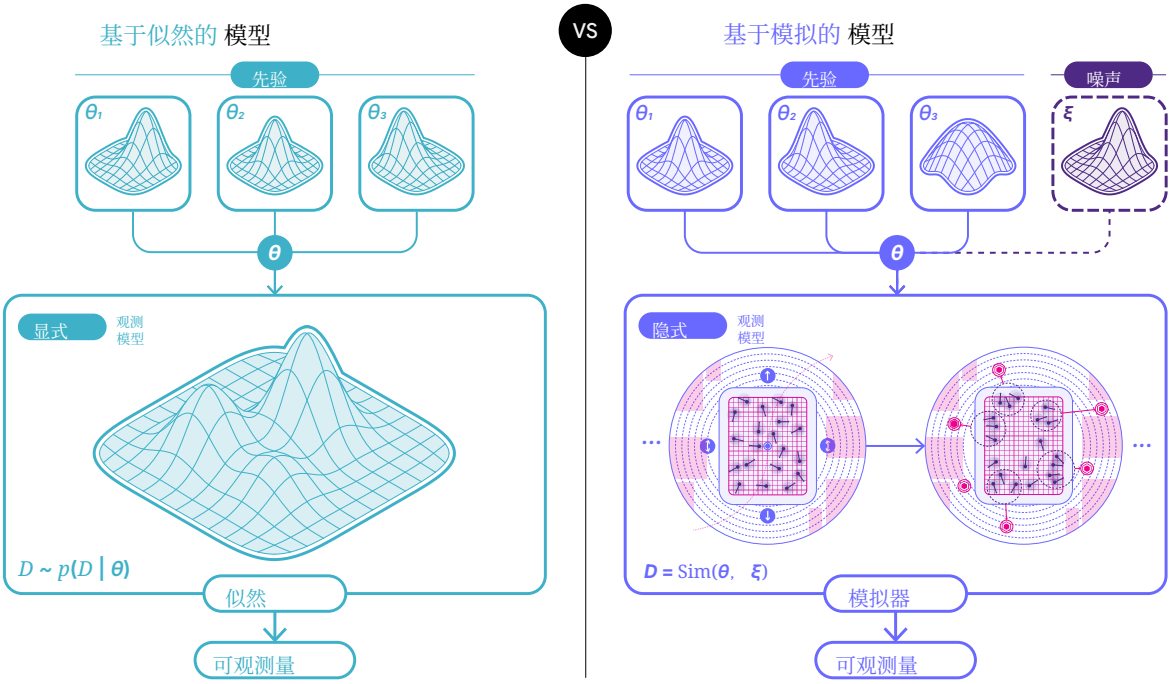


图1：从贝叶斯视角并列比较基于似然（显式）和基于模拟（隐式）的模型。在基于似然的模型中，可以显式地对数据模型$p(\mathcal{D} \mid \theta)$和先验 $p(\theta)$进行采样和评估。在基于模拟的模型中，可以通过随机模拟对数据模型$p_{\text{implicit}}(\mathcal{D} \mid \theta)$和先验 $p(\theta)$进行采样，但无法（轻易地）进行评估。

2. 摊销贝叶斯推断

2.1. 从模拟中学习

BayesFlow 要求建模者指定一个关于经验（可观测）量，$\mathcal{D}$和潜在（不可观测）量的概率模型 $p(\theta, \mathcal{D})\mathcal{D}$ θ。传统上，联合模型由似然函数 $p(\mathcal{D} \mid \theta)$ 和先验 $p(\theta)$ 完全定义：

$$p(\theta, \mathcal{D}) = p(\mathcal{D} \mid \theta)p(\theta). \tag{1}$$

与 Stan、PyMC 等现代基于 MCMC 的软件（Stan (Stan 开发团队 2025年) 或 **PyMC**(Abril-Pla、Andreani、Carroll、Dong、Kochurov、Kumar、Lao、Luhmann、Martin、Osthege、Vieira、Wiecki 和 Zinkov, 2023)）不同，**BayesFlow** 无需在推断过程中评估这些分布。相反，它只依赖于从联合模型 $p(\theta,\mathcal{D})$ 中进行模拟的能力，具体如下：

$$\mathcal{D} = \text{Sim}(\theta, \xi), \quad \theta \sim p(\theta),\ \xi \sim \text{RNG}(\cdot) \quad \Longleftrightarrow \quad \mathcal{D} \sim p_{\text{implicit}}(\mathcal{D} \mid \theta), \quad \theta \sim p(\theta), \tag{2}$$

其中，$\xi \in \Xi$ 表示随机（外包）噪声变量，$p_{\text{implicit}}(\mathcal{D} \mid \theta)$ 表示由左侧定义的似然。满足某些条件时，这个隐式似然可以写成对模拟器随机组件的积分：

$$p_{\text{implicit}}(\mathcal{D} \mid \theta) = \int_{\Xi} p(\mathcal{D}, \xi \mid \theta)\, \mathrm{d}\xi, \tag{3}$$

这通常难以计算，因此促使人们应用基于模拟的推断（SBI；Cranmer 等人2020年）。在现代 SBI 中，模拟数据用作专用神经网络的训练数据。训练完成后，这些网络便能以极低的额外开销推断任何感兴趣的模型量（例如参数），从而摊销网络训练的初始投入。这种方法既适用于具有闭式似然的传统模型，也适用于只能通过模拟生成数据的“无似然”模型（见图1）。

2.2. 贝叶斯参数估计

BayesFlow 采用贝叶斯框架进行参数估计，从而能够对完整后验分布进行推断：

$$p(\theta \mid \mathcal{D}) \propto p(\mathcal{D} \mid \theta) p(\theta). \tag{4}$$

我们将后验学习表述为在联合抽样 $(\theta, \mathcal{D}) \sim p(\theta, \mathcal{D})$ 上最小化期望目标函数：

$$\hat{q} = \operatorname*{argmin}_q \mathbb{E}_{(\theta,\mathcal{D}) \sim p(\theta,\mathcal{D})} \left[\mathcal{J}\left(q(\cdot \mid \mathcal{D}), \theta\right) \right] \tag{5}$$

$$\approx \operatorname*{argmin}_q \frac{1}{B} \sum_{b=1}^{B} \mathcal{J}(q(\cdot \mid \mathcal{D}^{(b)}), \theta^{(b)}), \tag{6}$$

其中 B 是模拟预算，$\mathcal{J}$ 是一种严格适当评分规则（例如，对数评分），并且q 是由生成式神经网络定义的近似分布。对目标函数 $\mathcal{J}$ 的不同选择可得到众所周知的估计器。使用对数评分可得到最大似然目标函数（Papamakarios、Nalisnick、Rezende、Mohamed 和 Lakshminarayanan，2021），而使用基于评分的目标函数则对应于去噪评分匹配（Arruda、Bracher、Köthe、Hasenauer 和 Radev，2025）。尽管后者并不依赖于适当评分规则，但它可以被解释为一种在特定条件下近似最大似然训练的替代目标函数（Song、Durkan、Murray 和 Ermon，2021）。

BayesFlow还可以学习任意贝叶斯点估计器（Sainsbury-Dale、Zammit- Mangion 和 Huser 2024年），例如后验均值、方差或分位数。所得的优化目标只需将公式 q 5 中的 近似分布替换为所需的点估计$\hat{} f(\mathcal{D})$：

$$\hat{f} = \operatorname*{argmin}_f \mathbb{E}_{(\theta,\mathcal{D}) \sim p(\theta,\mathcal{D})} \left[\mathcal{J}\left(f(\mathcal{D}), \theta\right) \right], \tag{7}$$

其中，$\mathcal{J}$ 现在是针对点估计器的评分规则。无论哪种情况，推断都是摊销的：训练完成后，$\hat{} q$ 可以高效地从近似后验中采样，或者 f 可以为任何即将到来的、与联合模型 $p(\theta, \mathcal{D})$ 兼容的数据 $\mathcal{D}_{\text{new}}$ 生成点估计。

2.3. 贝叶斯模型验证

模型验证是指在将模型应用于真实数据之前，在计算机模拟中评估模型的假设，因为这些假设在真实数据中可能并不成立（Bürkner 等 2025年）。通常，这涉及根据模拟数据估计后验分布，并验证校准（即可信区间是否可靠）和恢复（即能否准确估计参数）等关键属性。传统上，模型验证与模型重新拟合的高计算成本相关；而在**BayesFlow** 中，由于摊销，模型验证几乎无需额外成本即可完成。下面我们将讨论模型验证的两个关键方面。

校准 考虑一个目标关注量 $T = T(\theta)$，它是参数 θ 的函数。这可以包括参数本身，也可以是任何推前量，例如后验预测。如果模型设定正确（即与真实数据生成分布相同），则以下等式成立：

$$\alpha = \mathbb{E}_{(\tilde{\theta},\mathcal{D})\sim p(\theta,\mathcal{D})}\left[\mathbb{I}(T(\tilde{\theta}) \in U_\alpha(T(\theta) \mid \mathcal{D}))\right], \tag{8}$$

对于所有目标量 T 以及所有不确定性区域 U_α（T（θ）$\mid \mathcal{D}$）从解析的˜后验分布 p（$\theta \mid \mathcal{D}$）在名义覆盖概率 α下，其中 θ 表示模拟真实值（Bürkner、Scholz 和 Radev，2023）。换言之，覆盖概率为 α 的不确定性区域（例如可信区间）包含真实值的概率，平均而言必须等于 α。这一性质可用于验证近似后验的校准：如果近似足够好，那么对于该近似所蕴含的不确定性区域，式（8）应当成立。不过，验证校准需要将模型拟合到许多（通常是数百个）模拟数据集（Modrák、Moon、Kim、Bürkner、Huurre、Faltejsková、Gelman 和 Vehtari，2023）。对于基于马尔可夫链蒙特卡罗或其他非摊销算法的方法，这在计算上可能颇具挑战。在**BayesFlow**中，摊销带来了显著优势：训练完成后，后验估计器便能在整个样本空间内高效地自我诊断，使基于模拟的校准（SBC）成为常规操作。

参数可恢复性 模型验证的另一个关键方面，是模型能否准确恢复自身参数。形式上，可将参数恢复性表示为

$$\mathrm{Recovery}(\mathcal{L}) := \mathbb{E}_{(\tilde{\theta},\mathcal{D})\sim p(\theta,\mathcal{D})}\left[\int_\Theta \mathcal{L}(\tilde{\theta},\theta)\, p(\theta \mid \mathcal{D})\, \mathrm{d}\theta\right], \tag{9}$$

其中 $\mathcal{L}$表示一种损失度量（例如均方根误差；RMSE）。与校准类似，传统的恢复分析需要在许多模拟数据集上反复进行后验推断，因此计算开销很大（Schad、Betancourt 和 Vasishth 2021年）。相比之下，**BayesFlow**可在训练后即时完成恢复分析。

2.4. 似然估计

在某些应用中，学习一个替代模型 $q(\mathcal{D} \mid \theta)$ 来近似不可处理似然$p_{\text{implicit}}(\mathcal{D} \mid \theta)$可能更为理想。学习似然可将模型训练与先验设定解耦，从而支持灵活复用，例如用于层级模型或回归模型，其中参数 θ 是其他变量的函数（Papamakarios、Sterratt 和 Murray 2019年；Lueckmann、Bassetto、Karaletsos 和 Macke 2019年；Fengler、Govindarajan、Chen 和 Frank 2021年；Boelts、Lueckmann、Gao 和 Macke 2022）。此外，神经似然还可作为快速、可微的仿真器，替代原本计算成本高昂的模拟器。
从 **BayesFlow** 的角度来看，似然估计只需交换公式 5 中的参数位置，此时近似密度 $q(\cdot \mid \theta)$ 以参数为条件。若 $\mathcal{D}$ 的维度相较于 θ较高，学习似然可能需要更大的模拟预算。此外，如果最终目标是估计后验分布，那么所得流程 并非 摊销式的，因为对每个$\mathcal{D}$ (Radev、Schmitt、Pratz、Picchini、Köthe 和 Bürkner，2023a)。

与似然估计相关的是似然-证据比估计，即近似密度比 $p(\mathcal{D} \mid \theta)\ /\ p(\mathcal{D})$ (Hermans、Begy 和 Louppe 2020；Durkan、Murray 和

Papamakarios 2020b；Miller、Weniger 和 Forré 2022b）。由于该比率与似然成正比，因此似然-证据比估计可代替似然密度使用，例如用于后续通过马尔可夫链蒙特卡罗进行后验估计。**BayesFlow** 实现了 Miller 等人 (2022b提出的稳定对比训练方法，该方法使用一个简单的多类别分类器。

2.5. 贝叶斯模型比较

前述概率量隐含地取决于各种模型假设。这些假设可抽象地表示为 M，而比较不同假设的实用价值则需要进行贝叶斯模型比较（BMC；MacKay 2003）。形式上，先验预测 BMC 取决于模型的边际似然（即式 4）的归一化常数：

$$p(\mathcal{D} \mid M) = \int_{\Theta} p_{\text{implicit}}(\mathcal{D} \mid \theta, M)\, p(\theta \mid M)\, \mathrm{d}\theta, \tag{10}$$

