centralelille

Centrale Lille

THÈSE

présentée en vue d'obtenir le grade de

DOCTEUR

dans la spécialité

Traitement du Signal et des Images

par

Maxime BOUTON

Doctorat délivré par Centrale Lille

Titre de la thèse :

Distributed Plug-and-Play MCMC algorithms for high-dimensional Bayesian inference

Algorithmes MCMC Plug-and-Play distribués pour l'inférence bayésienne en grande dimension

Soutenue publiquement le **9 décembre 2025** devant le jury d'examen :

Rapporteure	Julie Delon	Professeure	Université Paris Cité
Rapporteur	Marcelo Pereyra	Professeur	Heriot-Watt University (UK)
Examinateur	Andrés Almansa	Directeur de recherche	Université Paris Cité
Examinateur (Président du jury)	Bruno Galerne	Professeur	Université d'Orléans
Invitée	Audrey Repetti	Maîtresse de conférences	Heriot-Watt University (UK)
Directeur	Pierre Chainais	Professeur	Centrale Lille Institut
Encadrant	Pierre-Antoine Thouvenin	Maître de conférences	Centrale Lille Institut

Thèse préparée dans le laboratoire CRIStAL

École Graduée MADIS-631

"Je sers la science et c'est ma joie."

–Le disciple, *Léonard*, Turk & De Groot

Remerciements

Au terme de ces trois années de labeur, il me tenait à cœur de remercier celles et ceux qui m'ont accompagné, de près ou de loin, dans cette aventure.

Tout d'abord, j'exprime ma profonde gratitude à ce jury d'exception pour l'évaluation de ce travail. Je remercie tout particulièrement Mme Julie Delon et M. Marcelo Pereyra d'avoir accepté la charge de rapporteurs, et pour leurs retours aussi pertinents que bienveillants.

J'ai une pensée chaleureuse pour l'ensemble de mes collègues de l'équipe SigMA, qui ont su créer un environnement de travail si stimulant et convivial. Je n'oublie pas mes co-bureaux et compagnons de galère : Nico, Thomas, Maxence et Markus. Merci pour ces discussions passionnantes, les pauses café revigorantes, les rigolades, et aussi pour ces quelques Zazous bien mérités ! J'en profite pour embrasser l'ex-team lilloise : Coco, Loulou, Elliot et Baba, pour leurs encouragements et leur enthousiasme.

C'est avec beaucoup d'émotion que je remercie Pierre et Pierre-Antoine, mes directeurs de thèse, ainsi qu'Audrey, qui l'était tout comme. Merci pour votre confiance indéfectible et vos conseils avisés, et ce, dès mon arrivée à Centrale. Vous m'avez épaulé, surtout quand les résultats n'étaient pas au rendez-vous, me poussant toujours à persévérer. En plus d'être de brillants scientifiques, vous êtes des personnes formidables. Servir la science à vos côtés a été un honneur et un réel plaisir. Merci, tout simplement.

Je ne pourrais conclure sans m'adresser aux membres de ma famille, présents physiquement ou en pensée. Merci à mes parents pour leur accompagnement sans faille et pour m'avoir toujours encouragé à donner le meilleur de moi-même, convaincus que l'effort finirait par payer. Vous aviez raison. À toi Pauline, ma grande sœur et mon modèle depuis toujours : je ne serais jamais arrivé là sans ton soutien. Je vous dédie ce succès, car je suis convaincu qu'il est aussi le vôtre.

Finalmente, muchas gracias a ti mi vidita, Andi, por estar siempre ahí, por tu paciencia infinita, tu amor incondicional y por creer en mí incluso cuando yo dudaba. Gracias por cuidar de mí en los momentos difíciles y por celebrar conmigo cada pequeño logro. Eres mi roca y mi mayor inspiración. Te amo mi Patita.

Résumé substantiel

Contexte et problématique de la thèse

Les problèmes inverses, tels que la reconstruction d'images ou la restauration de signaux, consistent à estimer des paramètres inconnus à partir d'observations dégradées et souvent bruitées. En les formulant dans un cadre bayésien [Robert, 2007], on cherche à caractériser la distribution *a posteriori* des paramètres inconnus. Celle-ci combine un terme de vraisemblance, qui décrit le modèle d'observation, et une distribution *a priori*, qui incorpore des connaissances préalables sur la solution recherchée.

La qualité des estimations dépend fortement du choix de la distribution *a priori*. Les modèles classiques, tels que la variation totale [Rudin et al., 1992] ou la régularisation de Tikhonov [Benning et al., 2018], imposent des contraintes simples (parcimonie, régularité,...). Cependant, ils ne parviennent pas à modéliser les dépendances complexes présentes dans des données structurées, en particulier les images naturelles. Pour mieux capturer ces structures, une approche possible consiste à introduire des *a priori* appris à partir de données. Les réseaux neuronaux profonds, capables d'apprendre des représentations hiérarchiques riches, constituent une solution adaptée [Ongie et al., 2020]. Leur intégration dans les algorithmes d'optimisation a conduit au développement des méthodes *Plug-and-Play* (PnP) [Kamilov et al., 2023]. Celles-ci remplacent la régularisation explicite par une étape de débruitage réalisée par un réseau neuronal préentraîné. Ces approches ont en particulier montré des performances remarquables en termes de qualité de reconstruction et de rapidité de convergence.

Toutefois, les méthodes d'optimisation ne fournissent qu'une estimation ponctuelle. Elles ne permettent pas de quantifier l'incertitude sur les estimations générées, ce qui est essentiel en l'absence de données de validation. Contrairement à ces méthodes, les algorithmes de Monte Carlo par chaînes de Markov (MCMC) [Meyn et al., 2009] visent à échantillonner la distribution *a posteriori*. Ce procédé permet ainsi de construire des intervalles de crédibilité ou d'estimer des variances a pos-

teriori. Des variantes inspirées des approches PnP, comme PnP-ULA [Laumont et al., 2022], ont été proposées pour intégrer des *a priori* appris. Néanmoins, leur extension à des problèmes de très grande dimension reste difficile. Les coûts de calcul et de mémoire associés à la génération et au stockage d'échantillons deviennent rapidement prohibitifs. De plus, la convergence des chaînes se dégrade lorsque la dimension augmente, ce qui entraîne des temps de mélange très longs.

Pour surmonter ces difficultés, plusieurs stratégies ont été explorées. D'une part, l'introduction de noyaux de transition plus efficaces, fondés sur la dynamique de Langevin [Durmus et al., 2018 ; Renaud et al., 2025], permet d'améliorer la convergence. Les approches d'augmentation de données, telles que l'augmentation de données asymptotiquement exacte (AXDA) [Vono et al., 2021], introduisent des variables auxiliaires qui permettent de scinder le problème en sous-problèmes plus simples à traiter. D'autre part, le calcul distribué [Eijkhout, 2023] offre une voie prometteuse pour traiter des problèmes de très grande dimension. Il permet de répartir la charge de calcul et de mémoire sur plusieurs unités, comme des GPUs. Cependant, la nature séquentielle des chaînes de Markov complique la parallélisation directe. Les approches existantes, telles que les méthodes de consensus, la fusion d'échantillons indépendants ou le découpage de l'espace des variables, présentent chacune des limites. Aucune ne permet actuellement d'exploiter efficacement des *a priori* profonds dans un cadre MCMC distribué.

Contributions principales

Cette thèse propose un cadre à la fois méthodologique et pratique pour l'échantillonnage MCMC *Plug-and-Play* distribué. L'objectif est de permettre une inférence bayésienne en grande dimension à l'aide d'*a priori* encodés par des réseaux de neurones. La méthodologie s'appuie sur les récents progrès en MCMC distribuée, notamment l'approche de Thouvenin et al., 2024. Celle-ci exploite les structures d'indépendance conditionnelle et la localité des opérateurs propres aux problèmes inverses. Ces propriétés permettent une mise en œuvre efficace dans un paradigme *Single Program Multiple Data* (SPMD), où la charge de calcul est répartie sur plusieurs GPUs. Contrairement à la plupart des approches distribuées existantes, le cadre proposé conserve les garanties théoriques de l'algorithme séquentiel sous-jacent. Les échantillons produits possèdent donc les mêmes propriétés statistiques que ceux obtenus avec une implémentation standard. La méthodologie, illustrée à partir de PnP-ULA [Laumont et al., 2022] utilisant des réseaux de neurones convolutifs, peut s'étendre naturellement à une famille plus

large d'algorithmes MCMC partageant des propriétés de localité similaires.

Une contribution majeure de ce travail réside dans la conception d'une bibliothèque Python modulaire dédiée à cette méthodologie. Les bibliothèques MCMC existantes offrent rarement un contrôle explicite à la fois sur la structure de l'échantillonneur et sur les schémas de communication. Cela limite leur utilisation en contexte distribué, et en particulier multi-GPU. La bibliothèque développée dans cette thèse est organisée en plusieurs couches qui reflètent la structure générique des algorithmes MCMC : les noyaux de transition, la définition du modèle, l'orchestration globale de l'échantillonneur et les opérateurs distribués. Cette architecture favorise la modularité tout en donnant à l'utilisateur un contrôle précis sur les communications. En reliant directement les blocs mathématiques aux abstractions logicielles, elle combine efficacité, flexibilité et reproductibilité. Elle facilite ainsi le développement et l'évaluation d'échantillonneurs distribués, notamment pour des chercheurs sans expertise approfondie en calcul distribué.

La pertinence du cadre est évaluée à l'aide d'expériences numériques sur plusieurs problèmes inverses en imagerie, notamment l'inpainting et la déconvolution sous différents types de bruits. Un aspect important de cette évaluation concerne la comparaison des débruiteurs utilisés comme *a priori* implicites. Trois modèles ont été étudiés : DRUNet [K. Zhang et al., 2021], DnCNN [K. Zhang et al., 2017] et des réseaux légers DDFB [Repetti et al., 2022]. Les résultats mettent en évidence un compromis clair entre qualité de reconstruction et coût computationnel. DRUNet fournit les meilleures reconstructions, mais requiert beaucoup de mémoire et de temps de calcul. DnCNN offre un bon équilibre entre performance et efficacité. Les réseaux DDFB se distinguent par leur légèreté et leur stabilité dans les scénarios de très haute dimension.

Les expériences d'échantillonnage ont confirmé l'efficacité du cadre proposé. L'algorithme distribué exploite pleinement les environnements multi-GPU tout en restant numériquement équivalent à l'implémentation séquentielle. Les performances dépendent néanmoins du comportement propre à chaque débruiteur face aux paramètres fixés par PnP-ULA. Les réseaux DDFB ont montré la meilleure robustesse et la plus grande stabilité sur l'ensemble des tests. DnCNN et DRUNet présentent parfois des instabilités dans les cas les plus difficiles, comme la déconvolution avec bruit de Poisson. Ces observations soulignent l'importance d'un bon accord entre le débruiteur et la dynamique d'échantillonnage.

Enfin, les analyses de scalabilité ont démontré la capacité du cadre à exploiter efficacement les architectures distribuées. Les débruiteurs profonds présentent de bonnes propriétés de passage à l'échelle et permettent des accélérations substan-

tielles. En revanche, les *a priori* plus simples, comme la variation totale, n'offrent pas un ratio calcul/communication suffisant pour être efficaces en contexte distribué. Ces résultats montrent l'importance d'un équilibre entre coûts de calcul local et coûts de communication. Il a été possible de traiter des images couleur de taille 4096×4096, ce qui n'était pas permis par une implémentation séquentielle.

En conclusion, cette thèse présente une méthodologie complète pour la conception et l'implémentation d'algorithmes MCMC distribués exploitant des *a priori* profonds. Le cadre proposé permet le passage à l'échelle d'algorithmes existants tout en préservant leurs garanties théoriques et la qualité des estimations. Il rend également possible une quantification d'incertitude dans des contextes de très grande dimension.

Structure du manuscrit

Le manuscrit est organisé en six chapitres.

Le **Chapitre 1** introduit le contexte et les problématiques abordées dans cette thèse. Il discute les défis liés à l'inférence bayésienne en grande dimension et met en lumière les difficultés rencontrées pour intégrer des *a priori* profonds dans des algorithmes MCMC distribués. Le chapitre passe en revue les approches existantes en MCMC distribuée, en soulignant leurs avantages et leurs limites. Il présente enfin les contributions principales de cette thèse.

Le **Chapitre 2** développe une méthodologie pour la construction d'algorithmes MCMC distribués inspirés du cadre PnP, et étudie en détail les structures favorables à leur mise en œuvre, en considérant à la fois la vraisemblance et l'*a priori* appris. La méthodologie est illustrée à travers PnP-ULA, utilisant un réseau de neurones convolutif. L'algorithme distribué proposé exploite les propriétés structurelles du modèle au sein d'un cadre SPMD.

Le **Chapitre 3** présente une architecture logicielle modulaire permettant d'implémenter des algorithmes MCMC distribués, en insistant sur la nécessité de stratégies de communication efficaces et d'une bonne adaptabilité aux environnements de calcul hétérogènes, notamment multi-GPU. L'implémentation généralise la structure des algorithmes MCMC afin de faciliter l'intégration de différents échantillonneurs, structures de modèles et types d'*a priori*.

Le **Chapitre 4** valide l'approche distribuée proposée pour PnP-ULA à travers plusieurs expériences sur des problèmes d'imagerie synthétiques de grandes dimensions. Les résultats mettent en évidence des gains significatifs en termes de passage à l'échelle par rapport aux implémentations séquentielles, avec une ana-

lyse détaillée des temps de calcul et des coûts de communication pour différentes tailles de problème et architectures de débruiteurs profonds.

Le **Chapitre 5** synthétise les conclusions principales et résultats expérimentaux, en soulignant les atouts et les limites de la méthodologie distribuée proposée. Il évoque également des perspectives de recherche, notamment l'étude d'extensions asynchrones, de nouveaux schémas de communication, ainsi que la généralisation à des architectures de réseaux et à des noyaux de transition plus variés.