对于隐式模型，这一计算具有双重难解性。由于边际似然是归一化的概率分布，它们通过在尖锐度与分散度之间权衡来惩罚模型复杂度：复杂模型能够生成范围更广的数据，但必须将其概率质量分布在更大的样本空间范围内。

两个模型的边际似然之比称为贝叶斯因子（BF；Kass 和 Raftery，1995年）：

$$\mathrm{BF}_{12} := \frac{p(\mathcal{D} \mid M_1)}{p(\mathcal{D} \mid M_2)} = \frac{p(M_1 \mid \mathcal{D})}{p(M_2 \mid \mathcal{D})} \Big/ \frac{p(M_1)}{p(M_2)}. \tag{11}$$

右侧第一项表示后验赔率，第二项则表示先验赔率。对于一组 J 模型，假设真实模型属于所考虑的模型集合，我们也可以等价地计算相应的 J后验模型概率 $p(M_j \mid \mathcal{D})$。
BayesFlow 将贝叶斯模型比较视为概率分类任务，以此估计后验模型概率：

$$\hat{q} = \operatorname*{argmin}_{q} \mathbb{E}_{(M,\theta,\mathcal{D}) \sim p(M)p(\theta,\mathcal{D} \mid M)} \left[\mathcal{J}\left(q(\cdot \mid \mathcal{D}), M\right) \right], \tag{12}$$

其中 $\mathcal{J}$ 表示一个适当的评分规则（例如，损失或对数评分），并且 M 是真实模型索引。本质上，我们在模拟配对上定义一个监督学习问题 $(M, \mathcal{D})$ 并训练一个概率分类器来预测模型标签（Radev、D'Alessandro、Mertens、Voss、Köthe 和 Bürkner 2021年；Pudlo、Marin、Estoup、Cornuet、Gautier 和 Robert 2016年；Jeffrey 和 Wandelt 2024年）。

或者，BayesFlow 可以使用一对经过训练的后验分布和似然估计器，并通过重新整理贝叶斯定理来近似对数边际似然（LML）：

$$\log \hat{q}(\mathcal{D} \mid M) = \log \hat{q}(\mathcal{D} \mid \theta, M) + \log p(\theta \mid M) - \log \hat{q}(\theta \mid \mathcal{D}, M). \tag{13}$$

由于左侧的量相对于 θ是常数，因此从理论上讲，我们可以在右侧代入每一个 $\theta \in \Theta$，从而得到 LML 的估计。然而，由于后验分布和似然估计并不完美，不同的 θ 会导致$\log \hat{}q(\mathcal{D} \mid M)$取值不同。除了给出 LML 的点估计外，这也提供了近似误差的度量。只要解析似然可用，就可以通过使用解析似然来降低这一误差（Radev 等2023a；Kucharský、Mishra、Habermann、Radev 和 Bürkner，2025年）。

最后，我们还可以根据后验预测量对模型进行排序，这些量根据˜新数据$\mathcal{D}$计算得出，例如期望对数预测密度（ELPD；Vehtari 和 Ojanen 2012）：

$$\mathrm{ELPD}(M) = \mathbb{E}_{\tilde{\mathcal{D}} \sim p(\mathcal{D})} \left[\log \int_{\Theta} p_{\text{implicit}}(\tilde{\mathcal{D}} \mid \theta, M)\, p(\theta \mid \mathcal{D}) \mathrm{d}\theta \right]. \tag{14}$$

ELPD具有三重不可解性，但外层期望可以通过交叉验证估计，而内层期望则可以通过对近似后验样本上的似然替代模型取平均来估计（Radev等人2023a）。

2.6. 贝叶斯敏感性分析

任何（贝叶斯）分析的结果都会受到与似然设定、先验 elicitation 和数据处理相关的建模选择的影响。例如，贝叶斯分析中一个常见的问题是：如果采用不同的先验，结果会如何？敏感性分析为这类问题提供了正式的解答，通常需要在不同假设下重新拟合模型。通过将摊销范围扩展至上下文变量 C，即可解决由此产生的计算瓶颈；这些变量用于指示推断变化的潜在因素。例如， C 可以编码先验的宽度，也可以表示完全不同的模型设定。

为实现摊销的、敏感性感知的分析，我们可以将联合模型扩展为 $p(C,\ \theta,\ \mathcal{D})$，并将 C 作为神经估计器的额外输入。敏感性感知估计只需扩展式（5）中的目标函数，将 C纳入其中：

$$\hat{q} = \underset{q}{\operatorname{argmin}}\, \mathbb{E}_{(C,\theta,\mathcal{D}) \sim p(C,\theta,\mathcal{D})} \left[\mathcal{J}\left(q(\cdot \mid \mathcal{D}, C), \theta\right) \right]. \tag{15}$$

在推断过程中，建模者可以提供 C 的不同取值，并定性或定量地评估由此产生的差异（Elsemüller, Olischläger, Schmitt, Bürkner, Koethe 和 Radev 2024a）。此外，为了估计结果对网络和训练超参数的敏感性，可以训练一个网络集成，随后分析各集成成员之间的差异。

2.7. 处理模型设定错误

在无数情境中，研究人员会使用误设定模型对复杂现象进行建模（Walker 2013年）。有许多标准可用于判断一个误设定（即错误的）模型是否仍然有用（Bürkner 等人2023）。在 SBI 的语境下，模型设定错误意味着模拟数据缺少真实数据中存在的相关结构。模拟差距不仅表明模型设定错误；当差距严重时，训练后的网络可能不再估计其在训练期间原本要估计的同一量（Schmitt, Bürkner, Köthe 和 Radev 2023；Gloeckler, Deistler 和 Macke 2023；Frazier, Kelly, Drovandi 和 Warne 2024年）。

要理解这一问题的独特之处，一个有用的方法是将摊销推断与马尔可夫链蒙特卡罗进行对比。如果马尔可夫链蒙特卡罗采样器能够充分混合并收敛，那么按照构造，它会返回假设模型依据贝叶斯定理所蕴含的后验分布，即使该后验分布并不能很好地反映现实。摊销神经估计器则有所不同：在存在模拟差异的情况下，它们可能偏离名义模型的贝叶斯后验，转而输出反映训练分布和网络归纳偏置的结果。在这种情况下，后验估计

$q(\theta \mid \mathcal{D})$ 可能不再对应模型的任何一致贝叶斯更新，这会削弱我们对所报告的不确定性和点估计的解读。
因此，检测并缓解模型设定错误如今被广泛视为摊销贝叶斯工作流不可或缺的一部分（Li 等 2024 年）。**BayesFlow** 提供了相关工具，用于通过检查真实数据相对于所学习的数据嵌入分布或汇总统计量的典型性，来检测 $\phi(\mathcal{D})$ 潜在的模拟差距。任何分布外（OOD）或离群值检测得分，都可以作为潜在“外推偏差”的代理度量（Frazier 等 2024年）。嵌入分布 $p(\phi(\mathcal{D}))$ 还可以通过额外的训练损失进行“高斯化”（Schmitt 等2023），以满足简单参数方法（如马氏距离）的假设（Li 等 2024年）。

缓解模型设定错误对摊销估计器的影响仍是一个活跃的研究领域。有时，最简单的解决方案是假设一个污染模型 $p_\epsilon(\theta, \mathrm{D})$，该模型可以模拟离群值并诱导出稳健神经估计器（Ward, Cannon, Beaumont, Fasiolo, and Schmon 2022），以牺牲一定准确度为代价，换取高得多的崩溃点（Wu, Radev, and Tuerlinckx 2024）。其他损失函数，例如贝叶斯自洽（Mishra, Habermann, Schmitt, Radev, and Bürkner 2025）或无监督域适应（Huang, Bharti, Souza, and Acerbi 2023；Elsemüller, Pratz, von Krause, Voss, Bürkner, and Radev 2025），也能提升鲁棒性。或者，对于被标记为分布外（OOD）的数据集，如果有似然可用，可以尝试事后通过帕累托平滑重要性采样（PSIS）校正后验分布（Vehtari, Simpson, Gelman, Yao, and Gabry 2024），也可以更一般地采用推断时适应（Siahkoohi, Rizzuti, Orozco, and Herrmann 2023）。至关重要的是，BayesFlow 提供了一个接口，可用于实现和测试各种提升摊销贝叶斯推断（ABI）鲁棒性与可信度的方法（Schmitt et al. 2023；Huang et al. 2023；Mishra et al. 2025；Elsemüller et al. 2025, 2024a；Li et al. 2024）。

3. 软件

BayesFlow 软件分为多个模块，这些模块协同工作，可实现完整的端到端 ABI 工作流。下面将介绍 **BayesFlow** 的所有主要模块（另见 图 2）。

3.1. 模拟器

由于 ABI 执行基于模拟的推断，模拟器在 BayesFlow 中发挥着关键作用。该软件包提供低层级和高层级的模拟器接口，用户可据此灵活权衡表达能力、计算效率和易用性。

在较低层级上，BayesFlow 中的模拟器被定义为一个包含 `sample(batch_size)` 方法的类，该方法返回由 NumPy 数组构成的数据字典。字典的键是任意的、由用户选择的字符串，用于表示变量名。例如，考虑以下简单的概率模型：

$$
\begin{aligned}
\mu &\sim \mathrm{Normal}(0, 1) \\
\sigma &\sim \mathrm{Exponential}(1) \\
x_n &= \mathrm{Normal}(\mu, \sigma) \quad n = 1, \ldots, N
\end{aligned}
$$

这可以编码为如下形式的 BayesFlow 模拟器：

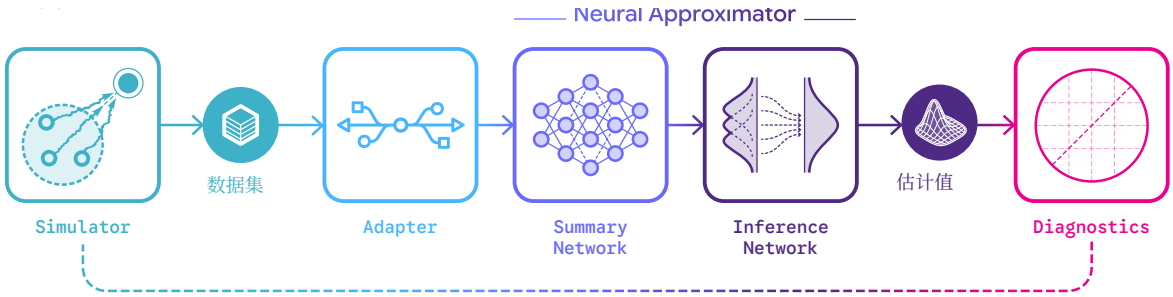


图 2：BayesFlow 提供的基本摊销贝叶斯工作流概览。`Simulator` 接口可自动将数据生成函数与模拟数据进行分组。 `Adapter`接口会转换原始模拟器输出，以便训练神经近似器。神经近似器通常由一个 `SummaryNetwork`组成，用于将可观测量编码为潜在摘要向量；此外还包括一个 `InferenceNetwork` ，以该向量为条件来近似目标分布。随后，所得估计值将通过 BayesFlow 的 `diagnostics` 模块用于模型验证，并借助丰富的图形和数值诊断来检查计算忠实度与模型敏感性。整个工作流通过 `Workflow` 对象在高层次上进行封装，使用户能够快速迭代贝叶斯工作流。

```
import numpy as np
import bayesflow as bf

class MySimulator(bf.simulators.Simulator):
    def sample(self, batch_size, N=10):
        mu = np.random.normal(size=(batch_size, 1))
        sigma = np.random.exponential(size=(batch_size, 1))
        x = np.random.normal(size=(batch_size, N), loc=mu, scale=sigma)
        return {"mu": mu, "sigma": sigma, "x": x}
```

然后，我们可以通过模拟器模拟一批参数-数据对：

```
simulator = MySimulator()
sims = simulator.sample(batch_size=5)
```

为提高计算效率，BayesFlow 假设对 `sample(batch_size)`的一次调用将返回一个批处理字典，即所有值数组的形状均为 `(batch_size,...)`。对于上述示例， `sims["theta"]`的形状为 `(batch_size, 2)` ，而 `sims["x"]`的形状为 `(batch_size, N)`。对于没有深度学习背景的用户而言， `batch_size` 的需求可能并不直观，但它是小批量优化所必需的，而小批量优化对于大多数深度学习应用至关重要。

为方便起见，BayesFlow 还为用户提供了一种实用工具，可将未批处理的函数自动转换为批次模拟器：`bf.make_simulator`。使用这一更高级别的模拟器接口，现在可以将上述相同的概率模型编写为

```
def prior():
    mu = np.random.normal()
```

```
    sigma = np.random.exponential()
    return {"mu": mu, "sigma": sigma}

def likelihood(mu, sigma, N=10):
    x = np.random.normal(size=N, loc=mu, scale=sigma)
    return {"x": x}

simulator = bf.make_simulator([prior, likelihood])
```

在上述模拟器中，整个批次的样本量 `N`被视为固定值。这一点很重要，因为 `N`会影响 `x` 的形状。只有当每次模拟器运行的形状相同时，模拟才能安全地拼接成批次。若要在模拟器中改变 `N`，我们需要按批次改变它，而不是在批次内按模拟改变。为此，`bf.make_simulator` 提供了可选参数 `meta_fn`，该参数接受一个函数，用于返回一组变量字典，这些变量只按批次变化（在批次内保持不变）。如果提供了该参数，BayesFlow 会为每个批次调用一次 `meta_fn`，并将其输出添加到常规模拟器调用的前面。若要扩展上述模拟器，使其在 10 到 100 之间均匀抽取 N，可以这样编写：

```
def meta():
    return {"N": np.random.randint(10, 20)}

simulator = bf.make_simulator([prior, likelihood], meta_fn=meta)
```

`simulator.sample()` 的输出现在包含一个新条目 `"N"`，其值只是一个整数：该批次中所有模拟所使用的样本量。为了提高速度，也可以使用 `simulator.sample_parallel()`方法并行运行模拟。

对于概率分解比单组参数推导数据更复杂的模型，可以使用 `HierarchicalSimulator` 类等工具，以支持多层级模型和混合模型的模拟（Habermann、Schmitt、Kühmichel、Bulling、Radev 和 Bürkner 2024年；Kucharský 和 Bürkner 2025年）。

3.2. 适配器

原始模拟器输出无法直接用于训练神经网络。相反，必须先执行多个预处理步骤。具体包括：(a) 将变量调整为正确的形状（例如，广播、拆分或串联变量）；(b) 约束变量（例如，确保变量非负）；以及 (c) 告知网络哪些变量用于何种用途（例如，哪些变量是参数，哪些是数据）。BayesFlow 为所有这些预处理任务提供了统一接口： `Adapter`，它以字典作为输入（通常由 `Simulator`生成），随后返回一个经过转换的变量字典。

为了更具体地说明，对于上面的简单模型，我们需要：(1) 将 `"N"` 广播到与 `"x"`相同的形状，因为最终我们希望将二者结合起来作为神经网络的条件；(2) 指明 `"x"`是一组可交换值，其顺序对推断没有影响；(3) 约束 `"sigma"` 始终预测为正值；(4) 对 `"N"`进行平方根变换，以提高数值稳定性。

此外，我们还需要指明摊销推断任务中各个变量所扮演的角色。在我们的示例中，"mu" 和 "sigma"是感兴趣参数，我们最终希望根据数据推断它们的后验分布。因此，这两个变量将串联为 "inference_variables"，这是后续模块能够识别的一个受保护变量名。数据 "x" 应首先由摘要网络进行汇总，我们通过在适配器中将 "x"重命名为 "summary_variables" 来指明这一点。最后，我们还希望在推断时以样本量 "N"作为条件，但由于无需对其进行汇总，因此打算将它直接传递给推断网络；为此，将 "N" 重命名为 "inference_conditions"。通过以下代码，可以将所有这些变换整合到一个适配器中：

```
adapter = (
    bf.Adapter()
    .broadcast("N", to="x")
    .as_set("x")
    .constrain("sigma", lower=0)
    .sqrt("N")
    .concatenate(["mu", "sigma"], into="inference_variables")
    .rename("x", "summary_variables")
    .rename("N", "inference_conditions")
)
```

在上述每一行中，都会向适配器添加一个额外的变换，以便在传入数据后按顺序执行。所示语法避免了在每一行都重复写入 adapter.<trans>。更详细地写，上述代码如下：

```
adapter = bf.Adapter()
adapter = adapter.broadcast("N", to="x")
adapter = adapter.as_set("x")
...
```

定义完成后，我们可以直接将 simulator.sample 的输出传递给 adapter：

```
sims = simulator.sample(batch_size=5)
adapted_sims = adapter(sims)
```

话虽如此，用户很少需要像这样直接调用适配器。相反，其他模块会在训练和推断阶段自动调用适配器。
适配器的另一个重要特性是，大多数变换都是可逆的。也就是说，我们可以调用

```
adapter(adapted_sims, inverse=True)
```

以恢复我们原来的模拟。有些变换根据定义不可逆，因此在求逆过程中会被忽略。例如 adapter.drop("<variable name>")会从字典中丢弃给定变量，这种变换无法通过求逆撤销。

在实践中，可逆性仅对涉及推断变量的变换是必需的（此处为参数 "mu"和 "sigma"）。这样，在推断时，神经网络预测可以按逆序遍历适配器，从而得到具有原始名称和尺度的推断出的变量。

生成族	架构	采样	密度评估
归一化流	受约束	单步	Fast
自由形式流	半约束	单步	中等
扩散模型	无约束	多步	Slow
流匹配	无约束	多步	Slow
一致性模型	无约束	少步	N/A

表1：BayesFlow 中可用的生成式神经网络系列。

3.3. 网络

在大多数应用中，**BayesFlow** 采用两个神经网络，它们共同实现对目标分布的学习：一个 SummaryNetwork 将（可能长度不定的）观测编码为固定长度的潜在摘要向量，另一个 InferenceNetwork以该潜在摘要为条件，生成目标分布的近似（另请参见图 2）。

对于给定应用， SummaryNetwork 的具体选择应取决于需要被摘要的数据。例如，当观测集合是可交换的（正如 x在我们的简单模型中那样）时， SummaryNetwork 的常见选择包括基于池化的编码器（例如， DeepSet；Zaheer、Kottur、Ravanbakhsh、Poczos、Salakhutdinov 和 Smola 2018），以及基于注意力的集合编码器，如 SetTransformer（Lee、Lee、Kim、Kosiorek、Choi 和 Teh 2019年）。这些选项以计算成本换取表达能力：基于池化的编码器通常成本更低，而基于注意力的编码器可能以更高的内存和计算成本恢复更复杂的样本间交互。再举一例，当观测构成时间序列时，时间神经网络、TimeSeriesNetwork（LSTN；Zhang 和 Mikelsons 2023）或 TimeSeriesTransformer（Wen、Zhou、Zhang、Chen、Ma、Yan 和 Sun 2022）是合适的选择。 wrappers 模块允许纳入以自定义后端编写的任意摘要网络（例如，Mamba，Gu 和 Dao 2024年）。

该 InferenceNetwork实现了一个条件密度估计器，至少支持从目标分布中采样。根据架构的选择，也支持密度评估（参见表1）。通常，推断网络（例如 FlowMatching(Lipman、Chen、Ben-Hamu、Nickel 和 Le，2023；Tong、Fatras、Malkin、Huguet、Zhang、Rector-Brooks、Wolf 和 Bengio，2024年）基于某种子网络架构，例如多层感知机（MLP）或卷积神经网络（CNN），由其执行实际的前向传播，而外层的 InferenceNetwork 则实现特定的损失函数和采样过程，并可选地支持密度评估。

对于 InferenceNetwork的选择很大程度上取决于应用场景。通常，表达能力更强的架构，例如多步 DiffusionModel(Ho、Jain 和 Abbeel 2020；Rombach、Blattmann、Lorenz、Esser 和 Ommer 2022；Song、Sohl-Dickstein、Kingma、Kumar、Ermon 和 Poole 2020；Kingma 和 Gao 2023；Arruda 等 2025年）、少步ConsistencyModel(Song、Dhariwal、Chen 和 Sutskever 2023；Song 和 Dhariwal 2023；Schmitt、Pratz、Köthe、Bürkner 和 Radev 2024b）或（最优传输） FlowMatching

以准确度换取采样速度。单步模型，例如 `CouplingFlows`(Dinh、Krueger 和 Bengio 2015年；Dinh、Sohl-Dickstein 和 Bengio 2017年），可以加快推断速度，但可能难以应对维度和多模态性。一般而言，更大的子网络可以提升任何 `InferenceNetwork`的表达能力，但仍受相应方法的限制。最后，自由形式模型（例如扩散、流匹配）可以实现任意子网络，并将摘要网络纳入其架构。

为了提供一个具体的代码示例，我们可以如下定义一个深度为 4 的摘要网络和推断网络：

```
summary_network = bf.networks.SetTransformer(summary_dim=8)
inference_network = bf.networks.CouplingFlow(depth=4)
```

`summary_dim` 的参数尤为重要，因为它定义了要提取的学习摘要的数量。根据经验，我们建议`summary_dim` 至少是推断变量总维度的两倍，甚至四倍。在我们的示例中，μ 和 σ 是推断变量——二者都是标量，因此它们的总维度为 2。在这里，选择 `summary_dim=8` 应该是稳妥的。

3.4. 近似器

`Approximator` 类将推断目标（由 `Simulator` 和 `Adapter`给出）与推断方法（由 `Networks` 给出）结合起来。在 `Approximator`中，可以使用模拟数据训练神经网络，随后对任何新数据执行推断。在大多数应用中，我们的推断变量都是连续的，因此会使用 `ContinuousApproximator` 子类。此外，还有其他近似器类型，例如用于学习任意贝叶斯估计量的 `ScoringRuleApproximator`（例如用于后验点估计）、用于学习似然-证据比的 `RatioApproximator` ，以及用于学习具有复杂概率因子分解的模型（如多层模型和混合模型）后验分布的`GraphicalApproximator` 。

近似器的训练

我们通过向近似器提供先前定义的 `Networks` 和`Adapter`来初始化它：

```
approximator = bf.approximators.ContinuousApproximator(
inference_network=inference_network,
summary_network=summary_network,adapter=adapter,)
```

初始化后，需要使用合适的优化器编译 `approximator` 。由于 BayesFlow 基于 Keras 3 构建，因此我们使用 `keras`优化器来完成此操作。例如：

```
import keras

optimizer = keras.optimizers.AdamW(learning_rate=1e-4)
approximator.compile(optimizer=optimizer)
```

编译完成后，我们就可以通过 `approximator.fit` 方法训练神经网络了。该方法需要提供训练流程的详细信息，包括训练轮数（参数 `epochs`）、每个训练周期的批次数（`num_batches`）以及批次大小（`batch_size`；每批模拟次数）。这些都是深度学习流程中的标准参数。

一般来说，训练流程分为两种：在线训练和离线训练。将 `Simulator` 传递给 `approximator.fit` 的 `simulator`参数表示采用在线训练，此时会通过反复调用模拟器，即时生成模拟训练数据：

```
approximator.fit(simulator=simulator, epochs=50, num_batches=100,
 batch_size=64)
```

这里的 `epochs`、 `num_batches` 和 `batch_size`参数仅为示例，并不一定是合适的默认值。

直接向 `approximator.fit`提供模拟数据，表示采用离线训练。为此，我们首先定义一个 `OfflineDataset` 对象，用于存放模拟训练数据，并负责在每个训练周期中将数据拆分成多个批次。然后，我们将数据传递给 `approximator.fit`的 `dataset`参数：

```
sims = simulator.sample(1024)
data = bf.OfflineDataset(sims, batch_size=64, adapter=adapter)
approximator.fit(dataset=data, epochs=50)
```

对于训练数据已预先模拟好的情况，离线训练尤为适用。它会直接将模拟数据存储为变量字典，而不是模拟器对象。当模拟器运行速度较慢，或难以在 Python 中执行时，通常更适合采用离线训练。此外，即使模拟器速度较快，离线训练通常也更快，因为可以预先将训练数据加载到图形处理器中，从而更高效地进行神经网络训练。不过，由于预先模拟的训练数据是固定且有限的，在神经网络训练期间，每条模拟数据都会被多次使用（更准确地说：每个训练周期使用一次）。因此，离线训练策略容易导致过拟合。为了在训练过程中及早发现潜在的过拟合，我们可以将其他模拟数据传递给 `approximator.fit`的 `validation_data` 参数：

```
val_sims = simulator.sample(128)
val_data = bf.OfflineDataset(val_sims, batch_size=64, adapter=adapter)
approximator.fit(dataset=data, validation_data=val_data, epochs=50)
```

按照深度学习中的惯例，会评估验证数据上的损失并将其显示在历史记录中，但不会用它来训练神经网络。与离线训练不同，在线训练不会受到过拟合的影响，因为训练过程中每次模拟只会被看到一次（因此不需要验证数据）。

使用近似器进行推断

训练完成后，我们可以让近似器从目标分布中生成样本，或评估目标的对数密度。对于我们的示例，目标是给定数据 x时参数 μ 和 σ 的后验分布 $p(\mu, \sigma \mid x)$。由于训练神经网络后，推断过程已实现摊销，因此我们可以近乎即时地获得多个新数据集（模拟数据集或真实数据集）的后验样本：

```
test_sims = simulator.sample(10)
samples = approximator.sample(conditions=test_sims, num_samples=100)
```

`samples` 对象是一个字典，其键名对应于推断变量（此处为`"mu"` 和 `"sigma"`）。每个元素都是形状为 `(n_datasets, n_samples,len_parameter)`的 NumPy 数组。因此，在我们的示例中，`"mu"` 和 `"sigma"`的样本形状均为 `(10, 100, 1)`，因为我们针对 10 个模拟数据集中的每个数据集评估了 100 个样本，而我们的参数都是标量（`len_parameter = 1`）。

除了采样之外，我们还可以在给定的推断变量值和条件下评估目标分布的对数密度。例如：

```
log_densities = approximator.log_prob(data=test_sims)
```

`data` 的 `approximator.log_prob`参数必须同时包含目标分布的条件（此处为 x）以及应评估密度处的推断变量值（μ和 σ）。这与 `approximator.sample` 形成对比，后者的`conditions`参数只需要目标分布的条件，而不需要推断变量本身的值，因为这些值将通过采样获得。