Contents

List of Figures xv

List of Tables xv

List of Algorithms xvi

List of Listings xvi

Notations xvii

1 Introduction 1

1.1 MCMC algorithms for inverse problems . . . 2

1.1.1 Context and motivations . . . 2

1.1.2 Bayesian inference framework . . . 3

1.1.3 Markov Chain Monte Carlo algorithms . . . 4

1.2 Learned priors for high-quality estimation . . . 5

1.2.1 From handcrafted to learned priors . . . 5

1.2.2 Using deep denoisers as learned priors . . . 9

1.3 MCMC tools for high-dimensional sampling . . . 11

1.3.1 Langevin transition kernels for scalability . . . 12

1.3.2 Model augmentation for efficient sampling . . . 13

1.3.3 Integration of learned priors within scalable samplers . . . 15

1.4 Distributed MCMC for large inverse problems . . . 16

1.4.1 High-Performance Computing fundamentals . . . 17

1.4.2 Existing distributed MCMC algorithms . . . 20

1.4.3 Towards distributed MCMC algorithms with deep priors . . . 23

1.5 Contributions . . . 24

2 A distributed Plug-and-Play MCMC algorithm 27

2.1 Problem statement . . . 28

2.1.1 Operator locality . . . 29

2.1.2 Model assumptions . . . 31
2.2 Choice of the denoiser for distributed inference . . . 33
2.3 Proposed distributed sampler . . . 38
2.4 Application to imaging inverse problems . . . 41
2.4.1 Inpainting under Gaussian noise . . . 41
2.4.2 Deconvolution under Gaussian noise . . . 42
2.4.3 Deconvolution under Poisson noise . . . 42
2.5 Conclusion . . . 44

3 From distributed algorithms to their implementation 45
3.1 Bayesian inference software development . . . 46
3.1.1 Review on Bayesian inference software . . . 46
3.1.2 Design specifications and development choices . . . 49
3.2 Library design principles . . . 52
3.2.1 High-level orchestration . . . 53
3.2.2 Model specification . . . 54
3.2.3 Transition kernels . . . 55
3.2.4 Distributed computing components . . . 55
3.3 Complete example . . . 57
3.3.1 Implementing transition kernels . . . 57
3.3.2 Implementing distributed denoisers . . . 59
3.3.3 Implementing distributed operators . . . 61
3.3.4 Main script . . . 62
3.4 Conclusion . . . 63

4 Application to imaging using distributed deep denoisers 65
4.1 Experimental setup . . . 66
4.1.1 Computational Environment and Resources . . . 66
4.1.2 Evaluation metrics . . . 67
4.2 Experimental settings . . . 68
4.2.1 Denoiser benchmarking . . . 68
4.2.2 Distributed sampler assessment . . . 69
4.3 Experimental results . . . 70
4.3.1 Denoising performance . . . 71
4.3.2 Reconstruction quality . . . 72
4.3.3 Scalability experiments . . . 73
4.4 Conclusion . . . 79

5 Conclusions and perspectives **80**
5.1 Conclusions 80
5.2 Perspectives 81

References **84**

Appendices **97**
A Running Multi-GPU scripts in Python 97
B Reproducibility and RnG Handling on GPU 98
B.1 Generating Uncorrelated Random Sequences 99
B.2 Saving and Loading Random States 99

Abstract **101**

Résumé **101**

List of Figures

1.1 DDPM illustration 8
1.2 Examples of MPI collective communication operations. 18

2.1 Illustration of a 2D linear convolution operator 30
2.2 Communications and operations underlying $\boldsymbol{W}_k$ 37

3.1 Simplified class diagram of the proposed library. 54
3.2 Illustration of the distributed convolution in DnCNN. 62

4.1 Restoration estimates 74
4.2 Runtime of denoisers applied sequentially or in a distributed manner 76

List of Tables

2.1 Notation for localized operators 32

3.1 Major Bayesian inference libraries 48

4.1 Denoisers' characteristics 67
4.2 Denoiser comparison in terms of restoration quality 71
4.3 Reconstruction quality 72
4.4 Denoiser costs 73
4.5 Strong scaling results 77
4.6 Weak scaling results 78

List of Algorithms

1 Proposed distributed sampler. 40

2 Generic MCMC Sampler. 53

List of Listings

1 Comparison example of NumPy and CuPy APIs. 51
2 Interoperability between CuPy and PyTorch. 52
3 PSGLA implementations 58
4 PnP-ULA implementation 58
5 Standard DnCNN implementation 59
6 Proposed Distributed DnCNN implementation 60
7 Simplified script to run the proposed distributed sampling algorithm. 63
8 Example of a multi-GPU Python script initialization. 98
9 GPU-specific random number generator initialization. 99

Notations

This section introduces the notations used throughout this manuscript.

- Vectors are denoted by bold lowercase letters, e.g., $\boldsymbol{x} \in \mathbb{R}^N$.
- Matrices and linear operators are denoted by bold uppercase letters, e.g., $\boldsymbol{H} \in \mathbb{R}^{M \times N}$.
- The union and disjoint union of sets are denoted by $\cup$ and $\sqcup$, respectively.
- The cardinality of a set is denoted by $\sharp$.
- The adjoint of $\boldsymbol{H} \in \mathbb{R}^{M \times N}$ is denoted by $\boldsymbol{H}^*$, and $\mathbf{1}_N \in \mathbb{R}^N$ is the unit vector with all coefficients equal to 1.
- The indicator function of a non-empty set $\mathcal{C} \subset \mathbb{R}^N$ is denoted by $\iota_{\mathcal{C}}$, with $\iota_{\mathcal{C}}(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in \mathcal{C}$, and $+\infty$ otherwise, and the notation $\mathbb{1}_{\mathcal{C}} : \boldsymbol{x} \mapsto e^{-\iota_{\mathcal{C}}(\boldsymbol{x})}$ is used.
- The proximal operator [Moreau, 1965] of $\varphi \colon \mathbb{R}^N \to (-\infty, +\infty]$ at $\boldsymbol{x} \in \mathbb{R}^N$ is defined as $\operatorname{prox}_{\varphi}(\boldsymbol{x}) = \operatorname{argmin}_{\boldsymbol{u} \in \mathbb{R}^N} \left\{ \varphi(\boldsymbol{u}) + \frac{1}{2} \|\boldsymbol{u} - \boldsymbol{x}\|_2^2 \right\}$.
- In particular, $\operatorname{prox}_{\iota_{\mathcal{C}}}$ is the orthogonal projector onto $\mathcal{C}$, written $\operatorname{proj}_{\mathcal{C}}$.

Chapter 1

Introduction

Contents

1.1 MCMC algorithms for inverse problems 2
1.1.1 Context and motivations 2
1.1.2 Bayesian inference framework 3
1.1.3 Markov Chain Monte Carlo algorithms 4
1.2 Learned priors for high-quality estimation 5
1.2.1 From handcrafted to learned priors 5
1.2.2 Using deep denoisers as learned priors 9
1.3 MCMC tools for high-dimensional sampling 11
1.3.1 Langevin transition kernels for scalability 12
1.3.2 Model augmentation for efficient sampling 13
1.3.3 Integration of learned priors within scalable samplers . 15
1.4 Distributed MCMC for large inverse problems 16
1.4.1 High-Performance Computing fundamentals 17
1.4.2 Existing distributed MCMC algorithms 20
1.4.3 Towards distributed MCMC algorithms with deep priors 23
1.5 Contributions 24

This chapter provides the methodological and computational background supporting the work presented in this thesis. Section 1.1 focuses on inverse problems formulated within a Bayesian framework, emphasizing the role of Markov Chain Monte Carlo (MCMC) algorithms to compute estimates and quantify associated uncertainties. Section 1.2 reviews learned priors based on neural networks, and

discusses their role in so-called Plug-and-Play (PnP) approaches for high-quality estimation. Section 1.3 reviews tools for high-dimensional sampling, including scalable transition kernels and (approximate) model augmentation techniques. Section 1.4 outlines scalability challenges associated with high-dimensional problems. High performance computing are discussed to address these issues, with a review of distributed MCMC algorithms already proposed in the literature. Finally, Section 1.5 summarizes the contributions of this thesis.

1.1 MCMC algorithms for inverse problems

Bayesian inference offers a powerful framework to address inverse problems by combining observational data with prior information about the unknown quantities of interest. It has become increasingly relevant in modern imaging and signal processing applications where uncertainty quantification is crucial. This section reviews the foundational elements of Bayesian inverse problems and their inherent challenges. The Bayesian framework is then introduced, encompassing posterior distributions, point estimators, and uncertainty quantification. Finally, MCMC algorithms are presented as a general approach to perform Bayesian inference in high-dimensional settings, highlighting the advantages and scalability challenges that motivate this work.

1.1.1 Context and motivations

Inverse problems consist in recovering an unknown signal or parameter vector $\overline{\boldsymbol{x}} \in \mathbb{R}^N$ from indirect and degraded observations $\boldsymbol{y} \in \mathbb{R}^M$. A common formulation relies on a measurement model

$$\boldsymbol{y} = \mathcal{A}(\overline{\boldsymbol{x}}), \tag{1.1}$$

where the operator $\mathcal{A} : \mathbb{R}^N \to \mathbb{R}^M$ accounts for the acquisition process as well as perturbations due to noise or model inaccuracies. Inverse problems are often *ill-posed* in the sense of Hadamard, 1902, e.g., solutions may not exist or may not be unique. Ill-posedness makes direct inversion unstable or even infeasible, and motivates the introduction of regularization strategies.

The dimensionality of modern inverse problems can be extremely large, with both N and M larger than 10^6 in typical imaging applications [Pereyra et al., 2016]. Such dimensions are encountered in a wide range of fields, including medical imaging (e.g., magnetic resonance imaging [Luo et al., 2023]) and astronomy

(e.g., radio-interferometric reconstruction [X. Cai et al., 2018]). These challenges motivate the use of principled statistical approaches that can incorporate prior knowledge, quantify uncertainties, and scale to high-dimensional settings.

1.1.2 Bayesian inference framework

The Bayesian paradigm [Robert, 2007] provides a natural formalism to address inverse problems by combining the information carried by the observations with prior knowledge about the unknown model parameters. The likelihood function $\pi(\boldsymbol{y}|\boldsymbol{x})$ models the probability of observing $\boldsymbol{y}$ given a candidate $\boldsymbol{x}$, while the prior distribution $\pi(\boldsymbol{x})$ encodes *a priori* knowledge about $\boldsymbol{x}$, such as smoothness, sparsity, or consistency with learned data-driven models. Bayes' theorem combines these two ingredients into the posterior distribution

$$\pi(\boldsymbol{x}|\boldsymbol{y}) \propto \pi(\boldsymbol{y}|\boldsymbol{x})\pi(\boldsymbol{x}). \tag{1.2}$$

The posterior distribution serves as the fundamental object of inference, from which point estimates and uncertainty measures can be derived. In the remainder of this work, problems are described through posterior distributions of the form

$$\pi(\boldsymbol{x}|\boldsymbol{y}) \propto p(\boldsymbol{x}) \exp\big(-f_1(\boldsymbol{H}_1\boldsymbol{x}) - f_2(\boldsymbol{H}_2\boldsymbol{x})\big), \tag{1.3}$$

where $f_1 \colon \mathbb{R}^{M_1} \to (-\infty, +\infty]$ is a smooth function, $f_2 \colon \mathbb{R}^{M_2} \to (-\infty, +\infty]$ a convex but possibly non-smooth function, and $\boldsymbol{H}_1 \colon \mathbb{R}^N \to \mathbb{R}^{M_1}, \boldsymbol{H}_2 \colon \mathbb{R}^N \to \mathbb{R}^{M_2}$ are linear operators. The function $p : \mathbb{R}^N \to \mathbb{R}$ may represent a component of the prior distribution, which can be encoded either by classical models or by modern machine learning approaches such as deep generative networks [Faye et al., 2025; Holden et al., 2022; Laumont et al., 2022]. This formulation encompasses a wide range of likelihoods and priors used in imaging and signal processing [Ongie et al., 2020; Pereyra et al., 2016; Wen et al., 2023a,b].

Several Bayesian estimators are commonly considered [Robert, 2007, Chapter 4]. Among standard estimators, the maximum a posteriori (MAP) estimator relates to the posterior mode,

$$\hat{\boldsymbol{x}}_{\mathrm{MAP}} = \operatorname*{argmax}_{\boldsymbol{x}\in\mathbb{R}^N} \pi(\boldsymbol{x}|\boldsymbol{y}), \tag{1.4}$$

providing a regularized reconstruction that balances data fidelity with prior information. The minimum mean square error (MMSE) estimator corresponds to

the posterior mean and can be expressed as the integral

$$\hat{\boldsymbol{x}}_{\text{MMSE}} = \mathbb{E}_\pi[\boldsymbol{x}|\boldsymbol{y}] = \int \widetilde{\boldsymbol{x}}\, \pi(\widetilde{\boldsymbol{x}}|\boldsymbol{y})\, d\widetilde{\boldsymbol{x}}, \tag{1.5}$$

which minimizes the expected quadratic error. Other estimators, such as the posterior median, can be obtained from different loss functions and similarly require integration over the posterior distribution.

Beyond point estimates, the posterior distribution enables the computation of credibility intervals (CIs) and other uncertainty quantification tools [J. Zhang et al., 2020][Robert, 2007, Chapter 5], which are essential in applications where decisions depend on the reliability of the reconstruction and no validation data is available. For instance, credibility intervals at level $1-\alpha$ with $\alpha \in (0,1)$ correspond to the interval $\mathcal{C}_\alpha$ such that $\mathbb{P}_\pi(\boldsymbol{x} \in \mathcal{C}_\alpha|\boldsymbol{y}) = 1-\alpha$, which amounts to finding a set $\mathcal{C}_\alpha$ satisfying

$$\int_{\mathcal{C}_\alpha} \pi(\boldsymbol{x}|\boldsymbol{y})\, d\boldsymbol{x} = 1-\alpha. \tag{1.6}$$

In imaging applications, computing exact credible regions for the full vector $\boldsymbol{x}$ is generally intractable due to the high dimensionality and the complexity of the posterior distribution. Instead, practitioners often resort to pixelwise credible intervals, obtained by estimating quantiles of the marginal posterior distributions.

1.1.3 Markov Chain Monte Carlo algorithms

A distinctive advantage of Bayesian inference lies in its ability to quantify uncertainty through integrals involving the probability density function (p.d.f.) of the posterior distribution. Point estimators, such as the MMSE, and uncertainty measures, such as credibility intervals, both require evaluating expectations or quantiles with respect to $\pi(\boldsymbol{x}|\boldsymbol{y})$. In high-dimensional inverse problems, however, such integrals are generally analytically intractable without conjugate models, and numerically challenging due to the curse of dimensionality.

Monte Carlo methods [Robert et al., 2004] provide a principled approximation by replacing integrals with empirical averages. If independent samples $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(S)}$ can be drawn independently from $\pi(\boldsymbol{x}|\boldsymbol{y})$, any posterior expectation $\mathbb{E}_\pi[g(\boldsymbol{x})|\boldsymbol{y}]$ of a test function $g\colon \mathbb{R}^N \to \mathbb{R}$ may be approximated as

$$\mathbb{E}_\pi[g(\boldsymbol{x})|\boldsymbol{y}] \approx \frac{1}{S}\sum_{s=1}^{S} g(\boldsymbol{x}^{(s)}), \tag{1.7}$$

with convergence guaranteed by the law of large numbers. This formulation extends naturally to quantile estimation and other posterior summaries. However, generating independent samples from complex, high-dimensional posteriors is generally challenging due to the unknown normalizing constant and the potential absence of efficient direct sampling schemes.

Markov Chain Monte Carlo (MCMC) algorithms [Meyn et al., 2009] overcome this limitation by generating correlated samples from a Markov chain whose stationary distribution is $\pi(\boldsymbol{x}|\boldsymbol{y})$. Under mild conditions, the chain converges to the target distribution, enabling Monte Carlo approximations after a suitable burn-in period. Classical algorithms include the Metropolis-Hastings (MH) [Hastings, 1970] method, which accepts or rejects proposals according to a balanced ratio, and Gibbs sampling [Gelfand et al., 1990], which updates components sequentially from their full conditional distributions. Both approaches only require evaluating the posterior p.d.f. up to a normalizing constant, making them well suited to complex models.