3.5. 诊断

我们训练出一个近似器，并不意味着它对目标分布的近似一定足够好。为了验证训练是否成功，BayesFlow 在`Diagnostics`模块中提供了丰富的诊断工具。在该模块中，我们区分图形诊断（子模块 `diagnostics.plots`）和数值诊断（子模块 `diagnostics.metrics`）。训练后通常首先查看的图形诊断是损失历史：

```
bf.diagnostics.plots.loss(approximator.history)
```

理想情况下，训练损失（以及存在时的验证损失）应随训练轮数增加而下降，并最终收敛到较低水平（示例参见 图4 中的 第 4 节 ）。
大多数其他图形诊断会比较从（近似）目标分布中获得的样本（参数 `estimates`）与模拟真实值（`targets`）。例如，要进行图形化的基于模拟的校准检验（SBC；Modrák 等 2023；Talts、Betancourt、Simpson、Vehtari 和 Gelman， 2020），我们可以运行：

```
test_sims = simulator.sample(10)
test_samples = approximator.sample(conditions=test_sims, num_samples=100)
```

```
bf.diagnostics.plots.calibration_ecdf(
estimates=test_samples,
targets=test_sims,)
```

大多数数值诊断遵循相同的模式。例如，要计算与上述校准 ECDF 图对应的检验统计量（Säilynoja、Bürkner 和 Vehtari 2022），我们可以使用：

```
bf.diagnostics.metrics.calibration_log_gamma(
estimates=test_samples, targets=test_sims, )
```

第 4 节讨论了更多诊断以及用于定制这些诊断的其他参数。

3.6. 工作流

作为较低层级 `Approximator` 模块的替代方案，BayesFlow 提供了高级`Workflow` 接口，可通过该接口便捷地调用训练和推断阶段的所有步骤。首先，我们将所有主要对象—— `Simulator`（可选）、 `Adapter(optional)`以及 `Networks`（可选的摘要网络）——汇集到一个对象中：

```
workflow = bf.BasicWorkflow(
simulator=simulator,adapter=adapter,
inference_network=inference_network,
summary_network=summary_network,)
```

接下来，我们只需一行代码即可训练该工作流，例如使用在线训练：

```
workflow.fit_online(epochs=50, batch_size=64, num_batches_per_epoch=100)
```

随后，我们可以一次运行多项诊断，以全面了解获得的近似：

```
test_data = simulator.sample(10)
workflow.plot_default_diagnostics(test_data=test_data, num_samples=100)
workflow.compute_default_diagnostics(test_data=test_data, num_samples=100)
```

要对新数据进行实际推断， `workflow.sample`和 `workflow.log_prob` 的用法与处理 `approximator`时相同。更一般地说， `workflow` 对象仍可访问其组件的所有底层元素和方法，因此，使用它虽然更加方便，却不会显著牺牲灵活性或表达能力。在我们自己的应用中，相较于底层替代方案，我们更倾向于使用工作流。

4. 案例研究：Lotka-Volterra 动力学

为了展示**BayesFlow**的端到端工作流，并采用比第3节中概述的更复杂的场景，本节将以 Lotka-Volterra 系统（Lotka 2002；Wangersky1978；Bunin 2017）作为观测模型，介绍一个案例研究。为清晰易懂起见，我们仅着重介绍最重要的代码片段和输出。该案例研究的完整可复现教程可在以下代码仓库中找到：
https://github.com/bayesflow-org/bf2-paper-case-study。

4.1. 预备知识：选择后端

开始使用 BayesFlow 之前，我们需要选择它支持的一种深度学习后端。一般来说，后端的选择取决于多种因素，其中最重要的是可用硬件和速度要求，以及与现有代码和工作流中使用的外部工具的兼容性。

安装完成后，BayesFlow 会尝试自动检测并使用用户已安装的后端。具体如下所示：

```
import bayesflow as bf
> INFO:bayesflow:Using backend 'torch'
```

要使用此自动检测功能，必须先导入 BayesFlow，再导入 Keras。我们也可以通过在导入 BayesFlow 之前设置 `KERAS_BACKEND`环境变量来覆盖此行为，例如使用 Python 内置的 `os` 模块。当同一环境中安装了多个后端时，这尤其有用。

```
import os
os.environ["KERAS_BACKEND"] = "tensorflow"
import bayesflow as bf
> INFO:bayesflow:Using backend 'tensorflow'
```

在这个案例研究中，我们选择 JAX，因为它性能快速、可复现性最佳且具有函数式纯粹性。我们显式设置此后端，并通过使用 `TF CUDNN DETERMINISTIC`环境变量来确保确定性，从而保证可复现性：

```
import os
os.environ["KERAS_BACKEND"] = "jax"
os.environ["TF_CUDNN_DETERMINISTIC"] = "1"
```

4.2. 观测模型定义

正如我们在图1中所述，定义观测模型包括：1）从指定的先验分布中采样控制参数 θ，2）采样一些测量噪声 ξ，以及 3）通过观测模型生成可观测量 $\mathcal{D} = \mathrm{Sim}(\theta, \xi)$。

Lotka-Volterra（LV）系统（Lotka 2002；Wangersky1978年；Bunin2017年）是计算生态学中的一个经典示例，用于模拟捕食者-猎物种群动力学

这两种物种之间的动态。形式上，LV 系统定义为一组非线性常微分方程：

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - \beta xy, \tag{16}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -\gamma y + \delta xy, \tag{17}$$

其中，x 是猎物种群，y 是捕食者种群，α，β 分别是猎物种群的增长率和衰减率，而 γ，δ 分别是捕食者种群的增长率和衰减率。总之，$\theta = (\alpha, \beta, \gamma, \delta)$ 是 LV 系统方程的控制参数。虽然我们可以使用这些方程直接测量种群时间序列，但无法直接测量控制参数。

在其原始形式下，我们可以按如下方式实现 LV 系统方程：

```
def lotka_volterra_equations(state, t, alpha, beta, gamma, delta):
    x, y = state
    dxdt = alpha * x - beta * x * y
    dydt = - gamma * y + delta * x * y
    return [dxdt, dydt]
```

不过，我们仍需对 LV 系统做出一些假设。首先，需要为这两个物种提供一个初始状态。其次，需要为整个模拟指定一个时间范围。结合这两个假设，我们可以通过使用 `scipy.integrate.odeint` 求解 ODE，完整实现 LV 系统：

```
def ecology_model(alpha, beta, gamma, delta,t_span=[0, 5],
 t_steps=100, initial_state=[1, 1]):
t = np.linspace(t_span[0], t_span[1], t_steps)
state = odeint(lotka_volterra_equations,initial_state, t,
 args=(alpha, beta, gamma, delta))x, y = state.T
return dict(x=x, y=y, t=t)
```

此外，我们还假设观测模型并不完美。在实践中，我们可以通过以下方式体现这一假设：1）在理想观测 `x`、`y` 和 `t` 中引入测量噪声；2）选择部分时间步进行观测，从而纳入计数误差。我们在理想模拟之后引入这些变化，以确保模拟输出符合实际情况。

```
def observation_model(x, y, t, subsample=10, obs_prob=1.0, noise_scale=0.1):
t_steps = x.shape[0]
# Add noise to observations and determine the observed time steps# ...
return {"observed_x": noisy_x[observed_indices],
```

```
        "observed_y": noisy_y[observed_indices],
    "observed_t": t[observed_indices]}
```

为完成观测模型的定义，我们从缩放的 logit 正态分布中采样先验：

```
def prior():
    x = rng.normal(size=4)
    theta = 1 / (1 + np.exp(-x)) * 3.9 + 0.1
    return dict(alpha=theta[0], beta=theta[1], gamma=theta[2], delta=theta[3])
```

我们可以通过调用 `priors()`和 `observation_model(**ecology_model(**prior())`，轻松检查采样先验分布和模拟观测量。

4.3. 模拟器

如第 3.1 节所述，我们可以使用 `bf.make_simulator()` 函数将实现中的所有组件连接起来，构成一个模拟器：

```
simulator = bf.make_simulator([prior, ecology_model, observation_model])
```

然后，我们可以使用 `simulator.sample()` 运行多次模拟。这包括参数集$\{\theta\}$，以及模拟得到的理想（$\{x,\ y,\ t\}$和观测（$\{x_{\text{obs}},\ y_{\text{obs}},\ t_{\text{obs}}\}$时间序列。在训练之前，我们可以采取许多步骤来检查模型，并建议遵循严谨的先验预测检验方法（Gelman 等人 2020年）。为简洁起见，我们将可视化部分轨迹，以观察指定的先验是否会使观测模型产生预期行为。对于 LV 系统，我们预期每对模拟轨迹都呈现总体振荡趋势，彼此之间的差异仅来自先验中采样的参数（见图3）。

4.4. 原始数据的端到端后验估计

最常见的 **BayesFlow** 用途是神经后验估计。在此工作流程中，将模拟器生成的数据集用于训练神经近似器，并从模拟输出中学习参数的后验分布。我们可以通过 `BasicWorkflow`对象轻松设置这一工作流程，具体如 图 2所示，其中我们定义了 `simulator`、 `adapter`、 `summary_network`以及 `inference_network`。

创建适配器

如 第 3.2 节所述，模拟输出可以通过自定义的 `Adapter` 对象整理为适合神经网络的格式，该对象将多个无状态变换组合在一起：

```
adapter = (
    bf.adapters.Adapter()
    .convert_dtype("float64", "float32")
    .drop(["x", "y", "t"])
```

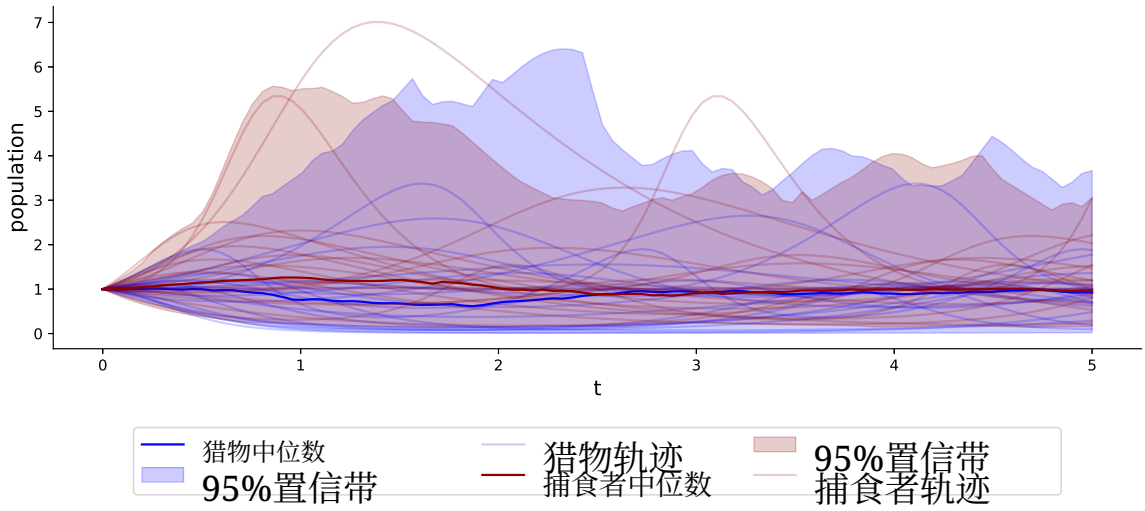


图3：使用采样先验分布生成的 LV 模型轨迹汇总。

```
    .as_time_series(["observed_x", "observed_y", "observed_t"])
    .concatenate(["alpha", "beta", "gamma", "delta"],
                 into="inference_variables")
    .concatenate(["observed_x", "observed_y", "observed_t"],
                 into="summary_variables")
)
```

我们可以通过打印 `adapter` 来检查变换的顺序。下面的列表介绍了此处使用的变换：

- `convert_dtype` 变换会将所有模拟数据的数据类型统一为单精度浮点数，这是深度学习中最常用的精度。
- `drop` 变换会从数据集中移除未使用的可观测量和参数，使其不会在训练和推断期间被纳入。
- `as_time_series` 变换可确保相应的摘要网络将变量正确识别为时间序列。具体来说，它会在末尾添加一个长度为 1 的维度，使得此时倒数第二个维度成为时间序列维度。
- `concatenate`变换会将特定的可观测量或参数沿最后一个维度（默认情况下）汇集到单个 NumPy 数组中。

拼接将时间序列分组为 `summary_variables` ，作为`summary_network`的输入；同时将模型参数分组为 `inference_variables`，作为由 `inference_network`学习其后验分布的变量。

定义网络

接下来，我们可以通过定义 summary_network 和inference_network来构建神经近似器。由于可观测轨迹都是时间序列，因此可以使用 BayesFlow 的现成 TimeSeriesNetwork 将它们压缩为固定长度的汇总统计量。 inference_network的选择相当灵活，但同样简单明了。这里，我们选择 FlowMatching网络，因为它具有较高的性价比。

```
time_series_network = bf.networks.TimeSeriesNetwork(summary_dim=32)
flow_matching = bf.networks.FlowMatching()
```

使用基本工作流进行离线训练

我们将训练和推断所需的所有要素收集到一个 BasicWorkflow对象中，详情见第3.6节：

```
workflow = bf.BasicWorkflow(
simulator=simulator,adapter=adapter,
summary_network=time_series_network,
inference_network=flow_matching)
```

随后，我们使用由 5000 次模拟组成的离线数据集训练神经近似器：

```
training_data = workflow.simulate(5000)
history = workflow.fit_offline(
data=training_data, epochs=50, batch_size=32,
validation data=validation data_                _)
```

在我们的案例中，考虑到模拟预算，训练神经近似器应该非常快速（只需几分钟），无论是否使用图形处理器。

自动诊断

神经近似器训练完成后，我们可以通过检验模型的计算保真度和模型敏感性来验证模型。该 plot_default_diagnostics() 方法结合了模型校准和参数恢复的多种可视化，以及用于确保收敛的损失轨迹（参见图4）：

```
figures = workflow.plot_default_diagnostics(test_data=validation_data,
variable_names=[r"$\alpha$", r"$\beta$", r"$\gamma$", r"$\delta$"],
loss_kwargs={...}, recovery_kwargs={...}, calibration_ecdf_kwargs={...})
```

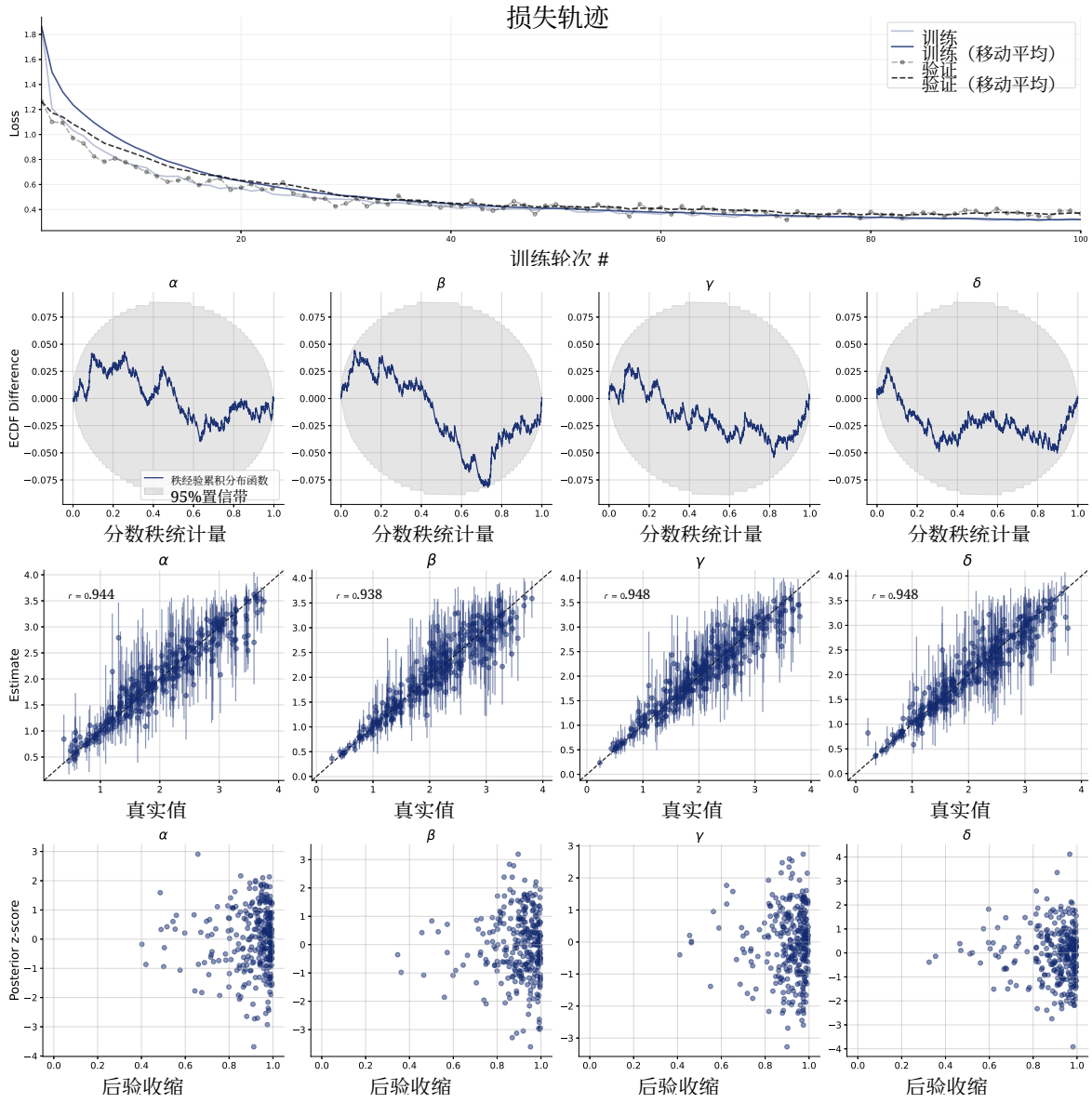


图4：从 `BasicWorkflow`对象自动生成的可视化诊断。从上到下：训练损失和验证损失、校准经验累积分布函数、参数恢复，以及后验 z-分数收缩。

表2：后验估计的诊断指标。

指标	α	β	γ	δ
归一化均方根误差	0.110	0.106	0.100	0.103
对数伽马	1.334	2.036	0.634	0.986
校准误差	0.037	0.040	0.033	0.035
后验收缩	0.944	0.926	0.943	0.941

结果表明，神经网络训练已经收敛，边缘后验分布校准良好，参数恢复和后验收缩程度都很高。总体而言，这意味着我们的神经 ABI 程序已成功准确近似 LV 模型所蕴含的后验分布，至少对于给定的验证数据而言是如此。

图4中显示的诊断正在总结从多个数据集中获得的后验分布。我们还可以更仔细地研究单个数据集的后验分布，例如通过以下方式：

```
estimates = workflow.sample(num_samples=300, conditions=validation_data)

f = bf.diagnostics.plots.pairs_posterior(estimates=estimates,
 targets=validation_data,  dataset_id=0)
```

该 `dataset_id` 参数指明了应在图中显示哪个采样数据集（及其对应的后验分布）（见图5）。

除了图形诊断之外，**BayesFlow**还提供了多种数值诊断，例如归一化均方根误差（NRMSE）、后验收缩和预期校准误差。这些诊断可通过 `compute_default_diagnostics()`方法轻松访问（见表2）：

```
workflow.compute_default_diagnostics(test_data=validation_data)
```

后验预测检验

最后一步，我们可以使用神经近似器生成的后验样本，从 LV 模型中重新模拟数据，并将其与观测数据进行比较。此类后验预测检验通常旨在评估统计模型（由模拟器编码）对给定数据的适用性（Gabry、Simpson、Vehtari、Betancourt 和 Gelman，2019年）。在我们的案例中，由于验证数据是根据模型生成的，因此我们按定义知道该模型是适用的。相反，在验证数据上应用后验预测检验时，我们可以将其作为另一项证据，说明神经近似器确实能够以足够的准确度近似该模型的解析后验。

如图6所示，后验隐含轨迹与观测数据吻合良好，并且比先验隐含轨迹集中得多（请与图3进行比较）。该模型还能够预测超出观测数据时间范围的轨迹；随着预测延伸到更远的未来，不确定性也会增加，这正是校准适当的模型及其对应后验分布所应呈现的特征。

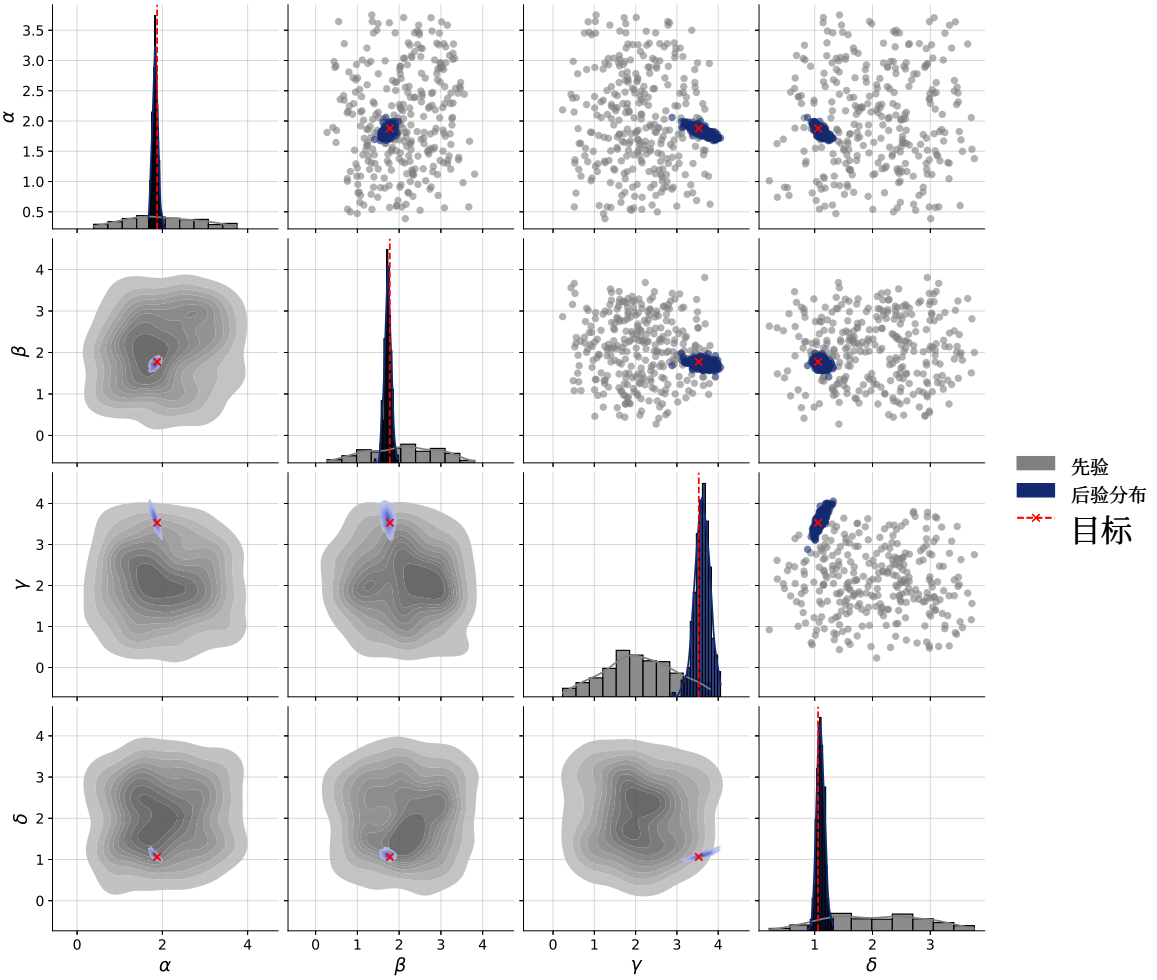


图5：单个数据集的 LV 模型参数估计后验分布，显示其相较于先验向估计目标收缩。

4.5. 使用汇总统计量进行点估计

在某些情况下，完整的后验估计可能不可行，例如，规模庞大或运行缓慢的模拟器输出会限制模拟预算。在这些情况下，后验点估计成为估计完整后验分布的可行替代方案。此外，对于 LV 模型，领域专家已经开发出可供我们利用的汇总统计量，从而加快工作流程。点估计器和固定汇总统计量都能让我们简化神经近似器，并显著提高工作流程的计算效率。在这里，我们将展示 BayesFlow 处理这些情况的能力。

专家汇总统计量

如第 4.4.5 节所示，与捕食者种群和猎物种群相比，观测到的后验预测轨迹呈现出周期性、滞后以及一定程度的同步性（相关性）等特征。因此，可以通过一个简单的函数提取这些观测的汇总统计量。

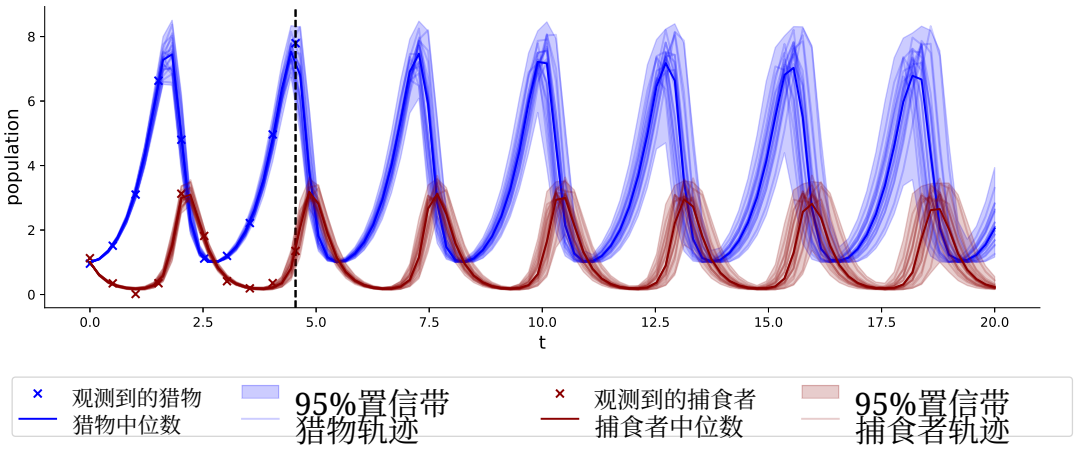


图6：LV 模型对单个验证数据集所隐含的捕食者-猎物种群轨迹的后验预测检验。

```
def expert_stats(observed_x, observed_y, lags=[2,5]):
means = np.array([observed_x.mean(), observed_y.mean()])...
return dict(means=means, log_vars=log_vars,
auto_corrs=auto_corrs, cross_corr=cross_corr,period=T)
```