The efficiency and accuracy of MCMC sampling critically depend on the choice of prior. A well-specified prior can guide the sampling toward regions close to the ground truth $\overline{\boldsymbol{x}}$, leading to more plausible and accurate estimates. However, the complexity of the prior can also make exploration more difficult, potentially resulting in slow mixing. This highlights that prior modeling is a central element in achieving efficient and high-quality inference.

1.2 Learned priors for high-quality estimation

While classical MCMC algorithms rely on generic or handcrafted priors, their ability to produce accurate estimates is limited by the expressiveness of the prior. Richer, data-driven priors can encode more detailed structural knowledge and complex dependencies, enabling inference to better capture the underlying posterior distribution. This section explores the evolution from traditional handcrafted regularizers to modern neural network-based priors, and their integration into existing algorithms.

1.2.1 From handcrafted to learned priors

Handcrafted priors such as total variation (TV) [Rudin et al., 1992], Tikhonov regularization [Benning et al., 2018], and sparsity-based approaches [Carvalho et al., 2010; Mitchell et al., 1988; Pustelnik et al., 2016] have traditionally formed the

backbone of regularized inverse problems. For instance in imaging applications, the TV prior admits a probability density of the form

$$\big(\forall \boldsymbol{x} \in \mathbb{R}^N\big) \quad \pi(\boldsymbol{x}) \propto \exp\big(-\beta \|\boldsymbol{D}(\boldsymbol{x})\|_{2,1}\big), \tag{1.8}$$

with $\beta > 0$ a regularization parameter, $\boldsymbol{D} : \mathbb{R}^N \to \mathbb{R}^{N\times 2}$ the discrete 2D gradient operator, and

$$\big(\forall \boldsymbol{z} = (\boldsymbol{z}_n)_{1\leq n\leq N} \in \mathbb{R}^{N\times 2}\big) \quad \|\boldsymbol{z}\|_{2,1} = \sum_{n=1}^{N} \|\boldsymbol{z}_n\|_2, \tag{1.9}$$

which promotes piecewise-constant reconstructions by penalizing the sum of gradient magnitudes. More generally, sparsity-promoting priors can be defined in a transformed domain, such as wavelets bases [Mallat, 2009], where the signal is assumed to have only a few significant coefficients. Similarly, Tikhonov regularization enforces smoothness. These traditional regularizers can lead to optimization problems with well-established theoretical guarantees and computationally efficient algorithms [Combettes et al., 2011].

However, such handcrafted priors face significant limitations when dealing with complex large-scale problems, such as high-resolution image restoration. Natural images exhibit intricate textures, edges, and semantic structures that cannot be properly captured by simple mathematical models. The analytic simplicity of traditional regularizers, while computationally advantageous, often results in over-smoothed reconstructions that lack fine details and realistic appearance. In imaging applications, traditional handcrafted priors such as the TV norm or other sparsity-based regularizers may fail to capture the intricate structures present in high-resolution images.

Learned priors encoded by neural networks have emerged as powerful tools to incorporate data-driven knowledge [Ongie et al., 2020]. Neural networks trained on large datasets can model this complexity more effectively, and generally lead to more accurate estimates. Several families of neural network priors have been proposed for image restoration and inverse problems, each with distinct modeling assumptions and computational capabilities for large-scale problems.

Convolutional neural networks as denoisers

Convolutional Neural Networks (CNNs) have become a standard for modern image denoising, owing to their ability to learn complex mappings from noisy obser-

vations to clean images [Alzubaidi et al., 2021; Fleuret, 2024]. By applying convolutional filters in a hierarchical manner, CNNs extract spatial features ranging from edges and textures to higher-level patterns, which makes them particularly well suited for image restoration tasks.

Formally, a denoising CNN is a feedforward mapping $D_\epsilon : \mathbb{R}^N \to \mathbb{R}^N$, that estimates an unknown clean image $\overline{\boldsymbol{x}}$ from noisy data $\boldsymbol{y} = \overline{\boldsymbol{x}} + \epsilon \boldsymbol{w}$, where $\boldsymbol{w}$ is a realization of a standard multivariate Gaussian distribution and $\epsilon > 0$ the noise standard deviation. The network is composed of K successive layers, each layer $k \in \{1, \dots, K\}$ performing the transformation

$$T_k : \boldsymbol{v} \mapsto \eta_k\big(\boldsymbol{W}_k \boldsymbol{v} + \boldsymbol{b}_k\big), \tag{1.10}$$

where $\boldsymbol{W}_k$ is a convolution operator, $\boldsymbol{b}_k$ a bias term, and η_k a non-linear activation function. The overall denoiser is obtained by composing such layers.

Modern CNN architectures, such as U-Nets [Ronneberger et al., 2015] and residual networks [He et al., 2016], extend this principle by incorporating multi-scale feature extraction and skip connections. These refinements enhance the network's ability to preserve fine details while removing noise. In practice, CNN denoisers act as implicit prior models of natural images [Ulyanov et al., 2020], making them particularly attractive for integration into Bayesian frameworks.

Explicit density modeling: normalizing flows and diffusion models

Normalizing Flows (NF) [Kobyzev et al., 2021] are probabilistic models that transform a simple, tractable distribution into a more complex target distribution through a sequence of invertible and differentiable mappings. Formally, a normalizing flow learns a transformation $f_\theta : \mathbb{R}^N \to \mathbb{R}^N$ such that if $\boldsymbol{z}$ is drawn from a simple base distribution π_0, typically a standard multivariate Gaussian, then $\boldsymbol{x} = f_\theta(\boldsymbol{z})$ is approximately distributed according to the target law [Coeurdoux et al., 2024a]. The density is given by the change of variables formula

$$\pi(\boldsymbol{x}) = \pi_0(f_\theta^{-1}(\boldsymbol{x})) \left| \det \frac{\partial f_\theta^{-1}}{\partial \boldsymbol{x}}(\boldsymbol{x}) \right|. \tag{1.11}$$

This enables explicit density evaluation and exact sampling, making normalizing flows suitable for explicit prior modeling in inverse problems [Z. Cai et al., 2024].

Denoising Diffusion Probabilistic Models (DDPM) [Daras et al., 2024; Kawar et al., 2022; Sohl-Dickstein et al., 2015] follow a different paradigm, learning to generate data by reversing a gradual noising process, as illustrated in Figure 1.1.

The forward process progressively corrupts data through Gaussian transitions

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t\mathbf{I}) \tag{1.12}$$

while the reverse process is modeled as

$$p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}; \mu_\theta(\boldsymbol{x}_t, t), \Sigma_\theta(\boldsymbol{x}_t, t)), \tag{1.13}$$

where the mean μ_θ and covariance Σ_θ are parameterized by neural networks. By learning these reverse transitions, DDPMs approximate the distribution of natural images and serve as expressive prior models [Coeurdoux et al., 2024b; Mbakam et al., 2024; Wu et al., 2024; Xu et al., 2024; Zheng et al., 2025]. Note that the reverse process can be interpreted using iterative noising-denoising steps with additive Gaussian noise [Leclaire et al., 2025].

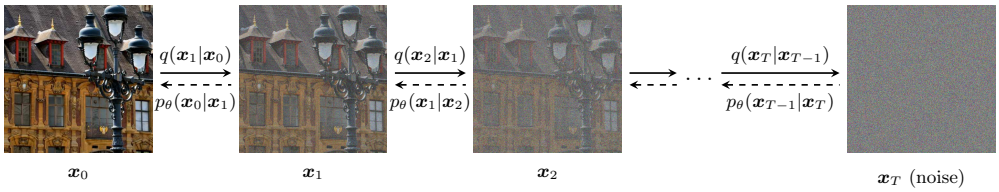

Figure 1.1: Denoising Diffusion Probabilistic Models principle illustration. The forward process (top arrows) gradually adds noise to an image, while the reverse process (bottom arrows) denoises it step by step.

Despite their expressivity, NF and DDPM models generally require a large number of parameters and dense operations (e.g., attention layers), which can hinder scalability on large datasets or high-resolution images.

Algorithm unrolling and interpretable networks

Algorithm unrolling represents a design methodology that bridges optimization theory and deep learning by deriving network architectures from iterative optimization algorithms [Monga et al., 2021]. These networks do not form a class of deep priors strictly speaking, but rather a structured approach to design networks, where each layer corresponds to an iteration of a specific optimization algorithm.

Consider an iterative algorithm that produces updates of the form $\boldsymbol{x}^{(k+1)} = T(\boldsymbol{x}^{(k)})$ for some operator $T : \mathbb{R}^N \to \mathbb{R}^N$. Algorithm unrolling transforms this into a neural network where layer k implements a learnable version

$$\boldsymbol{x}^{(k+1)} = T_{\boldsymbol{\theta}_k}(\boldsymbol{x}^{(k)}), \tag{1.14}$$

where $\boldsymbol{\theta}_k$ represents learned parameters specific to layer k.

The resulting networks inherit the structure of the original iterative algorithm while allowing learned parameters to adapt to specific problem instances. For example, Deep Dual Forward-Backward (DDFB) networks [Le et al., 2022, 2024; Repetti et al., 2022] apply this approach to dual forward-backward algorithms.

In practice, unfolded algorithms offer strong performance using a limited number of parameters [Le et al., 2024; Monga et al., 2021]. They often rely on term-wise operations and convolutions, making them suitable for large-scale problems while still benefiting from the interpretability and theoretical foundations of optimization algorithms. Beyond algorithm unrolling, other CNN architectures, e.g. Input-Convex Neural Networks (ICNNs) [Gagneux et al., 2025] and the Convex Ridge Regularizer (CRR) [Goujon et al., 2023], have been proposed to preserve convex structures, which enhance interpretability and facilitate inference by providing stronger theoretical guarantees.

1.2.2 Using deep denoisers as learned priors

Plug-and-Play (PnP) approaches provide a general paradigm for incorporating learned priors into optimization and sampling algorithms. These methods rely on the use of a denoiser, often implemented as a neural network, to implicitly encode complex prior distributions. These approaches are derived from the MAP and MMSE estimators for the following Gaussian denoising task.

In this problem, the goal is to recover an estimate of $\overline{\boldsymbol{x}}$ from $\boldsymbol{y} = \overline{\boldsymbol{x}} + \epsilon \boldsymbol{w}$, where $\boldsymbol{w}$ is a realization of a standard multivariate Gaussian distribution and $\epsilon > 0$ the noise standard deviation. Combined with a prior distribution $\pi(\boldsymbol{x})$, the likelihood of this model leads to the posterior distribution

$$\pi(\boldsymbol{x}|\boldsymbol{y}) \propto \exp\left(-\tfrac{1}{2\epsilon^2}\|\boldsymbol{y}-\boldsymbol{x}\|_2^2\right)\pi(\boldsymbol{x}). \tag{1.15}$$

Plug-and-Play framework

The Plug-and-Play framework [Kamilov et al., 2023; Venkatakrishnan et al., 2013; K. Zhang et al., 2021] is derived from the MAP estimator for (1.15), given by

$$\hat{\boldsymbol{x}}_{\text{MAP}} = \operatorname*{argmin}_{\boldsymbol{x}\in\mathbb{R}^n} \left\{ \tfrac{1}{2\epsilon^2}\|\boldsymbol{y}-\boldsymbol{x}\|_2^2 - \log \pi(\boldsymbol{x}) \right\}. \tag{1.16}$$

This formulation highlights that MAP estimation is equivalent to a regularized optimization problem, where $-\log\pi(\boldsymbol{x})$ acts as a regularization function. The PnP approach exploits this connection through the *proximal operator* [Beck,

2017; Moreau, 1965], a fundamental object in optimization. For any proper lower semicontinuous function $\varphi : \mathbb{R}^n \to (-\infty, +\infty]$, the proximal operator reads

$$(\forall \boldsymbol{x} \in \mathbb{R}^n) \quad \operatorname{prox}_{\varphi}(\boldsymbol{y}) = \underset{\boldsymbol{x} \in \mathbb{R}^n}{\operatorname{argmin}} \left\{ \tfrac{1}{2} \|\boldsymbol{u} - \boldsymbol{x}\|_2^2 + \varphi(\boldsymbol{u}) \right\}. \tag{1.17}$$

For the regularization function $\varphi(\boldsymbol{x}) = -\epsilon^2 \log \pi(\boldsymbol{x})$, the proximal operator corresponds precisely to the MAP estimator (1.16). This observation justifies replacing a proximal operator with a denoiser D_ϵ trained for noise level ϵ:

$$D_\epsilon(\boldsymbol{x}) \approx \operatorname{prox}_{-\epsilon^2 \log \pi}(\boldsymbol{x}). \tag{1.18}$$

The PnP framework thus enables the integration of sophisticated learned denoisers into algorithms that rely on proximal operators without requiring explicit knowledge of the underlying prior distribution $\pi(\boldsymbol{x})$.

Score-based framework

Instead of proximal operator replacements, the score-based framework enables the substitution of gradient-based information with denoiser residuals. The RED paradigm [Romano et al., 2017] explores related ideas from a different perspective.

Assuming the existence of an optimal prior distribution for $\boldsymbol{x}$, denoted $p^\star(\boldsymbol{x})$, Laumont et al. (2022) established a relation between a learned Gaussian denoiser D_ϵ and the gradient of the log-prior, $\nabla_{\boldsymbol{x}} \log \pi(\boldsymbol{x})$. Their derivation introduces a smoothed version of the true marginal distribution, denoted $p^\star_{\epsilon^2}(\boldsymbol{x})$, obtained by convolving $p^\star(\boldsymbol{x})$ with a Gaussian kernel of variance ϵ^2. The gradient of the logarithm of this smoothed distribution can then be expressed through an oracle denoiser $D^\star_\epsilon$, defined as the optimal MMSE estimator for the Gaussian denoising problem (1.15). This connection builds upon Tweedie's identity [Efron, 2011]:

$$\frac{D^\star_\epsilon(\boldsymbol{x}) - \boldsymbol{x}}{\epsilon^2} = \nabla_{\boldsymbol{x}} \log p^\star_{\epsilon^2}(\boldsymbol{x}). \tag{1.19}$$

When the learned denoiser D_ϵ provides a sufficiently accurate approximation of the oracle denoiser $D^\star_\epsilon$, the following practical approximation holds

$$\frac{D_\epsilon(\boldsymbol{x}) - \boldsymbol{x}}{\epsilon^2} \approx \nabla_{\boldsymbol{x}} \log \pi(\boldsymbol{x}). \tag{1.20}$$

This relation makes it possible to define an implicit regularization functional whose gradient corresponds to the denoiser residual.

Practical considerations

Although PnP and score-based approaches differ in their theoretical foundations, their goals are closely related. PnP replaces proximal operators by denoisers, while the latter express the score of the prior through the denoiser residual.

In both cases, the central principle is the same: denoisers serve as implicit representations of complex prior distributions. This equivalence means that the choice between PnP and score-based formulations often becomes a matter of practical algorithmic convenience rather than fundamental differences.

In practice, both frameworks encounter similar theoretical challenges. Formal guarantees typically rely on assumptions such as Lipschitz continuity, local homogeneity or differentiability of the denoiser [Faye et al., 2025; Laumont et al., 2022; Renaud et al., 2025]. Verifying these conditions can be challenging in practice, especially when neural networks are employed without architectural constraints or explicit regularization during training. Nevertheless, empirical evidence consistently demonstrates that denoiser-based priors significantly outperform traditional handcrafted regularizers in complex problems.

Integrating these denoiser-based priors into sampling algorithms requires techniques that can effectively leverage the implicit nature of these priors. As detailed in the following section, Langevin-based sampling methods provide a natural framework for such integration, especially in high-dimensional settings where posterior exploration can be difficult.

1.3 MCMC tools for high-dimensional sampling

Sampling from complex posterior distributions in high-dimensional settings while exploiting learned priors can be challenging. To address this, MCMC techniques leveraging gradient information and model augmentation strategies have been developed. This section briefly reviews key ingredients for scalable sampling in high-dimensional settings. It focuses on Langevin-based transition kernels and data augmentation methods, which decompose complex posteriors into more tractable elementary transition mechanisms.

The different ingredients presented in this section are typically adapted to probability distributions with density of the form (1.3) with $p(\boldsymbol{x}) \propto \exp\big(-f_0(\boldsymbol{x})\big)$, leading to

$$\pi(\boldsymbol{x}|\boldsymbol{y}) \propto \exp\big(-f_1(\boldsymbol{H}_1\boldsymbol{x}) - f_2(\boldsymbol{H}_2\boldsymbol{x}) - f_0(\boldsymbol{x})\big), \tag{1.21}$$

where $f_0 \colon \mathbb{R}^N \to (-\infty, +\infty]$ is a proper, l.s.c. and convex function.

1.3.1 Langevin transition kernels for scalability

Several MCMC algorithms derived from Langevin transition kernels have proved efficient to sample high-dimensional probability distributions [Durmus et al., 2017, 2018; Pereyra, 2016], with opportunities to accommodate a distributed implementation [Thouvenin et al., 2024], as detailed in Section 1.4. They usually stem from the Euler-Maruyama discretization of the Langevin dynamic equation, using gradient information to guide the sampling process [Roberts et al., 1996]. Such kernels essentially require sampling from a multivariate standard Gaussian distribution (a computationally inexpensive task), and the evaluation of a gradient (generally more costly). Beyond these first-order methods, Hamiltonian Monte Carlo (HMC) [Neal, 2011] introduces auxiliary momentum variables and simulates Hamiltonian dynamics, allowing for long-distance moves in parameter space and more efficient exploration of complex posteriors. The main family of Langevin-based sampling methods are presented below, from the simplest to the most general cases. In the following, the parameter $L > 0$ denotes a Lipschitz constant of the gradient of $f_1 \circ \boldsymbol{H}_1$.

Unadjusted Langevin Algorithm

In the simple case where $-\log(\pi(\boldsymbol{x}|\boldsymbol{y})) \overset{c}{=} f_1(\boldsymbol{H}_1\boldsymbol{x})$[1], an Unadjusted Langevin Algorithm (ULA) [Durmus et al., 2017] can be used. It is defined following

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \gamma \boldsymbol{H}_1^* \nabla f_1(\boldsymbol{H}_1 \boldsymbol{x}^{(t)}) + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)}, \quad \boldsymbol{\xi}^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_N), \tag{1.22}$$

with $t \in \mathbb{N}^*$ the algorithm iteration index, $\boldsymbol{H}_1^*$ the adjoint of $\boldsymbol{H}_1$ and $\gamma \in (0, L^{-1})$ the selected step-size. Acceptance-rejection steps can be added to ULA to correct the bias due to the discretization, leading to the Metropolis-Adjusted Langevin Algorithm (MALA) [Roberts et al., 1996].

Proximal variants

Proximal operators [Moreau, 1965] play a central role in optimization methods involving both smooth and non-smooth terms. Closed-form expressions exist for many functions commonly used in applications [Combettes et al., 2011], including the ℓ_1-norm and the Kullback-Leibler divergence. This makes proximal operators a natural tool for extending Langevin-based transition kernels to settings involving

[1] The notation $\overset{c}{=}$ stands for equality up to a constant term.

non-differentiable components [Durmus et al., 2018; Pereyra, 2016; Renaud et al., 2025; Salim et al., 2019].

In the setting of (1.21), if $\boldsymbol{H}_2 = \mathbf{I}_N$, $f_0 = \mathbf{0}$, and f_2 is non-smooth, the negative log-likelihood reduces to $f_1(\boldsymbol{H}_1\boldsymbol{x})+f_2(\boldsymbol{x})$. In such cases, proximal-based Langevin schemes can incorporate the proximal operator of f_2. In particular, the Moreau-Yosida Unadjusted Langevin Algorithm (MYULA) [Durmus et al., 2018] update step can be expressed as

$$\begin{aligned} &\boldsymbol{\xi}^{(t+1)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_N), \\ &\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \gamma \boldsymbol{H}_1^* \nabla f_1(\boldsymbol{H}_1 \boldsymbol{x}^{(t)}) + \tfrac{\gamma}{\lambda}\Big(\operatorname{prox}_{\lambda f_2}\left(\boldsymbol{x}^{(t)}\right) - \boldsymbol{x}^{(t)}\Big) + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)}, \end{aligned} \tag{1.23}$$

with a smoothing parameter $\lambda > 0$ and stepsize $\gamma \in \big(0, \lambda(L\lambda+1)^{-1}\big]$. Another example is the Proximal Stochastic Gradient Langevin Algorithm (PSGLA) [Renaud et al., 2025; Salim et al., 2019], whose update step writes

$$\begin{aligned} &\boldsymbol{\xi}^{(t+1)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_N), \\ &\boldsymbol{x}^{(t+1)} = \operatorname{prox}_{\gamma f_2}\big(\boldsymbol{x}^{(t)} - \gamma \boldsymbol{H}_1^* \nabla f_1(\boldsymbol{H}_1 \boldsymbol{x}^{(t)}) + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)}\big), \end{aligned} \tag{1.24}$$

with $\gamma \in (0, L^{-1})$.

Extensions and other variants

The algorithms presented above represent the core framework for gradient-based MCMC sampling. However, numerous extensions and variants have been developed to address specific computational challenges and improve sampling efficiency. These include preconditioned variants that incorporate metric tensors to adapt to local geometry [Girolami et al., 2011], position-dependent diffusion schemes that adjust noise levels based on local curvature [Xifara et al., 2014], stochastic gradient approaches for large-scale data settings [Welling et al., 2011], and higher-order discretization schemes that require fewer evaluations of the gradient [Pereyra et al., 2023].

1.3.2 Model augmentation for efficient sampling

In the general case (1.21), the proximal operator of the function $f_2 \circ \boldsymbol{H}_2 + f_0$ is rarely known in closed-form, and often hard to compute. To cope with such situations, splitting techniques reminiscent of the optimization literature can be used to separately handle the linear operator $\boldsymbol{H}_2$ and the non-smooth functions

f_2 and f_0 [Rendell et al., 2020; Vono et al., 2021]. These approaches rely on data augmentation strategies that introduce auxiliary variables to decompose complex posterior distributions into more tractable components.

Exact Data Augmentation

Techniques of eXact Data Augmentation (XDA) [van Dyk et al., 2001] can sometimes provide analytically tractable solutions. When the posterior distribution admits exact augmentation schemes, auxiliary variables can be introduced such that the joint distribution $\widetilde{\pi}(\boldsymbol{x}, \boldsymbol{z})$ over the original parameters $\boldsymbol{x}$ and auxiliary variables $\boldsymbol{z}$ satisfies

$$\pi(\boldsymbol{x}) = \int \widetilde{\pi}(\boldsymbol{x}, \boldsymbol{z})\, d\boldsymbol{z}, \tag{1.25}$$

so that the target posterior $\pi(\boldsymbol{x})$ is recovered as a marginal without approximation. Classical examples, such as the Bayesian Lasso [Park et al., 2008] or the Bayesian logistic regression using [Polson et al., 2013], introduce latent variables in hierarchical models, which factorize in a way that enables efficient Gibbs sampling.

The key advantage of exact data augmentation is the preservation of the target distribution: samples from the augmented model correspond exactly to samples from the original posterior after marginalization. However, exact augmentation schemes are only available for specific model structures and prior specifications, limiting their applicability to a limited number of inverse problems.

Asymptotically Exact Data Augmentation

When exact augmentation schemes are not available for the problem structure at hand, approximate methods can provide a practical alternative. The Asymptotically eXact Data Augmentation (AXDA) [Vono et al., 2021] proposes to approximate (1.21) by

$$\pi_\rho(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{y}) \propto \exp\left(-f_1(\boldsymbol{H}_1\boldsymbol{x}) - f_2(\boldsymbol{z}) - f_0(\boldsymbol{x}) - \frac{1}{2\rho}\|\boldsymbol{z} - \boldsymbol{H}_2\boldsymbol{x}\|_2^2\right), \tag{1.26}$$

where $\boldsymbol{z} \in \mathbb{R}^{M_2}$ is an auxiliary variable, $\rho > 0$ is an hyperparameter controlling the coupling strength between $\boldsymbol{x}$ and $\boldsymbol{z}$[2], hence the approximation with respect to the original target distribution π.

[2]The coupling between $\boldsymbol{x}$ and $\boldsymbol{z}$ can be more generally enforced with a Bregman divergence instead of the Euclidean distance [Vono et al., 2021].

Using this framework, Vono et al., 2019 proposed to sample (1.26) with a Gibbs sampler, referred to as Split Gibbs Sampling (SGS), based on the corresponding marginal distributions

$$\pi_\rho(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) \propto \exp\left(-f_1(\boldsymbol{H}_1\boldsymbol{x}) - f_0(\boldsymbol{x}) - \frac{1}{2\rho}\|\boldsymbol{z}-\boldsymbol{H}_2\boldsymbol{x}\|_2^2\right), \tag{1.27}$$

$$\pi_\rho(\boldsymbol{z}|\boldsymbol{y},\boldsymbol{x}) \propto \exp\left(-f_2(\boldsymbol{z}) - \frac{1}{2\rho}\|\boldsymbol{z}-\boldsymbol{H}_2\boldsymbol{x}\|_2^2\right). \tag{1.28}$$

In practice, (1.27) and (1.28) can be significantly less expensive to sample from compared to (1.26), while offering opportunities for parallel computing when multiple auxiliary variables are used [Rendell et al., 2020; Vono et al., 2019, 2021]. In [Vono et al., 2022], sampling from the marginal distributions (1.27), (1.28) is compared to direct sampling from the original posterior (1.21). The authors also establish theoretical convergence guarantees for SGS in this setting.

For instance, using a MYULA transition for (1.27) and a PSGLA transition for (1.28) leads to

$$\begin{aligned}
\boldsymbol{\xi}^{(t+1)} &\sim \mathcal{N}(\boldsymbol{0},\mathbf{I}_N),\\
\boldsymbol{x}^{(t+1)} &= \boldsymbol{x}^{(t)} - \gamma\big(\boldsymbol{H}_1^*\nabla f_1(\boldsymbol{H}_1\boldsymbol{x}^{(t)}) + \boldsymbol{H}_2^*(\boldsymbol{H}_2\boldsymbol{x}^{(t)} - \boldsymbol{z}^{(t)})\big)\\
&\quad + \frac{\gamma}{\lambda}\Big(\operatorname{prox}_{\lambda f_0}\left(\boldsymbol{x}^{(t)}\right) - \boldsymbol{x}^{(t)}\Big) + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)},\\
\boldsymbol{\zeta}^{(t+1)} &\sim \mathcal{N}(\boldsymbol{0},\mathbf{I}_P),\\
\boldsymbol{z}^{(t+1)} &= \operatorname{prox}_{\kappa f_2}\left(\boldsymbol{z}^{(t)} - \frac{\kappa}{\rho}\left(\boldsymbol{z}^{(t)} - \boldsymbol{H}_2\boldsymbol{x}^{(t+1)}\right) + \sqrt{2\kappa}\boldsymbol{\zeta}^{(t+1)}\right),
\end{aligned} \tag{1.29}$$

with $\lambda > 0$, $\gamma \in \left(0, \lambda\big((L+\|\boldsymbol{H}_2\|_2^2)\lambda+1\big)^{-1}\right]$ and $\kappa \in (0,\rho)$. Note that the splitting parameter ρ can be tuned by marginal maximum likelihood estimation, as proposed in [Pereyra et al., 2023].

1.3.3 Integration of learned priors within scalable samplers

The algorithms presented above rely on the availability of closed-form proximal operators for the functions f_2 and f_0, which can typically correspond to hand-crafted regularizers or priors. While such classical regularizers have proven effective for many applications, they often fail to capture the complex structures present in high-resolution natural images and other high-dimensional data. This limitation motivates the integration of the learned priors introduced in Section 1.2 into the Langevin-based sampling frameworks.

The denoiser-based approaches presented in Section 1.2.2 provide a natural bridge between learned priors and Langevin sampling algorithms. For example, PnP-PSGLA [Renaud et al., 2025] can be seen as a PnP variant of PSGLA (1.24), defined as

$$\boldsymbol{x}^{(t+1)} = D_{\sqrt{\gamma}}\left(\boldsymbol{x}^{(t)} - \gamma \boldsymbol{H}_1^* \nabla f_1(\boldsymbol{H}_1 \boldsymbol{x}^{(t)}) + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)}\right), \tag{1.30}$$

with $\boldsymbol{\xi}^{(t+1)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_N)$.

Similarly, the PnP Unadjusted Langevin Algorithm (PnP-ULA) [Laumont et al., 2022] can be interpreted within the score-based framework. It assumes that $\mathbf{I}_N - D_\epsilon$ is Lipschitz continuous with constant $L_D > 0$ and can be seen as a score-based variant of ULA (1.22) [Faye et al., 2025]. The iteration reads

$$\begin{aligned}\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \gamma \boldsymbol{H}_1^* \nabla f_1(\boldsymbol{H}_1 \boldsymbol{x}^{(t)}) + \frac{\alpha\gamma}{\epsilon}\big(D_\epsilon(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big) \\ + \frac{\gamma}{\lambda}\big(\Pi_{\mathcal{C}}(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big) + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)},\end{aligned} \tag{1.31}$$

where $\mathcal{C} \subset \mathbb{R}^N$ is a non-empty convex and compact set containing most of the prior probability mass, and $\Pi_{\mathcal{C}}$ denotes the orthogonal projector onto $\mathcal{C}$. In imaging problems, $\mathcal{C}$ is often chosen as a Cartesian product, so that $\Pi_{\mathcal{C}}$ can be computed efficiently [Laumont et al., 2022]. It can be set such that it contains more than the theoretical pixel value range (e.g., $[0, 1]$ or $[0, 255]$) to avoid clipping effects, which makes it rarely activated in practice.