W我们可以在 BayesFlow 的 `make_simulator()`函数中加入这些汇总统计量 .

```
expert_simulator = bf.make_simulator(
   [prior, ecology_model, observation_model, expert_stats]
)
```

使用 `expert_simulator`重新模拟后，我们可以将这些汇总统计量作为数据提供给神经近似器。

```
samples_with_expert_stats = simulator.sample(3)
keras.tree.map_structure(keras.ops.shape, samples_with_expert_stats)
```

```
{'alpha': (3, 1), ... 'means': (3, 2), 'log_vars': (3, 2),
'auto_corrs': (3, 4),  'cross_corr': (3, 1),  'period': (3, 1)}
```

改进适配器

由于我们已经在模拟器中计算了汇总统计量，因此可以直接在 `Adapter as inference_conditions` 中为它们添加标签。自然，这样就无需再使用完整轨迹作为 `summary_variables`。

```
expert_adapter = (bf.adapters.Adapter()...
.concatenate(["means", "log_vars", "auto_corrs", "cross_corr", "period"],
into="inference_conditions"))
```

简化网络

由于汇总统计量现在已预先计算好，我们的工作流程中不再需要 `summary_network`。对于后验点估计，我们需要一个能够利用某些严格适当评分规则进行推断的推断网络（Gneiting 和 Raftery 2007）。**BayesFlow** 的 `ScoringRuleNetwork` 使我们能够利用均值和分位数等进行推断，并提供了两种这样的评分规则：`MeanScore()` 和 `QuantileScore()`：

```
quantile_levels = np.linspace(0.1, 0.9, 5)
scoring_rule_network = bf.networks.ScoringRuleNetwork(
mean=bf.scoring_rules.MeanScore(),
quantiles=bf.scoring_rules.QuantileScore(quantile_levels))
```

然后，我们可以创建一个新的 `BasicWorkflow`，它会使用 `scoring_rule_network`：

```
workflow = bf.BasicWorkflow(
simulator=expert_simulator,
adapter=expert_adapter,
inference_network=scoring_rule_network)
```

在线训练

除了离线训练，BayesFlow 的 `BasicWorkflow`object 还支持通过 `fit_online()` method 进行在线训练。在线训练不再根据训练数据集的模拟预算计算训练步骤，而是要求用户将每个训练周期的批次数指定为训练步骤数。

```
history = workflow.fit_online(epochs=50,
 num_batches_per_epoch=200, batch_size=32)
```

在线训练的一个好处是，由于训练数据通过模拟即时提供，因此可以最大限度地降低对预先生成的训练数据集过拟合的可能性。然而，代价是时间。与离线训练相比，离线训练的模拟时间受训练数据集和验证数据集规模的限制，而在线训练需要在每一步生成训练数据，从而增加总训练时间。

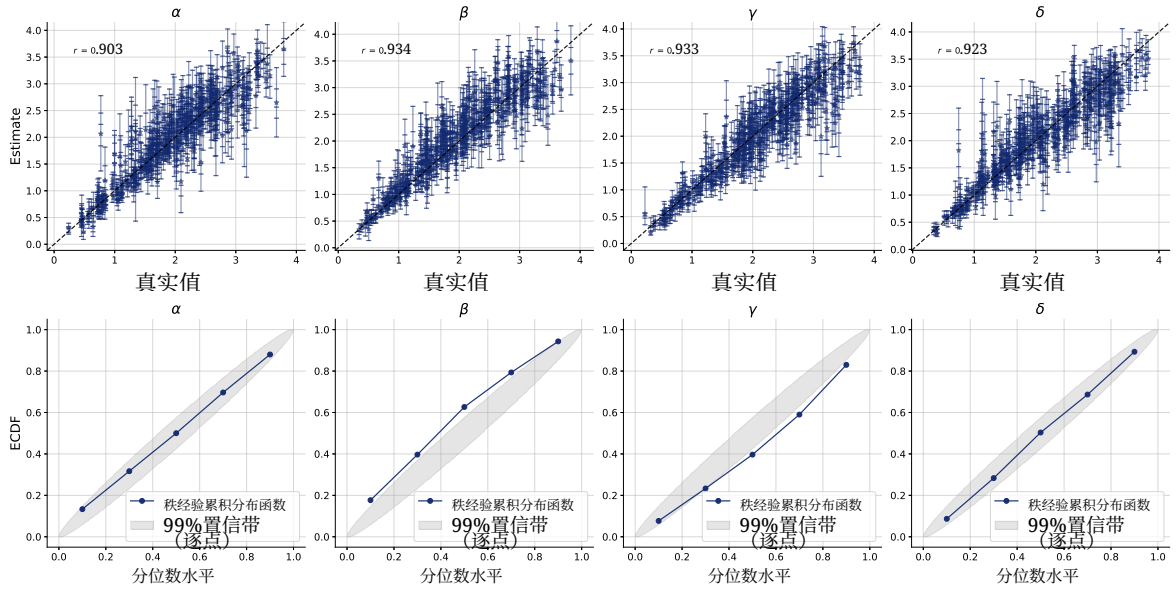


图 7：点估计的手动诊断，包括参数恢复（上）和校准经验累积分布函数（下）。

手动诊断

由于此工作流涉及使用自定义分位数统计的点估计，自动诊断并不适用。尽管如此，BayesFlow 提供了适用于点估计的诊断工具包，我们可以通过自定义指标手动扩展它。和之前一样，可以通过 `bf.diagnostics.loss()`检查损失函数。我们可以获得所有模型参数的后验均值和分位数的估计值。

```
validation_data = expert_simulator.sample(300)

point_estimates = workflow.estimate(conditions=validation_data)
keras.tree.map_structure(keras.ops.shape, point_estimates)

{'alpha': {'mean': (300, 1), 'quantiles': (300, 5, 1)},
 'beta': ...}
```

从这里开始，我们可以仅利用后验均值和分位数估计值进行可视化诊断，以评估神经近似器的准确度和校准情况：

```
f = bf.diagnostics.plots.recovery_from_estimates(
estimates=point_estimates,targets=validation_data,
variable names=variable names_                _)


f = bf.diagnostics.plots.calibration_ecdf_from_quantiles(
estimates=point_estimates,
```

```
    targets=validation_data,
quantile_levels=quantile_levels,
difference=True,variable names=variable names
_              _)
```

乍看之下，可视化诊断与完整后验估计的诊断相似（见图 7）。不过，关键区别在于，这些诊断只能反映点估计所提供的信息。最明显的例子是校准诊断，它仅针对五个估计分位数进行评估。另一个在上述图表中看不出来的区别是，各参数的点估计不包含参数间依赖关系的任何信息。例如，要绘制完整的后验配对图，就需要这些依赖关系。

5. 软件包比较

有若干软件包支持摊销贝叶斯推断。全面比较所有现有方法超出了本文的范围，实际价值也有限。因此，我们将关注范围限定在少数直接竞争者上：这些软件包都在积极维护、广泛使用，并提供最先进的推断方法。摊销方法的比较概览见表 3。

我们特别关注 sbi（Boelts、Deistler、Gloeckler、Tejero-Cantero、Lueckmann、Moss、Steinbach、Moreau、Muratore、Linhart 等，2024年）和 NeuralEstimators.jl（Sainsbury-Dale 等，2024年）。从概念上说，它们与 BayesFlow 最为接近，因为它们支持相关类别的神经估计器，包括后验分布、似然和比率估计器。其他软件包，如 sbijax（Dirmeier、Ulzega、Mira 和 Albert，2024年）、lampe（Rozet、Delaunoy、Miller 等，2021年）、pydelfi（Alsing、Charnock、Feeney 和 Wandelt，2019年）或 carl（Louppe、Cranmer 和 Pavez，2016年），要么只针对范围更有限的模拟式贝叶斯推断方法，要么尚处于开发早期，要么近年来未能持续开发。其他工具，包括 madminer（Brehmer、Kling、Espejo 和 Cranmer，2020年）和 swyft（Miller、Cole、Weniger、Nattino、Ku 和 Grootes，2022a），则更为专门，侧重于特定类型的估计器或应用领域，而非提供通用的模拟式贝叶斯推断框架。

在这些框架中，BayesFlow 独树一帜，原生支持多个深度学习后端。这一能力源于其基于 Keras 3 构建、与后端无关的实现方式，用户可以使用 PyTorch、JAX 或 TensorFlow 后端运行模型。相比之下，sbi 基于 PyTorch，这可能会给使用 TensorFlow 或 JAX 生态系统的用户带来不便。此外，PyTorch 模型的有效编译仍在持续探索中，并且可能因计算图中断或算子支持不足导致的兼容性问题而产生额外开销。因此在实践中，大型 PyTorch 模型往往仍不进行编译，导致性能差距显著，尤其是与使用 JAX 编写的模型相比。与此不同，NeuralEstimators.jl 库面向 Julia 生态系统，并使用 Flux 作为深度学习后端。

与 BayesFlow 一样，sbi 和 NeuralEstimators.jl 都支持平坦式（即非层级式）统计模型。sbi 通过与 pyro 的外部集成支持层级模型的非摊销推断，而 BayesFlow 则原生支持使用层级模型进行训练，这些模型可以包含任意嵌套和交叉的概率分解，以及数量不等的

表 3：基于模拟的推断软件包中摊销方法的功能比较。

	BayesFlow	sbi	NeuralEstimators.jl
支持的深度学习框架			
TensorFlow	✓	✗	✗
PyTorch	✓	✓	✗
JAX	✓	✗	✗
Flux	✗	✗	✓
支持的模型类型			
顺序（扁平）模型	✓	✓	✓
层级模型	✓	✗	✗
多模态模型	✓	(✓)	(✓)
混合类型模型	✓	✓	✗
支持的神经估计器			
后验估计	✓	✓	✓
似然估计	✓	✓	✓
比率估计	✓	✓	✓
评分规则估计	✓	✗	✓
支持的密度估计器			
高斯混合	✓	✓	✓
归一化流	✓	✓	✓
扩散模型	✓	(✓)	✗
流匹配	✓	(✓)	✗
自由形式流	✓	✗	✗
一致性模型	✓	✗	✗
集成	✓	✓	✓
支持的贝叶斯任务			
参数估计	✓	✓	✓
模型比较	✓	✗	✗
模型错设检测	✓	✓	✗

✓ 支持　　(✓) 部分支持　　✗ 不支持

参数。这使得端到端摊销能够无缝实现，无需耗费开发精力来对接多个概率编程环境。

这三个软件包都实现了核心的模拟基础推断估计器——神经后验分布、神经似然和神经比率估计。BayesFlow 还实现了模型比较工作流。BayesFlow 和 NeuralEstimators.jl 也提供后验分布和似然的点估计器，可用于快速原型开发：在需要完整的生成式不确定性量化之前，先判断点估计是否足够。

这三个软件包都支持归一化流：**sbi** 提供掩码自回归流（Papamakarios、Pavlakou 和 Murray 2017年）、神经样条流（Durkan、Bekasov、Murray 和 Papamakarios 2019年），并允许使用由以下项目实现的流：**zuko**(Zuko 开发团队 2025年) 和 **nflows** (Durkan、Bekasov、Murray 和 Papamakarios 2020a)。

BayesFlow 内置仿射耦合流（Dinh等人2017年）和神经样条流，而**NeuralEstimators.jl**仅提供仿射耦合流。此外，**BayesFlow** 支持灵活的潜在空间，例如，使用学生t分布增强鲁棒性，或使用混合分布提升表达能力（Papamakarios 等人 2021年）。

此外，**BayesFlow** 还整合了广泛的现代自由形式生成网络，包括各种流匹配形式（例如最优传输、条件最优传输）、扩散模型（兼容任意调度、引导和组合），以及许多其他近期的模型族（Arruda 等 2025年），例如少步（离散和连续）一致性模型（Schmitt 等 2024b）和自由形式流（Draxler、Sorrenson、Zimmermann、Rousselot 和 Köthe 2024年）。

至关重要的是，BayesFlow 对生成网络架构提供了统一处理，使其所有生成架构都能以即插即用的方式支持后验估计和似然估计。相比之下，sbi 仅通过独立的专用类支持用于后验估计的扩散模型和流匹配，可定制性有限；而 NeuralEstimators.jl 不提供自由形式的生成架构。

不同于 **BayesFlow**，**sbi** 原生支持非摊销方法，例如序贯和多轮后验估计、似然估计与比率估计，以及多种非摊销采样方法，如通过 **pyro** （Bingham、Chen、Jankowiak、Obermeyer、Pradhan、Karaletsos、Singh、Szerlip、Horsfall 和 Goodman 2019年）实现的、重要性抽样、拒绝采样或变分推断。 **BayesFlow** 则主要专注于摊销方法，鼓励直接从训练好的摊销估计器进行采样，或将预训练的神经似然估计/神经比率估计网络接入由 **PyMC** （Abril-Pla 等2023）或 **blackjax** （Cabezas、Corenflos、Lao、Louf、Carnec、Chaudhari、Cohn-Gordon、Coullon、Deng、Duffield 等2024年）提供的金标准马尔可夫链蒙特卡罗采样器中。

虽然这三个软件包都支持将预训练网络打包为一个集成估计器，但只有 BayesFlow 支持将未训练的 网络打包起来进行联合训练，并允许集成成员之间以任意方式共享摘要网络或推断网络。将集成视为一个统一的复合网络，还能最大限度地利用硬件：以单次联合训练步骤取代对各成员的顺序迭代。NeuralEstimators.jl 支持点估计器的联合训练，也允许打包多个未训练的摘要网络[1]，；但在 NeuralEstimators.jl 和 sbi 中，都无法实现其他颇具吸引力的设置，例如使用共享摘要网络训练多个推断网络。

使用存储在 CPU内存中的预计算模拟进行离线训练，已经涵盖了许多应用场景，且所有软件包都支持这种方式。为满足大规模科学问题的需求，BayesFlow 还允许用户从多种训练方案中进行选择，其中最值得一提的是在线训练：模拟器会即时调用，随着训练的进行生成新的样本；此外，也支持使用存储在磁盘而非内存中的样本进行离线训练。类似地，NeuralEstimators.jl 允许用户将采样器作为数据集传入，从而实现即时训练，但磁盘数据集则需要用户自行实现。
最后，所有软件包都提供诊断工具，用于评估训练后神经估计器的性能。**sbi** 和**BayesFlow** 都提供了一套全面的实用工具、指标和图表，用于进行恢复和基于模拟的校准（SBC）检查，包括可选的自定义检验量（Modrák 等，2023），以及模型错设检测（Schmitt等，2023）和与参考分布的基于样本的比较，例如 C2ST（Lopez-

[1]使用 Flux 内置的 `Parallel`层。

Paz 和 Oquab，2018）和最大均值差异（Gretton、Borgwardt、Rasch、Schölkopf 和 Smola，2012）。**NeuralEstimators.jl** 主要侧重于评估贝叶斯风险和估计值的频率学派覆盖率，并附带便捷的恢复图。

6. 结论

在这项工作中，我们介绍了 BayesFlow 版本 2，这是一个与后端无关、功能丰富的框架，适用于可扩展的摊销贝叶斯推断工作流。我们通过概述软件的核心组件，并将该框架应用于一个具有代表性的案例研究，展示了 BayesFlow 直观而全面的 API。通过原生集成多种最新的深度学习架构，例如各类扩散模型和流匹配，BayesFlow 能够针对广泛的任务，实现快速、灵活、端到端的贝叶斯推断。我们还展示了 BayesFlow 的 `diagnostics`模块，该模块可全面检查训练后的估计器，并通过数值诊断和可视化诊断确保估计质量。

BayesFlow 提供了精简的接口，紧密对应成熟的摊销贝叶斯推断术语，并仔细协调统计学和摊销贝叶斯推断中的命名惯例与现代机器学习中的惯例。模拟器、神经估计器和诊断工具各自的职责划分清晰，为用户提供了结构完善的工作流，涵盖生成模型设定、数据预处理、训练和推断。我们认为，通过减少概念负担、降低入门门槛，BayesFlow 的新设计能够吸引更广泛的科学界群体，让依赖基于模型的推断开展研究的科学家也能受益。归根结底，这一精简接口奠定了清晰的概念基础，既方便新用户使用，也为资深实践者保留了灵活性。

展望未来，我们有若干计划，旨在进一步提升**BayesFlow**的功能。生成建模领域正在迅速发展，新的架构及其改进不断涌现。随着这些方法陆续问世并展现出对摊销贝叶斯推断（ABI）有前景的结果，我们将继续在**BayesFlow**中实现它们。这也包括模块化架构组合，使其反映所近似统计模型的概率对称性（Habermann 等 2024年；Kucharský 和 Bürkner 2025年；Elsemüller、Schnuerch、Bürkner 和 Radev 2024b），以及专为摊销贝叶斯推断设计、能够提升其准确度和鲁棒性的损失函数（Schmitt、Habermann、Bürkner、Köthe 和 Radev 2024a；Mishra等2025年；Elsemüller等2025年）。

此外，我们将实现一些方法，以极低的计算开销在推断过程中对摊销贝叶斯推断结果进行事后校正，例如先进的重要性抽样方法（Vehtari等 2024年；Li 等 2024年）。此外，我们目前正在构建 **BayesFlow** 与常见概率编程语言的接口，例如Stan（Stan 开发团队 2025年）或 **PyMC**（Abril-Pla 等2023）。这将大幅扩大**BayesFlow**的潜在用户群，并整体简化模拟器及相应对数密度的设定。最后，我们将继续实现开箱即用的现代训练方法，例如自动低精度预训练，以支持高效、灵活地扩展到规模最大的贝叶斯问题。

致谢

本研究部分由美国国家科学基金会资助（资助编号 2448380）、德国研究联合会（DFG，德国研究基金会）项目 528702768 和 508399956，以及 DFG 协同研究中心 391（能源与交通转型的时空统计）资助——520388526。

参考文献

Abril-Pla O, Andreani V, Carroll C, Dong, Kochurov M, Kumar R, Lao J, Luhmann CC, Martin OA, Osthege M, Vieira R, Wiecki T, Zinkov R（2023年）。“PyMC：Python 中现代且全面的概率编程框架。”PeerJ Computer Science，9，e1516。doi:10.7717/peerj-cs.1516。Alsing J, Charnock T, Feeney S, Wandelt B（2019年）。“借助神经密度估计器和主动学习实现快速无似然宇宙学研究。”《皇家天文学会月刊》，488(3)，4440–4458。ISSN 0035-8711。doi:10.1093/mnras/stz1960。https://academic.oup.com/mnras/article-pdf/488/3/4440/29113037/stz1960.pdf，URL https://doi.org/10.1093/mnras/stz1960。Arruda J, Bracher N, Köthe U, Hasenauer J, Radev ST（2025年）。“基于模拟的推断中的扩散模型：教程综述。”arXiv 预印本 arXiv:2512.20685。Bingham E, Chen JP, Jankowiak M, Obermeyer F, Prad han N, Karaletsos T, Singh R, Szerlip PA, Horsfall P, Goodman ND（2019年）。“Pyro：深度通用概率编程。”《机器学习研究杂志》，20，28:1–28:6。URL http://jmlr.org/papers/v20/18-403.html。Boelts J, Deistler M, Gloeckler M, Tejero-Cantero Á, Lueckmann JM, Moss G, Steinbach P, Moreau T, Muratore F, Linhart J 等（2024年）。“sbi reloaded：基于模拟的推断工作流工具包。”arXiv 预印本 arXiv:2411.17337。Boelts J, Lueckmann JM, Gao R, Macke JH（2022）。“用于决策模型的灵活、高效的基于模拟的推断。”eLife，11，e77220。Brehmer J, Kling F, Espejo I, Cranmer K（2020）。“MadMiner：用于粒子物理学的基于机器学习的推断。”《大科学计算与软件》，4(1)，3。Brooks S, Gelman A, Jones G, Meng XL（2011）。《马尔可夫链蒙特卡罗手册》。CRC 出版社。Bunin G（2017）。“具有 Lotka-Volterra 动力学的生态群落。”《物理评论E》，95(4)，042414。Bürkner PC, Schmitt M, Radev ST（2025年）。“统计工作流中的模拟。”arXiv 预印本 arXiv:2503.24011。Bürkner PC, Scholz M, Radev ST（2023）。“有些模型很有用，但我们如何知道该用哪个？迈向统一的贝叶斯模型分类体系。”《统计综论》，17，216–310。 doi:10.1214/23-SS145。Cabezas A, Corenflos A, Lao J, Louf R, Carnec A, Chaudhari K, Cohn-Gordon R, Coullon J, Deng W, Duffield S 等（2024年）。“BlackJAX：JAX 中可组合的贝叶斯推断。”arXiv 预印本 arXiv:2402.10797。Chollet F 等（2025年）。“Keras 3。” https://keras.io/keras_3/。Cranmer K, Brehmer J, Louppe G（2020）。“基于模拟的推断前沿。”《美国国家科学院院刊》，117(48)，30055–30062。

Deistler M, Boelts J, Steinbach P, Moss G, Moreau T, Gloeckler M, Rodrigues PLC, Linhart J, Lappalainen JK, Miller BK, Gonçalves PJ, Lueckmann JM, Schröder C, Macke JH（2025年）。“基于模拟的推断：实用指南。”arXiv *[stat.ML]*. 2508.12939, URL http://arxiv.org/abs/2508.12939。Diggle PJ, Gratton RJ（1984年）。“隐式统计模型的蒙特卡罗推断方法。”《皇家统计学会学报B辑：统计方法学》，46(2)，193–212。Dinh L, Krueger D, Bengio Y（2015年）。“NICE：非线性独立成分估计。”1410.8516, URL https://arxiv.org/abs/1410.8516。Dinh L, Sohl-Dickstein J, Bengio S（2017年）。“使用 Real NVP 进行密度估计。”1605.08803，URL https://arxiv.org/abs/1605.08803。Dirmeier S, Ulzega S, Mira A, Albert C（2024年）。“使用 Python 软件包 sbijax 进行基于模拟的推断。”2409.19435，URL https://arxiv.org/abs/2409.19435。Draxler F, Sorrenson P, Zimmermann L, Rousselot A, Köthe U（2024年）。“自由形式流：让任意架构都成为归一化流。”2310.16624，URL https://arxiv.org/abs/2310.16624。Durkan C, Bekasov A, Murray I, Papamakarios G（2019年）。“神经样条流。”载于 H Wallach, H Larochelle, A Beygelzimer, F dAlché-Buc, E Fox, R Garnett（编），《神经信息处理系统进展》第32卷。Curran Associates, Inc. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/7ac71d433f282034e088473244df8c02-Paper.pdf。Durkan C, Bekasov A, Murray I, Papamakarios G（2020a）。“nflows：PyTorch 中的归一化流。”doi:10.5281/zenodo.4296287。URL https://doi.org/10.5281/zenodo.4296287。Durkan C, Murray I, Papamakarios G（2020b）。“关于无似然推断的对比学习。”载于《国际机器学习会议》，第2771–2781页。Elsemüller L, Olischläger H, Schmitt M, Bürkner PC, Koethe U, Radev ST（2024a）。“敏感性感知摊销贝叶斯推断。”《机器学习研究汇刊》。ISSN 2835-8856。URL https://openreview.net/forum?id=Kxtpa9rvM0。Elsemüller L, Pratz V, von Krause M, Voss A, Bürkner PC, Radev ST（2025年）。“无监督领域自适应能否提升摊销贝叶斯推断的鲁棒性？一项系统评估。”《机器学习研究汇刊》。ISSN 2835-8856。URL https://openreview.net/forum?id=ewgLuvnEw6。Elsemüller L, Schnuerch M, Bürkner PC, Radev ST（2024b）。“一种用于比较贝叶斯层级模型的深度学习方法。”Psychological Methods。在线先行发表。 doi:10.1037/met0000645。Fengler A, Govindarajan LN, Chen T, Frank MJ（2021年）。“用于快速推断认知神经科学模拟模型的似然近似网络（LAN）。”eLife，10，e65074。

Frazier DT、Kelly R、Drovandi C、Warne DJ（2024）。“神经后验估计与似然估计的统计准确性。” arXiv 预印本 arXiv:2411.12068。Gabry J、Simpson D、Vehtari A、Betancourt M、Gelman A（2019）。“贝叶斯工作流中的可视化。”《英国皇家统计学会杂志：A辑（社会统计学）》，182(2)，389–402。 doi:10.1111/rssa.12378。Gelman A、Vehtari A、Simpson D、Margossian CC、Carpenter B、Yao Y、Kennedy L、Gabry J、Bürkner PC、Modrák M（2020）。“贝叶斯工作流。” arXiv 预印本 arXiv:2011.01808。Gershman S、Goodman N（2014）。“概率推理中的摊销推断。”载于认知科学学会年会论文集，第36卷。Gloeckler M、Deistler M、Macke JH（2023）。“摊销贝叶斯推断的对抗鲁棒性。”载于国际机器学习会议，页11493–11524。PMLR。Gneiting T、Raftery AE（2007）。“严格适当评分规则、预测与估计。”《美国统计学会杂志》，102(477)，359–378。Gretton A、Borgwardt KM、Rasch MJ、Schölkopf B、Smola A（2012）。“核双样本检验。”《机器学习研究期刊》，13(null)，723–773。ISSN 1532-4435。Gu A、Dao T（2024）。“Mamba：使用选择性状态空间进行线性时间序列建模。”载于首届语言建模会议。Habermann D、Schmitt M、Kühmichel L、Bulling A、Radev ST、Bürkner PC（2024）。“摊销贝叶斯多层模型。” arXiv 预印本。 doi:10.48550/arXiv.2408.13230。Hermans J、Begy V、Louppe G（2020）。“使用摊销近似比率估计器进行无似然MCMC。”载于第37届国际机器学习会议论文集，页4239–4248。Ho J、Jain A、Abbeel P（2020）。“去噪扩散概率模型。”2006.11239，URL https://arxiv.org/abs/2006.11239。Huang D、Bharti A、Souza A、Acerbi（2023）。“在模型设定错误条件下，为基于模拟的推断学习稳健统计方法。”《神经信息处理系统进展》。Jeffrey N、Wandelt BD（2024）。“证据网络：用于快速摊销神经贝叶斯模型比较的简单损失函数。”《机器学习：科学与技术》，5(1)，015008。Kass RE、Raftery AE（1995）。“贝叶斯因子。”《美国统计学会杂志》，90(430)，773–795。Kingma D、Gao R（2023）。“将扩散目标函数理解为带有简单数据增强的证据下界。”《神经信息处理系统进展》，36，65484–65516。Kingma DP、Welling M（2013）。“自编码变分贝叶斯。” arXiv 预印本 arXiv:1312.6114。Kucharský Š、Bürkner PC（2025）。“摊销贝叶斯混合模型。” arXiv 预印本 arXiv:2501.10229。