These approaches enable the combination of rich learned priors and computational efficiency of Langevin-based sampling methods, creating a powerful framework for high-dimensional Bayesian inference in complex inverse problems.

1.4 Distributed MCMC for large inverse problems

While the different techniques detailed in Section 1.3 can improve MCMC scalability for high-dimensional problems, computational and memory constraints can be prohibitive for the largest inverse problems. This limitation is particularly visible when incorporating complex deep priors as presented in Section 1.2, which may involve neural networks with millions or billions of parameters requiring substantial computation loads. Even with optimal algorithmic choices, a single computational unit may lack resources to handle the simultaneous storage of high-resolution data, deep network parameters, and intermediate computations required for posterior sampling.

When single-machine implementations reach their computational or memory limits, distributed computing architectures become essential to conduct inference. These architectures enable the decomposition of computing tasks across multiple processing units, referred to as *workers*, while preserving the theoretical guarantees of MCMC sampling. The challenge lies in developing computationally-efficient distributed implementations of MCMC algorithms that, by nature, are sequential and often require access to the full current state of the chain.

This section first reviews fundamental notions in high-performance computing, and then focuses on distributed MCMC algorithms that build on these principles.

1.4.1 High-Performance Computing fundamentals

Sampling from models in which both the observations and the parameters are high-dimensional presents significant computational challenges that necessitate distributed computing approaches. High-Performance Computing (HPC) [Grama et al., 2003] provides the foundational framework for addressing these challenges through parallel and distributed architectures.

Distributed memory models and Message Passing Interface

Parallel computing typically involves multiple processors or cores operating within a shared memory space, such as multithreaded CPU programs or vectorized GPU kernels. Distributed computing, by contrast, follows a distributed-memory model where each process, or worker, maintains its own local address space, and communication occurs explicitly between processes [Eijkhout, 2023].

The Message Passing Interface (MPI) [Gropp et al., 2014] is the standard for distributed-memory programming in HPC. MPI defines protocols for independent processes, potentially running on different nodes or devices, to exchange data through well-defined communication primitives. Point-to-point operations, such as `Send` and `Recv`, allow explicit data transfers between specific processes, whereas collective operations like `Bcast`, `Scatter`, `Gather`, and `Allreduce` enable coordinated data distribution and aggregation across groups of processes. These communication patterns are illustrated in Figure 1.2.

A key advantage of MPI is that the distributed-memory abstraction is independent of the underlying hardware. For instance, multiple MPI processes running on the same CPU node may share physical RAM but remain isolated at the address-space level, requiring explicit communication. On multi-GPU systems, each GPU has private device memory. Modern MPI implementations support

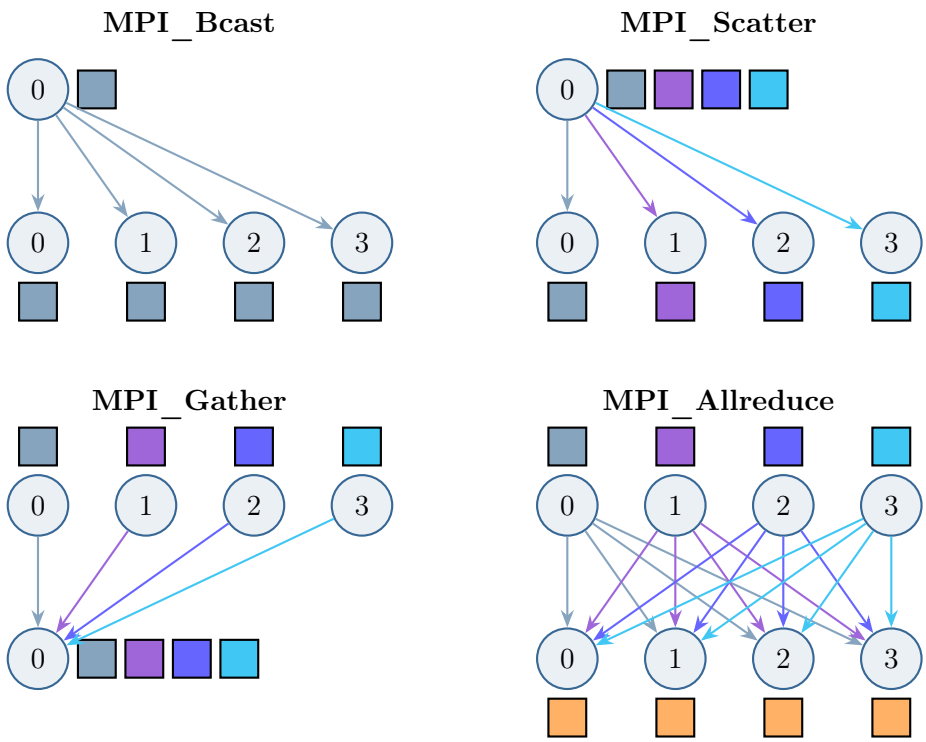

Figure 1.2: Examples of MPI collective communication operations. Circles represent workers (labeled by rank), and rectangles represent data blocks. `Bcast` broadcasts identical data from the root process to all workers. `Scatter` distributes unique data chunks from the root to each worker. `Gather` collects data from all workers to the root. `Allreduce` performs a reduction operation (e.g. sum, max) on distributed data and returns the result (orange blocks) to all workers.

CUDA-aware communication, enabling direct GPU-to-GPU data exchange without staging through host memory [Shamis et al., 2015]. This capability reduces transfer overhead and enables efficient execution of distributed algorithms on both multi-node and multi-GPU platforms. Combining distributed CPU computation with GPU acceleration allows massive parallelism for computationally intensive operations, particularly advantageous for tasks requiring highly parallel computations such as matrix operations or element-wise transformations on large datasets.

Tasks coordination and synchronization trade-offs

A fundamental design choice in distributed computing is between synchronous and asynchronous parallel algorithms [Eijkhout, 2023]. Synchronous algorithms enforce global consistency by requiring all workers to reach synchronization points before performing new tasks. When applied to MCMC algorithms, this ensures

that the distributed chain retains the same convergence guarantees as the sequential implementation. However, it can also lead to idle time when computational loads are unevenly balanced across workers: processes assigned smaller portions of the workload complete their tasks sooner and must wait for the others to finish, before the next iteration can begin.

Asynchronous algorithms relax synchronization constraints, allowing processes to advance independently based on locally available information. While this improves computational efficiency by reducing idle time, it introduces theoretical challenges: processes may operate on stale or inconsistent information, potentially affecting convergence properties.

Communication overhead is a critical factor in this trade-off. Synchronous algorithms require more frequent coordinated communication, which can become a bottleneck in large-scale systems. Effective load-balancing strategies must account for both heterogeneous node performance and variable computational costs associated with MCMC updates.

Distributed computing paradigms and architectures

Having established the communication models and synchronization trade-offs, this part examines the primary distributed computing paradigms that have been adapted for large-scale computational problems. Each paradigm [Eijkhout, 2023] offers distinct advantages depending on the problem structure and computational constraints, progressing from simple centralized coordination to more sophisticated parallel designs.

Centralized Coordination Architectures: Centralized paradigms, such as the *Master-Worker* and *Client-Server* models, rely on a designated coordinator node, *i.e.*, the master or the server, that maintains global state and orchestrates the computational process, while delegating specific operations to subordinate nodes. The coordinator assigns tasks to workers based on their availability and capabilities, monitors progress, handles fault tolerance through task reassignment, and aggregates results as they become available. This architecture provides clear separation of concerns, simplified state management, and flexibility in workload distribution through dynamic task allocation and load balancing. Depending on the design, the coordinator may either respond to worker requests for tasks (as in client-server systems) or actively push work to available workers (as in master-worker patterns), thus adapting to heterogeneous computational costs and varying worker capabilities. However, centralized approaches may encounter bottlenecks

or single points of failure when the coordinator becomes overwhelmed by coordination overhead, particularly as the system scales to larger numbers of workers. While these paradigms are well suited to heterogeneous or loosely coupled workloads, they often involve uneven task granularity and irregular communication patterns, making it challenging to achieve balanced utilization across nodes.

Single Program Multiple Data (SPMD): In contrast to centralized coordination, SPMD distributes responsibilities across all processes. Each process executes the same program but operates on distinct data subsets. SPMD implementations relying on MPI assign unique ranks to workers, which can act on different data subsets based on their rank. Rather than relying on a central coordinator, SPMD approaches distribute coordination through collective communication operations, where processes collaborate as peers to manage distributed state. This decentralized model scales better to larger processor counts since only the data is distributed, not the tasks with possibly different computational complexity. However, this architecture requires careful algorithm design to maintain consistency across all processes without centralized oversight.

1.4.2 Existing distributed MCMC algorithms

The landscape of distributed MCMC algorithms [Angelino et al., 2016] has evolved to tackle the computational and memory challenges inherent to high-dimensional Bayesian inference. Whether the goal is to accelerate sampling, improve scalability, or achieve both simultaneously, various strategies have been developed.

As discussed throughout this section, these approaches lead to different trade-offs between model assumptions, the level of approximation allowed, and communication constraints. In this work, the focus is on inverse problems where both the observations and the parameters are high-dimensional, *i.e.*, $M, N > 10^6$, which introduces additional difficulties. For problems of such scale, even running a single chain on a dedicated worker can be challenging. Consequently, distributed MCMC algorithms that generate multiple independent chains in parallel have not been considered much in this work. The emphasis is on methods that distribute the sampling of a single chain across multiple workers, following *divide-and-conquer* principles. The existing approaches proposed in the literature can be broadly classified into two categories: synchronous and asynchronous distributed samplers.

Synchronous distributed samplers

Aggregation and Consensus-based sampling methods: Consensus-based approaches are best adapted to *tall data* problems, where only the size of the observations is large. These methods assume conditional independence of the observations given the parameters, and typically partition the observation data across workers, each computing local subposteriors defined as

$$\pi(\boldsymbol{x}|\boldsymbol{y}) \propto \pi(\boldsymbol{x}) \prod_{i=1}^{M} \pi(\boldsymbol{y}_i|\boldsymbol{x}) \quad \rightsquigarrow \quad \pi_i(\boldsymbol{x}|\boldsymbol{y}_i) \propto \pi(\boldsymbol{x})^{1/M} \pi_i(\boldsymbol{y}_i|\boldsymbol{x}), \tag{1.32}$$

which are then combined to approximate the global posterior distribution. The diversity of aggregation strategies reflects different trade-offs between computational simplicity and approximation accuracy.

Building on similar ideas, Jordan et al., 2019 develop a Communication-efficient Surrogate Likelihood (CSL) framework that constructs global likelihood surrogates. When applied to Bayesian inference, their approach, based on Taylor series expansions of the negative log-likelihood, facilitates confidence interval construction while improving computational efficiency. However, this approach relies on a Master-Worker architecture in which only the main worker runs the MCMC chain after gathering local gradients from the others, which is not ideal for scalability. In addition, the closeness of the surrogate to the true posterior distribution highly depends on the choice of the expansion point, which requires extra computations to be selected appropriately.

In contrast, Tzikas et al., 2024 propose a decentralized sampling algorithm inspired by the Consensus Alternating Direction Method of Multipliers (C-ADMM). In this framework, the dual variables introduced in ADMM are updated *via* a consensus mechanism. Despite theoretical convergence guarantees in Wasserstein distance and strong empirical performance, the method has only been assessed on low-dimensional linear and logistic regression problems, with parameter spaces of dimension 2 and 3, respectively. The approach would likely struggle to scale to settings where N is large, since the full parameter vector $\boldsymbol{x}$ must be available for local updates and hence replicated on each worker.

This last comment can be generalized to most consensus-based methods, which are typically tailored to low-dimensional problems with large datasets, but not to high-dimensional inverse problems where both M and N are large.

Structured sampling and block-based approaches: Efficiently exploring multimodal distributions can become challenging, especially in high-dimensional spaces due to the curse of dimensionality. To address these challenges, Hafych et al. (2022) propose a distributed strategy based on parameter-space partitioning, where each worker is associated with a specific subspace, ideally corresponding to one mode of the target distribution. Workers independently sample within their assigned subspaces, and the resulting samples are later reweighted and combined. This design improves parallel efficiency and facilitates multimodal exploration. However, it relies on an exploration phase to identify relevant subspaces, which can become computationally expensive or inaccurate in high-dimensional settings. Moreover, each chain still evolves in the full-dimensional parameter space, which can be prohibitive, as discussed in the introduction of Section 1.4.2.

Instead of relying on a partitioning of the parameter space, Thouvenin et al., 2024 leverages the structural dependencies between parameters and observations. In particular, their work exploits the hypergraph structure underlying the posterior distribution of many inverse problems, to partition both the observations and the parameters over several workers in a SPMD framework. This approach was shown to offer more flexibility to use many workers with a balanced computing load, while reducing communication costs compared to alternatives based on centralized architectures. This approach is prone to high scalability and can potentially deal with high-dimensional problems with both large M and N values, under mild assumptions.

Asynchronous samplers

In practice, synchronous samplers can suffer from load imbalance and communication overhead, particularly in heterogeneous environments where differences in worker performance cause uneven progress in the construction of the samples.

To mitigate these issues, asynchronous sampling relaxes synchronization constraints and enables parallel updates of local variables without requiring strict coordination. This strategy can substantially reduce runtime in large-scale inference. However, these gains come at the cost of introducing more complex scheduling and communication patterns, potential model approximations, and additional challenges in ensuring convergence of the underlying MCMC.

A reference asynchronous sampler was proposed by Terenin et al., 2020, who provide a comprehensive theoretical framework for analyzing asynchronous Gibbs sampling. Their work extends the study of De Sa et al., 2016, and introduces an

exact version of the asynchronous Gibbs sampler. While standard asynchronous Gibbs can diverge for certain targets, their exact version, involving a Metropolis-Hastings accept-reject step, can ensure convergence under mild conditions.

Ahn et al., 2014 introduce a distributed MCMC based on the Stochastic Gradient Langevin Dynamics (SGLD) algorithm. The approach builds on a Master-Worker architecture, where each worker possesses a subset of the data and computes local SGLD updates, referred to as a *trajectory*, on the local chain it currently handles. The master periodically reassigns chains while managing load balancing. This approach can potentially be hard to scale to high-dimensional parameter spaces, since each worker still have to compute short MCMC chains over the full parameter space.

Considering also a Master-Worker architecture, Simsekli et al., 2018 develop an asynchronous sampler for non-convex optimization based on the L-BFGS algorithm. In this approach, a master node maintains the global iterate while multiple workers independently compute and send update vectors based on potentially outdated versions of that iterate. The master sequentially aggregates these asynchronous updates to refine the current estimate, enabling parallel progress but introducing delays and inconsistencies that complicate practical stability.