Kucharský Š、Mishra A、Habermann D、Radev ST、Bürkner PC（2025）。“使用自一致性提升摊销模型比较的准确度。” arXiv 预印本。 doi:10.48550/arXiv.2508.20614。Lavin A、Krakauer D、Zenil H、Gottschlich J、Mattson T、Brehmer J、Anandkumar A、Choudry S、Rocki K、Baydin AG 等（2021）。“仿真智能：迈向新一代科学方法。” arXiv 预印本 arXiv:2112.03235。Le TA、Baydin AG、Wood F（2017）。“推理编译与通用概率编程。”载于人工智能与统计学，页1338–1348。PMLR。Lee J、Lee Y、Kim J、Kosiorek AR、Choi S、Teh YW（2019）。“集合变换器：一种基于注意力的置换不变神经网络框架。”1810.00825，URL https://arxiv.org/abs/1810.00825。Li C、Vehtari A、Bürkner PC、Radev ST、Acerbi L、Schmitt M（2024）。“摊销贝叶斯工作流。” arXiv 预印本 arXiv:2409.04332。Lipman Y、Chen RTQ、Ben-Hamu H、Nickel M、Le M（2023）。“用于生成建模的流匹配。”2210.02747，URL https://arxiv.org/abs/2210.02747。Lopez-Paz D、Oquab M（2018）。“重新审视分类器双样本检验。” doi:10.48550/arXiv.1610.06545。1610.06545，URL http://arxiv.org/abs/1610.06545。Lotka AJ（2002）。“对周期反应理论的贡献。”《物理化学杂志》，14(3)，271–274。Louppe G、Cranmer K、Pavez J（2016）。“carl：无似然推断工具箱。” doi:10.5281/zenodo.47798。URL http://dx.doi.org/10.5281/zenodo.47798。Lueckmann JM、Bassetto G、Karaletsos T、Macke JH（2019）。“使用模拟器网络进行无似然推断。”载于近似贝叶斯推断进展研讨会，页32–53。PMLR。MacKay D（2003）。《信息论、推断与学习算法》。剑桥大学出版社。Miller BK、Cole A、Weniger C、Nattino F、Ku O、Grootes MW（2022a）。“swyft：用于 Python 的截断边缘神经比率估计。”《开放源代码软件杂志》，7(75)，4205。 doi:10.21105/joss.04205。URL https://doi.org/10.21105/joss.04205。Miller BK、Weniger C、Forré P（2022b）。“对比神经比率估计。”《神经信息处理系统进展》，35，3262–3278。Mishra A、Habermann D、Schmitt M、Radev ST、Bürkner PC（2025）。“在未标注数据上使用自一致性损失的稳健摊销贝叶斯推断。” arXiv 预印本 arXiv:2501.13483。Modrák M、Moon AH、Kim S、Bürkner PC、Huurre N、Faltejsková K、Gelman A、Vehtari A（2023）。“贝叶斯计算的基于模拟的校准检验：检验量的选择如何影响灵敏度。”《贝叶斯分析》。 doi:10.1214/23-BA1404。

Musslick S, Bartlett LK, Chandramouli SH, Dubova M, Gobet F, Griffiths TL, Hullman J, King RD, Kutz JN, Lucas CG, 等（2025年）。《科学实践自动化：机遇、挑战与影响》。发表于《美国国家科学院院刊》，122(5)，e2401238121。Paige B, Wood F（2016年）。《图模型中的序贯蒙特卡洛推断网络》。载于《国际机器学习会议》，第3040–3049页。PMLR。Papamakarios G, Nalisnick E, Rezende DJ, Mohamed S, Lakshminarayanan B（2021年）。《用于概率建模与推断的归一化流》。发表于《机器学习研究期刊》，22(57)，1–64。Papamakarios G, Pavlakou T, Murray I（2017年）。《用于密度估计的掩码自回归流》。载于 I Guyon, UV Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, R Garnett（编），《神经信息处理系统进展》，第30卷。Curran Associates, Inc. 网址 https://proceedings.neurips.cc/paper_files/paper/2017/file/6c1da886822c67822bcf3679d04369fa-Paper.pdf。Papamakarios G, Sterratt D, Murray I（2019年）。《序贯神经似然：利用自回归流实现快速无似然推断》。载于《第22届国际人工智能与统计会议》，第837–848页。PMLR。Pudlo P, Marin JM, Estoup A, Cornuet JM, Gautier M, Robert CP（2016年）。《利用随机森林可靠地进行ABC模型选择》。发表于《生物信息学》，32(6)，859–866。R核心团队（2025年）。R：用于统计计算的语言与环境。奥地利维也纳：R统计计算基金会。网址 https://www.R-project.org/。Radev ST, D'Alessandro M, Mertens UK, Voss A, Köthe U, Bürkner PC（2021年）。《基于证据深度学习的摊销贝叶斯模型比较》。2004.10629，网址 https://arxiv.org/abs/2004.10629。Radev ST, Schmitt M, Pratz V, Picchini U, Köthe U, Bürkner PC（2023a）。《JANA：复杂贝叶斯模型的联合摊销神经近似》。载于《人工智能中的不确定性》，第1695–1706页。PMLR。Radev ST, Schmitt M, Schumacher L, Elsemüller L, Pratz V, Schälte Y, Köthe U, Bürkner PC（2023b）。《BayesFlow：利用神经网络实现摊销贝叶斯工作流》。发表于《开放源代码软件杂志》，8(89)，5702。Ranganath R, Gerrish S, Blei D（2014）。《黑箱变分推断》。载于《第十七届国际人工智能与统计会议论文集》，第814–822页。PMLR。Rombach R, Blattmann A, Lorenz D, Esser P, Ommer B（2022）。《利用潜在扩散模型合成高分辨率图像》。2112.10752，网址 https://arxiv.org/abs/2112.10752。Rozet F, Delaunoy A, Miller B, 等（2021年）。《LAMPE：无似然摊销后验估计》。 doi:10.5281/zenodo.8405782。网址 https://pypi.org/project/lampe。

Rue H, Martino S, Chopin N（2009年）。《通过积分嵌套拉普拉斯近似对潜在高斯模型进行近似贝叶斯推断》。发表于皇家统计学会学报B辑：统计方法学，71(2)，319–392。Sainsbury-Dale M, Zammit-Mangion A, Huser R（2024年）。《利用神经贝叶斯估计器进行无似然参数估计》。发表于《美国统计学家》，78(1)，1–14。ISSN 0003-1305。doi:10.1080/00031305.2023.2249522。网址 https://doi.org/10.1080/00031305.2023.2249522。Schad DJ, Betancourt M, Vasishth S（2021年）。《迈向认知科学中有原则的贝叶斯工作流》。发表于 Psychological Methods，26(1)，103。Schmitt M, Bürkner PC, Köthe U, Radev ST（2023）。《利用神经网络检测摊销贝叶斯推断中的模型设定错误》。载于德国模式识别会议（DAGM），第541–557页。施普林格。Schmitt M, Habermann D, Bürkner PC, Köthe U, Radev ST（2024a）。《利用自一致性实现数据高效的摊销贝叶斯推断》。国际机器学习会议。Schmitt M, Pratz V, Köthe U, Bürkner PC, Radev ST（2024b）。《用于可扩展、快速基于模拟的推断的一致性模型》。载于《神经信息处理系统进展》，37，126908–126945。Siahkoohi A, Rizzuti G, Orozco R, Herrmann FJ（2023）。《利用基于物理的潜在分布校正实现可靠的摊销变分推断》。发表于《地球物理学》，88(3)，R297–R322。Song Y, Dhariwal P（2023）。《改进一致性模型训练的技术》。arXiv预印本 arXiv:2310.14189。Song Y, Dhariwal P, Chen M, Sutskever I（2023）。《一致性模型》。载于 A Krause, E Brunskill, K Cho, B Engelhardt, S Sabato, J Scarlett（编），《第40届国际机器学习会议论文集》，《机器学习研究论文集》第202卷，第32211–32252页。PMLR。网址 https://proceedings.mlr.press/v202/song23a.html。Song Y, Durkan C, Murray I, Ermon S（2021年）。《基于得分的扩散模型的最大似然训练》。发表于《神经信息处理系统进展》，34，1415–1428。Song Y, Sohl-Dickstein J, Kingma DP, Kumar A, Ermon S, Poole B（2020）。《通过随机微分方程进行基于得分的生成建模》。arXiv预印本 arXiv:2011.13456。Stan 开发团队（2025年）。《Stan建模语言用户指南与参考手册，2.37.0》。网址 https://mc-stan.org。Stuhlmüller A, Taylor J, Goodman N（2013）。《学习随机逆》。发表于《神经信息处理系统进展》，26。Säilynoja T, Bürkner PC, Vehtari A（2022）。《离散均匀性的图形检验及其在拟合优度评估和多样本比较中的应用》。发表于《统计与计算》，32(2)。ISSN 1573-1375。 doi:10.1007/s11222-022-10090-6。

Talts S、Betancourt M、Simpson D、Vehtari A、Gelman A（2020）。“通过基于模拟的校准验证贝叶斯推断算法。”1804.06788，网址 https://arxiv.org/abs/1804.06788。Tong A、Fatras K、Malkin N、Huguet G、Zhang Y、Rector-Brooks J、Wolf G、Bengio Y（2024）。“利用小批量最优传输改进并泛化基于流的生成模型。”2302.00482，网址 https://arxiv.org/abs/2302.00482。Vehtari A、Ojanen J（2012）。“用于模型评估、选择与比较的贝叶斯预测方法综述。”《统计综论》，6，142–228。doi:10.1214/12-SS102。Vehtari A、Simpson D、Gelman A、Yao Y、Gabry J（2024）。“帕累托平滑重要性采样。”《机器学习研究期刊》，25(72)，1–58。von Krause M、Radev ST、Voss A（2022）。“对逾百万名参与者的分析显示，心理速度在60岁之前一直处于高水平。”《自然·人类行为》，6(5)，700–708。Walker SG（2013）。“误设定模型下的贝叶斯推断。”《统计规划与推断期刊》，143(10)，1621–1633。doi:10.1016/j.jspi.2013.05.013。Wangersky PJ（1978）。“Lotka-Volterra种群模型。”《生态学与系统学年度评论》，9，189–218。Ward D、Cannon P、Beaumont M、Fasiolo M、Schmon S（2022）。“鲁棒的神经后验估计与统计模型批评。”《神经信息处理系统进展》，35，33845–33859。Wen Q、Zhou T、Zhang C、Chen W、Ma Z、Yan J、Sun L（2022）。“时间序列中的Transformer：综述。”arXiv预印本：2202.07125。Wu Y、Radev ST、Tuerlinckx F（2024）。“测试并提升认知模型摊销贝叶斯推断的鲁棒性。”arXiv预印本 arXiv:2412.20586。Ye Y、Pandey A、Bawden C、Sumsuzzman DM、Rajput R、Shoukat A、Singer BH、Moghadas SM、Galvani AP（2025）。“将人工智能与机制流行病学建模相结合：机遇与挑战范围综述。”《自然·通讯》，16(1)，581。Zaheer M、Kottur S、Ravanbakhsh S、Poczos B、Salakhutdinov R、Smola A（2018）。“深度集合。”1703.06114，网址 https://arxiv.org/abs/1703.06114。Zammit-Mangion A、Sainsbury-Dale M、Huser R（2025）。“用于摊销推断的神经方法。”《统计及其应用年度评论》，12(1)，311–335。ISSN 2326-8298,2326-831X。doi:10.1146/annurev-statistics-112723-034123。网址 http://dx.doi.org/10.1146/annurev-statistics-112723-034123。Zhang Y、Mikelsons L（2023）。“利用随机贝叶斯流求解随机逆问题。”载于《2023年IEEE/ASME先进智能机电一体化国际会议（AIM）》，第966–972页。Zuko 开发团队（2025）。“Zuko——PyTorch中的归一化流。”网址 https://zuko.readthedocs.io/stable/。

隶属机构：
Lars Kühmichel 统计系 多特蒙德工业大学
德国多特蒙德 44227 电子邮件：
lars.kuehmichel@tu-dortmund.de网址：
https://github.com/LarsKue