A more specific approach by Atchadé et al., 2023 focuses on sparse Bayesian inference problems using the spike-and-slab prior. This method demonstrates linear mixing time bounds and maintains correct signal recovery in high-dimensional regression. However, the methodology remains limited to sparse models and does not generalize easily to broader classes of inverse problems.

The diverse landscape of distributed MCMC methods reflects the various trade-offs between computational efficiency, theoretical guarantees, communication overhead, and ad-hoc approaches to exploit problem-specific structures. Selecting an approach typically depends on problem characteristics, available computational resources, and acceptable approximation levels. In particular, many existing distributed MCMC methods target either low-dimensional problems requiring speed-up or tall data settings, and often rely on centralized architectures that become bottlenecks at scale.

1.4.3 Towards distributed MCMC algorithms with deep priors

When both the sizes of the parameters and the observations increase, scalability becomes a major challenge for Bayesian inference. A promising yet relatively under-explored direction to address this limitation consists in reformulating ex-

isting sequential MCMC algorithms into distributed versions that preserve their theoretical guarantees and estimation accuracy. Among the existing contributions, the methodology proposed by Thouvenin et al., 2024 appears particularly promising, as it exploits the structural properties of many inverse problems to enable efficient sampling within a SPMD framework, without requiring approximations due to the distributed setting.

Building upon this line of work, an important next step consists in integrating deep priors into such distributed MCMC frameworks. This represents a significant opportunity, particularly in high-dimensional problems where neural networks excel at capturing complex dependencies. Furthermore, the growing accessibility of GPU clusters and supercomputing resources makes distributed Bayesian sampling with deep priors both timely and practically relevant.

To the best of our knowledge, no existing distributed MCMC methodology has yet addressed the incorporation of deep priors, particularly within the PnP framework. This thesis aims to fill this gap by proposing distributed versions of existing sequential MCMC algorithms involving deep priors. The study is restricted to synchronous implementations, as these preserve the theoretical guarantees of their sequential counterparts and do not require additional convergence analyses. The investigation of asynchronous extensions will be pursued in future research conducted by Nicolas Goeman as part of his doctoral work at the CRIStAL lab.

1.5 Contributions

This thesis makes both methodological and implementation-level contributions to the development of distributed MCMC algorithms inspired by PnP approaches. The work combines methodological aspects, software design, and experimental validation, with the objective of enabling scalable inference methods for high-dimensional imaging problems. The contributions are organized into four main chapters, summarized below.

Chapter 2 develops a methodology for constructing distributed PnP-inspired MCMC algorithms and carefully investigates favorable structures for their implementation, considering both the likelihood and the learned prior. The methodology is illustrated through the Plug-and-Play Unadjusted Langevin Algorithm (PnP-ULA) [Laumont et al., 2022], with a CNN. The proposed distributed algorithm leverages the structural properties of the model within an SPMD framework inspired by Thouvenin et al., 2024.

Chapter 3 introduces a modular software framework to implement distributed MCMC algorithms, emphasizing the need for communication-efficient strategies and adaptability to heterogeneous computing environments. The implementation abstracts out the generic structure of MCMC algorithms to facilitate the integration of various samplers, model structures, and prior types. The chapter also discusses practical considerations to deploy the framework on multi-GPU systems.

Chapter 4 validates the proposed distributed PnP-ULA through several experiments on large-scale synthetic imaging problems. The results demonstrate significant scalability gains compared to sequential implementations, with analysis of runtime and communication costs across different problem sizes and deep denoiser architectures.

Chapter 5 synthesizes the main conclusions and experimental results, highlighting the strengths and limitations of the proposed distributed methodology. It also outlines future research directions, including asynchronous extensions, different communication schemes, and generalization to broader network architectures and transition kernels.

Overall, this work has led to several publications and presentations during the course of the thesis, listed hereafter:

Journal papers

- Maxime Bouton et al. [2025a]. "A Distributed Plug-and-Play MCMC Algorithm for High-Dimensional Inverse Problems". Submitted to IEEE Transactions on Computational Imaging.

Conference papers

- Maxime Bouton et al. [June 2025b]. "Multi-GPU Distributed PnP-ULA for High-Dimensional Imaging Inverse Problems". In: *23rd IEEE Statistical Signal Processing Workshop (IEEE SSP 2025)*. Winner for Best Student Paper Award. Edimbourg, United Kingdom. URL: https://hal.science/hal-05143098
- Maxime Bouton et al. [Aug. 2025c]. "Un Échantillonneur Plug and Play Distribué Pour La Résolution de Problèmes Inverses de Très Grande Dimension". In: *Colloque Gretsi 2025*. Strasbourg, France. URL: https://hal.science/hal-05143079

Communications

- Maxime Bouton et al. [Mar. 27, 2025d]. *Une approche PnP distribuée pour léchantillonnage de problèmes inverses en grande dimension.* Journée GdR IASIS - "Evènements rares et réduction de modèles". IHP, Paris, France
- Maxime Bouton et al. [May 13, 2025e]. *Une approche PnP distribuée pour léchantillonnage de problèmes inverses en grande dimension.* Journée GdR IASIS - "Unrolling and un/self/supervised learning for inverse problems". INRIA, Paris, France

Chapter 2

A distributed Plug-and-Play MCMC algorithm

Contents

2.1 Problem statement 28

2.1.1 Operator locality 29

2.1.2 Model assumptions 31

2.2 Choice of the denoiser for distributed inference . . . 33

2.3 Proposed distributed sampler 38

2.4 Application to imaging inverse problems 41

2.4.1 Inpainting under Gaussian noise 41

2.4.2 Deconvolution under Gaussian noise 42

2.4.3 Deconvolution under Poisson noise 42

2.5 Conclusion 44

This chapter introduces a distributed Plug-and-Play sampler to address the scalability challenges outlined in Chapter 1. By exploiting the structure of the model's operators, the approach builds on the Split Gibbs sampler (SGS) [Vono et al., 2021] and the Plug-and-Play Unadjusted Langevin Algorithm (PnP-ULA) [Laumont et al., 2022], and investigates their deployment on a Single Program Multiple Data architecture [Thouvenin et al., 2024]. The interest of the proposed approach is illustrated on several typical imaging inverse problems.

The chapter is organized as follows. Section 2.1 introduces the considered model, with the operator locality assumptions required for an efficient distributed

sampling, *i.e.*, mitigating the cost induced by communications. Section 2.2 discusses CNN-based priors amenable to distributed implementation. Section 2.3 then details the proposed distributed sampler. Section 2.4 finally instantiates the method for several imaging inverse problems, including deconvolution and inpainting under different types of noise.

The material presented in this chapter is associated to the following publications:

- Maxime Bouton et al. [2025a]. "A Distributed Plug-and-Play MCMC Algorithm for High-Dimensional Inverse Problems". Submitted to IEEE Transactions on Computational Imaging.
- Maxime Bouton et al. [June 2025b]. "Multi-GPU Distributed PnP-ULA for High-Dimensional Imaging Inverse Problems". In: *23rd IEEE Statistical Signal Processing Workshop (IEEE SSP 2025)*. Winner for Best Student Paper Award. Edimbourg, United Kingdom. URL: https://hal.science/hal-05143098
- Maxime Bouton et al. [Aug. 2025c]. "Un Échantillonneur Plug and Play Distribué Pour La Résolution de Problèmes Inverses de Très Grande Dimension". In: *Colloque Gretsi 2025*. Strasbourg, France. URL: https://hal.science/hal-05143079

2.1 Problem statement

This section considers models whose posterior distribution takes the form

$$\pi(\boldsymbol{x}|\boldsymbol{y}) \propto p(\boldsymbol{x}) \exp\big(-f_1(\boldsymbol{H}_1\boldsymbol{x}) - f_2(\boldsymbol{H}_2\boldsymbol{x})\big), \tag{2.1}$$

where $f_1 \colon \mathbb{R}^{M_1} \to (-\infty, +\infty]$ is a smooth function, $f_2 \colon \mathbb{R}^{M_2} \to (-\infty, +\infty]$ a convex but possibly non-smooth function, and $\boldsymbol{H}_1 \colon \mathbb{R}^N \to \mathbb{R}^{M_1}$, $\boldsymbol{H}_2 \colon \mathbb{R}^N \to \mathbb{R}^{M_2}$ are linear operators. The function $p : \mathbb{R}^N \to \mathbb{R}$ may represent a component of the prior distribution. This work focuses on cases where both the observations and the parameter $\boldsymbol{x}$ are high-dimensional, and where the operators defining this posterior act locally with respect to a partition of $\{1, \dots, N\}$. This means that part of the output vector can be formed from only part of the input vector. Under this assumption, the algorithm decomposes into loosely dependent tasks, that can be executed by different workers. The limited dependencies between tasks are managed through lightweight inter-worker communications. This key notion of locality is formalized below.

2.1.1 Operator locality

Definition 1 (Local selection operators)**.** Let $(\mathbb{B}_b)_{1\leq b\leq B}$ be a partition of $\{1,\dots,N\}$ into $B>1$ subsets, and, for every $b\in\{1,\dots,B\}$, $\sharp\,\mathbb{B}_b = N_b \neq 0$. Let $(\widetilde{\mathbb{B}}_b)_{1\leq b\leq B}$ be an extension of this partition, such that, for every $b\in\{1,\dots,B\}$, $\mathbb{B}_b\subseteq\widetilde{\mathbb{B}}_b$ and $\sharp\,\widetilde{\mathbb{B}}_b = \widetilde{N}_b > N_b$. Let, for every $b\in\{1,\dots,B\}$, $\boldsymbol{S}_b\in\{0,1\}^{\widetilde{N}_b\times N}$ be a selection operator (*i.e.*, each row only contains 0 but a single 1) satisfying

$$\widetilde{\mathbb{B}}_b := \bigcup_{j\in\{1,\dots,\widetilde{N}_b\}} \{n\in\{1,\dots,N\}\,|\,\boldsymbol{S}_b^{(j,n)}=1\}. \tag{2.2}$$

Then, $\{\boldsymbol{S}_b\}_{1\leq b\leq B}$ is called a *family of local selection operators* with respect to $(\mathbb{B}_b)_{1\leq b\leq B}$ if

(i) for every $b\in\{1,\dots,B\}$, $\widetilde{N}_b - N_b \ll \min_{b'\in\{1,\dots,B\}} N_{b'}$;

(ii) $\max_{n\in\{1,\dots N\}} \sharp\{b\in\{1,\dots,B\}\,|\,n\in\widetilde{\mathbb{B}}_b\} \ll B$.

A few remarks can be done on Definition 1. First, (2.2) implies that $\{1,\dots,N\} = \bigsqcup_{b=1}^{B}\widetilde{\mathbb{B}}_b$. Further, condition (ii) ensures that each element $n\in\{1,\dots,N\}$ is only selected by a few operators $\{\boldsymbol{S}_b\}_{1\leq b\leq B}$. Combined with (i), it further controls the size of the overlapping areas. In other words, the combination of conditions (i) and (ii) ensures that $(\widetilde{\mathbb{B}}_b)_{1\leq b\leq B}$ is close to be a partition of $\{1,\dots,N\}$.

In a distributed computing setting, the overlapping areas determine the messages exchanged between workers. To reduce communication costs, the analysis considers *localized operators*, as defined below.

Definition 2 (Localized operator)**.** An operator $G\colon\mathbb{R}^N\to\mathbb{R}^M$ is said to be *localized* if it can be decomposed into a family of local selection operators $(\boldsymbol{S}_b)_{1\leq b\leq B}$ as

$$\forall b\in\{1,\dots,B\},\forall\boldsymbol{x}\in\mathbb{R}^N,\quad ([G(\boldsymbol{x})]^{(i)})_{i\in\mathbb{I}_b} = G_b(\boldsymbol{S}_b\boldsymbol{x}), \tag{2.3}$$

where $(\mathbb{I}_b)_{1\leq b\leq B}$ is a partition of $\{1,\dots,M\}$ into B subsets, with $M_b=\sharp\,\mathbb{I}_b>0$, and $G_b\colon\mathbb{R}^{\widetilde{N}_b}\to\mathbb{R}^{M_b}$. To simplify the notation, the expression $[G(\boldsymbol{x})]_b$ is used hereafter instead of $([G(\boldsymbol{x})]^{(i)})_{i\in\mathbb{I}_b}$. Example 1 and Figure 2.1 illustrate how this definition applies.

A localized operator $G\colon\mathbb{R}^N\to\mathbb{R}^M$ with respect to a family of local selection operators $(\boldsymbol{S}_b)_{1\leq b\leq B}$ can be evaluated in a distributed manner (see, *e.g.*, [Thouvenin et al., 2024] for details). Let $\boldsymbol{x}\in\mathbb{R}^N$ and $b\in\{1,\dots,B\}$. As per (2.3), each block $[G(\boldsymbol{x})]_b$ only depends on the locally selected block $\boldsymbol{S}_b\boldsymbol{x}$ (or equivalently, $\mathbb{I}_b$ only depends on the neighborhood $\widetilde{\mathbb{B}}_b$ as given in Definition 1). In a distributed

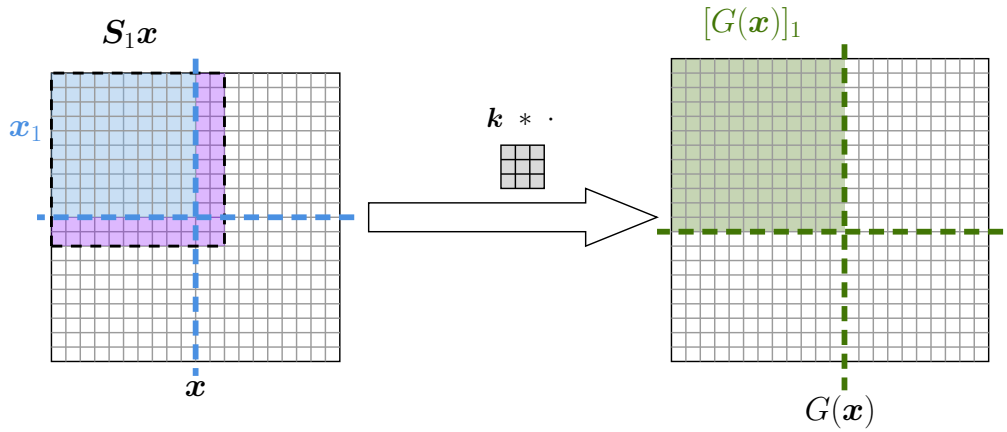

Figure 2.1: Illustration of a 2D linear convolution operator G defined by a 3×3 kernel $\boldsymbol{k}$, applied to an image $\boldsymbol{x}$ of size $N = 20 \times 20$ (hence $G(\boldsymbol{x}) \in \mathbb{R}^M$ with $M = 22 \times 22$). The input $\boldsymbol{x}$ is partitioned into $B = 4$ blocks (in dashed blue lines). Resulting blocks in $G(\boldsymbol{x})$ are delimited by dashed green lines. The block $\boldsymbol{x}_1$ assigned to worker 1 is in blue, with corresponding block $[G(\boldsymbol{x})]_1$ in green. The purple area contains pixels received by worker 1 to compute $[G(\boldsymbol{x})]_1$. Pixels in $\boldsymbol{S}_1\boldsymbol{x}$ are in the union of the blue and purple areas.

computing setting, computations underlying $[G(\boldsymbol{x})]_b$ can be assigned to a single worker b, with $\left(x^{(n)}\right)_{n\in\mathbb{B}_n}$ directly stored in its local memory. To compute $[G(\boldsymbol{x})]_b$, the worker b receives $\left(x^{(n)}\right)_{n\in\widetilde{\mathbb{B}}_b\setminus\mathbb{B}_b}$ from a small number of workers $b' \neq b$ through small size communications. The choice of the partition $(\mathbb{B}_b)_{1\leq b\leq B}$ directly impacts the local selection operators underlying the operator G, hence significantly affects the efficiency of the associated communications.

Example 1. Figure 2.1 illustrates Definitions 1 and 2 with a 2D linear convolution operator defined by a kernel $\boldsymbol{k}$ of size 3×3. An input image $\boldsymbol{x} \in \mathbb{R}^N$ of size $N = 20 \times 20$ is considered, leading to $G(\boldsymbol{x}) \in \mathbb{R}^M$ with $M = 22 \times 22$. Dashed blue lines represent the partition of $\boldsymbol{x}$ into $B = 4$ blocks $\boldsymbol{x}_b = (x^{(n)})_{n\in\mathbb{B}_b}$, with $b \in \{1, \ldots, 4\}$. Each block is assigned to a single worker. The partition $[G(\boldsymbol{x})]_b = \left([G(\boldsymbol{x})]^{(i)}\right)_{i\in\mathbb{I}_b}$ is indicated with dashed green lines in the output domain. The block $\boldsymbol{x}_1$ (in blue) is assigned to worker 1, as well as the corresponding output block $[G(\boldsymbol{x})]_1$ (in green). The pixels $(x^{(n)})_{n\in\widetilde{\mathbb{B}}_1\setminus\mathbb{B}_1}$ (in purple) are required by worker 1 to compute $[G(\boldsymbol{x})]_1$. These pixels form a *ghost region* [Eijkhout, 2023, Section 7.5.2], and are received from other workers before computations can be conducted on worker 1.

2.1.2 Model assumptions

Let $B \in \mathbb{N}^*$ be the number of workers available. The posterior distribution (2.1) is assumed to satisfy the following assumptions, with associated notation in Table 2.1. These assumptions ensure both sampling stability and scalability efficiency of the proposed approach. See Remark 1 below.

Assumption 1.

(A1) The composition $f_1 \circ \boldsymbol{H}_1 \colon \mathbb{R}^N \to (-\infty, +\infty]$ is Lipschitz-differentiable, with Lipschitz constant $L > 0$.

(A2) $f_2 \colon \mathbb{R}^{M_2} \to \big(-\infty, +\infty\big]$ is proper, l.s.c. and convex.

(A3) The linear operators $\boldsymbol{H}_1 \colon \mathbb{R}^N \to \mathbb{R}^{M_1}$ and $\boldsymbol{H}_2 \colon \mathbb{R}^N \to \mathbb{R}^{M_2}$ are localized in the sense of Definition 2, with respect to the same partition $(\mathbb{B}_b)_{1\leq b\leq B}$ of $\{1, \ldots, N\}$.

(A4) For $i \in \{1, 2\}$, the function $f_i \colon \mathbb{R}^{M_i} \to \big(-\infty, +\infty\big]$ is block-additively separable

$$(\forall \boldsymbol{z} \in \mathbb{R}^{M_i}) \quad f_i(\boldsymbol{z}) = \sum_{b=1}^{B} f_{i,b}(\boldsymbol{z}_b), \tag{2.4}$$

where, for every $b \in \{1, \ldots, B\}$, $f_{i,b} \colon \mathbb{R}^{M_{i,b}} \to \big(-\infty, +\infty\big]$, and $\boldsymbol{z}_b = (z^{(m)})_{m\in\mathbb{I}_{i,b}}$, with $(\mathbb{I}_{i,b})_{1\leq b\leq B}$ the partition of $\{1, \ldots, M_i\}$ associated with $\boldsymbol{H}_i$ (see Table 2.1).

(A5) There exists a family $(\boldsymbol{S}_{0,b})_{1\leq b\leq B}$ of local selection operators with respect to the partition $(\mathbb{B}_b)_{1\leq b\leq B}$ such that the function $\log p$ in (2.1) is block-additively separable

$$(\forall \boldsymbol{x} \in \mathbb{R}^N) \quad \log p(\boldsymbol{x}) = \sum_{b=1}^{B} \log p_b(\boldsymbol{S}_{0,b}\boldsymbol{x}). \tag{2.5}$$

Assumptions (A3)–(A5) are essential to the distributed sampler proposed in this chapter, and call for a few remarks.

Remark 1.

1) Assumptions (A1) and (A2) are instrumental for Langevin-based transition kernels such as PSGLA [Salim et al., 2019], MYULA [Durmus et al., 2018] or PnP-ULA [Laumont et al., 2022].

Table 2.1: Notation for localized operators $\boldsymbol{H}_i$, with $i \in \{1, 2\}$.

$\boldsymbol{H}_i$	linear operator from $\mathbb{R}^N$ to $\mathbb{R}^{M_i}$;
$(\boldsymbol{H}_{i,b})_{1\leq b\leq B}$	decomposition of $\boldsymbol{H}_i$ such that, for every $b \in \{1, \dots, B\}$, $\boldsymbol{H}_{i,b} \colon \mathbb{R}^{\widetilde{N}_{i,b}} \to \mathbb{R}^{M_{i,b}}$ and $[\boldsymbol{H}_i \boldsymbol{x}]_b = \boldsymbol{H}_{i,b} \boldsymbol{S}_{i,b} \boldsymbol{x}$;
$(\boldsymbol{S}_{i,b})_{1\leq b\leq B}$	family of local selection operators;
$(\mathbb{B}_b)_{1\leq b\leq B}$	partition of $\{1, \dots, N\}$ (same for both $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$);
$(N_b)_{1\leq b\leq B}$	cardinality for $(\mathbb{B}_b)_{1\leq b\leq B}$, with $\sum_{b=1}^{B} N_b = N$;
$(\widetilde{\mathbb{B}}_{i,b})_{1\leq b\leq B}$	extension of $(\mathbb{B}_b)_{1\leq b\leq B}$ with overlapping;
$(\widetilde{N}_{i,b})_{1\leq b\leq B}$	cardinality for $(\widetilde{\mathbb{B}}_{i,b})_{1\leq b\leq B}$;
$(\mathbb{I}_{i,b})_{1\leq b\leq B}$	partition of $\{1, \dots, M_i\}$;
$(M_{i,b})_{1\leq b\leq B}$	cardinality for $(\mathbb{I}_{i,b})_{1\leq b\leq B}$, with $\sum_{b=1}^{B} M_{i,b} = M_i$.

2) Assumption (A3) is motivated by the fact that if $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ are localized operators, there exists a basis where their matrix representation is block-sparse [Thouvenin et al., 2024]. This is satisfied for many usual inverse problems literature [Pereyra et al., 2016]. Typical examples are convolutions, finite pixel differences appearing in the TV norm [Rudin et al., 1992] or masking operators used in imaging applications (see also the experiments in Section 4.2).

3) When the likelihood is associated with $f_i \circ \boldsymbol{H}_i$, $i = 1$ or 2, Assumption (A4) ensures that the observations $\boldsymbol{y}$ can be partitioned into B conditionally-independent blocks $(y^{(m)})_{m\in\mathbb{I}_{i,b}}$ given $\boldsymbol{x}$. This is in particular the case when f_1 comes from the likelihood function associated with an additive white Gaussian noise.

When p in (2.1) is associated with a denoising network D_ϵ, its structure has an impact not only on estimation performance, but also on speed and scalability for distributed inference. For an efficient distributed implementation on an SPMD architecture, the structure of D_ϵ should be compatible with Assumption (A5) so that, after lightweight communications, it is fast to evaluate locally on each worker using subsets of locally available data. This is the topic of the next section.

2.2 Choice of the denoiser for distributed inference

Following the concerns on computing efficiency raised in the previous section, the focus is placed on Convolutional Neural Networks (CNNs) to encode the learned prior p. This choice is motivated by the fact that CNNs are essentially composed of convolutional layers and element-wise nonlinearities, which are localized in the sense of Definition 2. This section describes how their architecture can be distributed under Assumption (A5). In the simulations, particular attention is given to three state-of-the-art CNNs, namely DnCNN [K. Zhang et al., 2017], DDFB [Repetti et al., 2022], and DRUNet [K. Zhang et al., 2021], which are detailed in the examples below. These networks illustrate the variety within CNN architectures: DnCNN uses a traditional CNN architecture, DDFB encodes an unrolled algorithm, and DRUNet employs a more complex design including skip-connections and down-/upsampling operations.

Denoising CNNs

Let $D_\epsilon \colon \mathbb{R}^N \to \mathbb{R}^N$ be a denoising CNN composed of $K \in \mathbb{N}^*$ layers. As a reminder, D_ϵ aims to produce a good estimate of an unknown image $\overline{\boldsymbol{x}}$ from a noisy version $\boldsymbol{v} = \overline{\boldsymbol{x}} + \epsilon \boldsymbol{w}$, with $\epsilon > 0$ the standard deviation of the noise, and $\boldsymbol{w}$ a realization of a standard multivariate Gaussian distribution. In other words, the network is trained such that $D_\epsilon(\boldsymbol{v}) \approx \overline{\boldsymbol{x}}$. In the following, the input image dimension is $N = C \times N_y \times N_x$, with $C \in \mathbb{N}^*$ the number of image channels and $N_y \times N_x$ its spatial dimensions.

The architecture of D_ϵ can generally be described as a function of $G_\epsilon = T_K \circ \cdots \circ T_1$, where for every $k \in \{1, \dots, K\}$

$$T_k \colon \mathbb{R}^{N_k} \to \mathbb{R}^{N_{k+1}} \colon \boldsymbol{v} \mapsto \eta_k(\boldsymbol{W}_k \boldsymbol{v} + \boldsymbol{b}_k). \tag{2.6}$$

In (2.6), the input and output dimensions are $N_k = C_k \times N_{y,k} \times N_{x,k}$ and $N_{k+1} = C_{k+1} \times N_{y,k+1} \times N_{x,k+1}$, respectively, with $N_1 = N_{K+1} = N$. Each layer T_k contains a linear operator $\boldsymbol{W}_k \in \mathbb{R}^{M_k \times N_k}$, with $M_k = P_k \times N_{y,k} \times N_{x,k}$, a bias $\boldsymbol{b}_k \in \mathbb{R}^{M_k}$, and a non-linear activation function $\eta_k \colon \mathbb{R}^{N_k} \to \mathbb{R}^{N_k}$. For the sake of simplicity, the following directly considers the case in which the linear operator $\boldsymbol{W}_k$ represents a convolution. However, in practice it can also account for skip-connections, by defining $\boldsymbol{W}_k$ as the concatenation of a convolution and the identity operator (such that the current latent variable is carried forward until needed in deeper layers, *e.g.*, for DRUNet described in Example 4). The convolution

operator $\boldsymbol{W}_k$ can be expressed, for every $\boldsymbol{v} = (\boldsymbol{v}_c)_{1\leq c\leq C_k} \in \mathbb{R}^{N_k}$, as

$$\boldsymbol{W}_k\boldsymbol{v} = \left(\sum_{c=1}^{C_k} \boldsymbol{v}_c * \boldsymbol{w}_{c',c}\right)_{1\leq c'\leq P_k} \in \mathbb{R}^{N_k}, \tag{2.7}$$

where $*$ is a half-padding convolution with kernel $\boldsymbol{w}_{c',c} \in \mathbb{R}^{L_y\times L_x}$ for output feature $c' \in \{1,\dots,P_k\}$ and input feature $c \in \{1,\dots,C_k\}$.

Example 2. DnCNN [K. Zhang et al., 2017] is a simple version of (2.6), where $N_2 = \cdots = N_K = P \times N_y \times N_x$, and, for every $k \in \{1,\dots,K\}$, $P_k = P$. In other words, the first layer expands the input image from C channels to P features, keeping the same spatial dimension. All the following layers keep the same dimension, except the last layer that reduces from P features to C channels. It can be expressed as

$$D_\epsilon(\boldsymbol{v}) = (\mathrm{Id} - G_\epsilon)(\boldsymbol{v}). \tag{2.8}$$

In general, convolution kernels are chosen of size $L_y \times L_x = 3^2$, with $K = 17$ or 20 layers. The activation functions are ReLU non-linearities on all layers apart from the last, where no activation function is used (*i.e.*, η_K is identity).

Example 3. The DDFB denoiser [Repetti et al., 2022] can be written as a sequential CNN with layers of the form (2.6) (see [Le et al., 2024]), and is obtained by unrolling a dual forward-backward algorithm [Combettes et al., 2010, 2011]. In a nutshell, DDFB is given by

$$D_\epsilon(\boldsymbol{v}) = \mathrm{proj}_{[0,1]^N}\left(\boldsymbol{v} - \gamma_K \boldsymbol{W}_K^* G_{\epsilon,\boldsymbol{v}}(\boldsymbol{W}_K\boldsymbol{v})\right), \tag{2.9}$$

with $G_{\epsilon,\boldsymbol{v}} = T_{K-1,\epsilon,\boldsymbol{v}} \circ \cdots \circ T_{1,\epsilon,\boldsymbol{v}}$. Each layer $k \in \{1,\dots,K-1\}$ is the concatenation of two sub-layers: one expanding the dimension of the image from C channels to P features keeping the spatial dimensions fixed (*via* the operator $\boldsymbol{W}_k \in \mathbb{R}^{N\times M}$, with $N = C \times N_y \times N_x$ and $M = P \times N_y \times N_x$), and one reducing back from P features to C channels (*via* the adjoint operator $\boldsymbol{W}_k^* \in \mathbb{R}^{M\times N}$). Specifically, for every $\boldsymbol{u} \in \mathbb{R}^M$,

$$T_{k,\epsilon,\boldsymbol{v}}(\boldsymbol{u}) = \mathcal{HT}_\epsilon\big(\boldsymbol{u} + \gamma_k \boldsymbol{W}_k \,\mathrm{proj}_{[0,1]^N}(\boldsymbol{v} - \boldsymbol{W}_k^*\boldsymbol{u})\big), \tag{2.10}$$

where $\gamma_k \in (0, 2/\|\boldsymbol{W}_k\|_2^2)$, $\mathrm{proj}_{\mathcal{C}}$ is a term-wise thresholding operator [Combettes et al., 2011] and $\mathcal{HT}_\epsilon$ is the element-wise hard-tan operator with hyperparameter $\epsilon > 0$, corresponding to a projection onto the hypercube $[-\epsilon,\epsilon]^M$. In general, convolution kernels are chosen of size $L_y \times L_x = 3^2$.

Example 4. DRUNet [K. Zhang et al., 2021] is a CNN with a UNet structure which can be written as (2.6). It is composed of $K = 2I + 3$ layers, including $I \in \mathbb{N}^*$ downsampling and upsampling layers. The layers $k \in \{2, \dots, K-1\}$ include residual operators $R_k : \mathbb{R}^{N_k} \to \mathbb{R}^{N_k}$, composed of $J \in \mathbb{N}^*$ residual blocks $R_k = R_{k,J} \circ \cdots \circ R_{k,1}$. In the following, the spatial dimensions N_x and N_y are assumed divisible by 2^I,possibly after padding, so that downsampling and upsampling operators can be applied without communications. For $\boldsymbol{v} \in \mathbb{R}^N$, the DRUNet architecture is given by

$$D_\epsilon(\boldsymbol{v}) = \boldsymbol{W}_K\Big(\big(G_\epsilon + \mathrm{Id}\big)\big(\boldsymbol{W}_1[\boldsymbol{v}, \epsilon \mathbf{1}_{N_y \times N_x}]\big)\Big), \tag{2.11}$$

with $[\boldsymbol{v}, \epsilon \mathbf{1}_{N_y \times N_x}] \in \mathbb{R}^{N_\epsilon}$ the concatenation of $\boldsymbol{v}$ and $\epsilon \mathbf{1}_{N_y \times N_x}$, $N_\epsilon = (C+1) \times N_y \times N_x$, $\boldsymbol{W}_1 \in \mathbb{R}^{M \times N_\epsilon}$ with $M = P \times N_y \times N_x$, $\boldsymbol{W}_K \in \mathbb{R}^{N \times M}$, and $G_\epsilon = T_{K-1} \circ \cdots \circ T_2 \colon \mathbb{R}^M \to \mathbb{R}^M$. Each layer k is defined as

$$\begin{aligned}
&\text{for } k \in \{2, \dots, I+1\} : && T_k : (\boldsymbol{v}_{k'})_{2 \le k' \le k} \;\mapsto\; (\boldsymbol{v}_{k'})_{2 \le k' \le k+1}, \\
&\text{for } k \in \{I+3, \dots, K-1\} : && T_k : \big((\boldsymbol{v}_{k'})_{2 \le k' \le K-k+1}, \boldsymbol{v}_k\big) \;\mapsto \\
& && \quad \big((\boldsymbol{v}_{k'})_{2 \le k' \le K-k}, \boldsymbol{v}_{k+1}\big),
\end{aligned}$$

where $\boldsymbol{v}_{k'} \in \mathbb{R}^{N_{k'}}$ with $N_2 = N_{K-1} = M$, and

$$\boldsymbol{v}_{k+1} = \begin{cases} \boldsymbol{\Phi}_k \circ R_k(\boldsymbol{v}_k), \text{ if } 2 \le k \le I+1, \\ R_{I+2}(\boldsymbol{v}_{I+2}), \text{ if } k = I+2, \\ R_k \circ \boldsymbol{\Psi}_k(\boldsymbol{v}_k + \boldsymbol{v}_{K+2-k}), \text{ if } k \ge I+3, \end{cases} \tag{2.12}$$

with $\boldsymbol{\Phi}_k$ and $\boldsymbol{\Psi}_k$ downsampling and upsampling operators. The residual blocks are defined, for every $j \in \{1, \dots, J\}$, as

$$R_{k,j} = \big(\boldsymbol{W}_{k,j,2} \circ \eta_k \circ \boldsymbol{W}_{k,j,1} + \mathrm{Id}\big) \colon \mathbb{R}^{N_k} \to \mathbb{R}^{N_k}, \tag{2.13}$$

where η_k is the ReLU non-linearity, $\boldsymbol{W}_{k,j,1} \colon \mathbb{R}^{N_k} \to \mathbb{R}^{M_k}$ and $\boldsymbol{W}_{k,j,2} \colon \mathbb{R}^{M_k} \to \mathbb{R}^{N_k}$ are convolution operators with $L_y = L_x = 3$ and $P_k = C_k$. Downsampling operators are given by

$$\boldsymbol{\Phi}_k : \boldsymbol{v} \in \mathbb{R}^{N_k} \mapsto \boldsymbol{V}_k \boldsymbol{W}_k \boldsymbol{v} \in \mathbb{R}^{N_{k+1}}, \tag{2.14}$$

where $\boldsymbol{W}_k : \mathbb{R}^{N_k} \to \mathbb{R}^{M_k}$ is a convolution operator with $L_y = L_x = 2$, $P_k = 2C_k$, $\boldsymbol{V}_k : \mathbb{R}^{M_k} \to \mathbb{R}^{N_{k+1}}$ is a subsampling operator selecting elements at even indices along both spatial dimensions, and $N_{k+1} = (2C_k) \times \lfloor N_{y,k}/2 \rfloor \times \lfloor N_{x,k}/2 \rfloor$.

Upsampling operators are given by

$$\boldsymbol{\Psi}_k : \boldsymbol{u} \in \mathbb{R}^{N_k} \mapsto \boldsymbol{W}_k \boldsymbol{U}_k \boldsymbol{u} \in \mathbb{R}^{N_{k+1}}, \tag{2.15}$$

where $L_y = L_x = 2$, $P_k = \lfloor C_k/2 \rfloor$, $\boldsymbol{U}_k : \mathbb{R}^{M_k} \to \mathbb{R}^{N_{k+1}}$ inserts a zero between each consecutive entries along the spatial dimensions, and $N_{k+1} = \lfloor C_k/2 \rfloor \times (2N_{y,k}) \times (2N_{x,k})$.

Proposed distributed implementation

This section presents a distributed strategy for (2.7). It relies on a Cartesian domain decomposition and communications detailed below. The chosen strategy builds on the overlap-save approach [Vetterli et al., 2014].

Domain decomposition and communications: For simplicity, a 1D domain decomposition of $\{1, \dots, C\} \times \{1, \dots, N_y\} \times \{1, \dots, N_x\}$ is described, where only the axis $\{1, \dots, N_y\}$ is partitioned. A 2D tessellation can be obtained using the same strategy on the axis $\{1, \dots, N_x\}$[1]. To obtain a partition into blocks of similar size, Assumption (A3) considers the sets $(\mathbb{B}_b)_{1 \leq b \leq B}$ with, for every $b \in \{1, \dots, B\}$,

$$\mathbb{B}_b = \{1, \dots, C\} \times \left\{ \left\lfloor \frac{(b-1)N_y}{B} \right\rfloor + 1, \dots, \left\lfloor \frac{bN_y}{B} \right\rfloor \right\} \times \{1, \dots, N_x\}, \tag{2.16}$$

with $\sharp \mathbb{B}_b = C \times N_{y,b} \times N_x$ and $N_{y,b} = \lfloor bN_y/B \rfloor - \lfloor (b-1)N_y/B \rfloor$. The sets $(\widetilde{\mathbb{B}}_b)_{1 \leq b \leq B}$ in (2.7) can be defined as

$$\widetilde{\mathbb{B}}_b = \{1, \dots, C\} \times \left\{ \left\lfloor \frac{(b-1)N_y}{B} \right\rfloor + 1, \dots, \left\lfloor \frac{bN_y}{B} \right\rfloor + L_y - 1 \right\} \times \{1, \dots, N_x\}, \tag{2.17}$$

with $\sharp \widetilde{\mathbb{B}}_b = \widetilde{N}_b = C \times \widetilde{N}_{y,b} \times N_x$ and $\widetilde{N}_{y,b} = N_{y,b} + L_y - 1$. Note that (2.16)–(2.17) satisfy Definition 1 (i) when $\min_{1 \leq b \leq B} N_b \geq L_y - 1$.

Figure 2.2 illustrates the partitioning and typical communication patterns in the proposed implementation. In particular, for an image $\boldsymbol{v} \in \mathbb{R}^N$, Figure 2.2 (a) represents the partition of $\boldsymbol{v}$ into $(\boldsymbol{v}_b)_{1 \leq b \leq B}$. Figure 2.2 (b) illustrates the communication pattern considered for the distributed implementation of (2.7). Each worker $b \in \{2, \dots, B\}$ is required to send an hyperslab covering the first $L_y - 1$ indices along axis N_y (in dashed purple lines) to worker $b - 1$, following the

[1] The same principle can be generalized to tensors of any order.

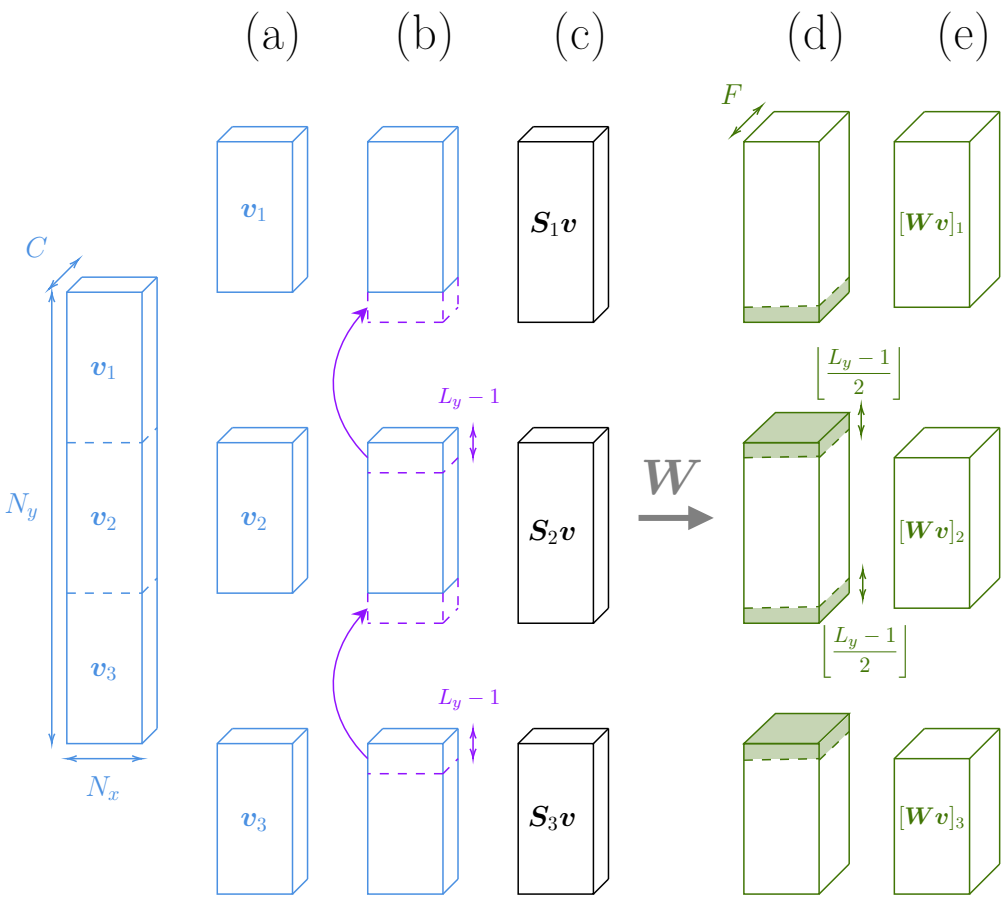

Figure 2.2: Communications and operations underlying the proposed distributed implementation of $\boldsymbol{W}_k$. The k index was omitted in all notations for clarity. Communications are represented by purple arrows, with messages in dashed purple lines. Discarded regions are shaded in green.

communication pattern illustrated by purple arrows. The message received by $b \in \{1, \dots, B-1\}$ is stored in a ghost region [Eijkhout, 2023, Section 7.5.2 p. 306], in dashed purple lines, to form $\boldsymbol{S}_b\boldsymbol{v} = (v^{(n)})_{n \in \widetilde{\mathbb{B}}_n}$.

Distributed implementation for $\boldsymbol{W}_k$: Let $k \in \{1, \dots, K\}$. The operator $\boldsymbol{W}_k$ in (2.7) can be decomposed thanks to an overlap-save approach [Vetterli et al., 2014, Section 3.9.2] as follows. For $\boldsymbol{v} \in \mathbb{R}^{N_k}$ and adapting notation in Section 2.2 to the distributed setting from Section 2.2,

$$\boldsymbol{W}_k\boldsymbol{v} = (\boldsymbol{W}_{k,b}\boldsymbol{S}_{k,b}\boldsymbol{v})_{1 \leq b \leq B}, \tag{2.18}$$

where $\boldsymbol{S}_{k,b} \in \{0,1\}^{\widetilde{N}_{k,b} \times N_k}$ with $\widetilde{N}_{k,b} = C_k \times \widetilde{N}_{y,k,b} \times N_{x,k}$, $\widetilde{N}_{y,k,b} = N_{y,k,b} + L_y - 1$, and $\boldsymbol{W}_{k,b} \in \mathbb{R}^{M_{k,b} \times \widetilde{N}_{k,b}}$ with $M_{k,b} = P_k \times N_{y,k,b} \times N_{x,k}$. More precisely, for $\widetilde{\boldsymbol{v}} = (\widetilde{\boldsymbol{v}}_c)_{1 \leq c \leq C_k} \in \mathbb{R}^{\widetilde{N}_{k,b}}$,

$$\boldsymbol{W}_{k,b}\widetilde{\boldsymbol{v}} = \left(\sum_{c=1}^{C_k} \boldsymbol{A}_{k,b}(\widetilde{\boldsymbol{v}}_c * \boldsymbol{w}_{c',c}) \right)_{1 \leq c' \leq P_k}, \tag{2.19}$$

with $\boldsymbol{A}_{k,b} \in \{0,1\}^{(N_{y,k,b} \times N_{x,k}) \times (\widetilde{N}_{y,k,b} \times N_{x,k})}$ a local selection operator. Steps (b)–(d) in Figure 2.2 illustrate the successive operations involved in (2.18)–(2.19). After communications (see Figure 2.2 (b)), each worker b can access $\boldsymbol{S}_{k,b}\boldsymbol{v}$ (Figure 2.2 (c)) for (2.19). Entries computed with the wrong boundary conditions are discarded via $\boldsymbol{A}_{k,b}$ (illustrated by the green-shaded areas in Figure 2.2 (d)), leading to the output blocks in Figure 2.2 (e).

The following paragraphs propose to distribute the computations of each successive layer $k \in \{1, \ldots, K\}$, by communicating the required boundary elements to enable the computation of local convolutions (2.19). In practice, the direction of communications in Figure 2.2 (b) is reversed from one convolution operator to another to maintain balanced workloads and memory usage for all workers over the sequence of convolutional layers. Indeed, workers at the edges of the partitioning, *i.e.* workers 1 and 3 in Figure 2.2, have respectively larger and smaller local outputs than in-between workers, *i.e.* worker 2 in this case. This occurs because the edge worker receives no data, while the other has incorrect boundary conditions on one side only, unlike in-between workers.

Other distributed strategies could be considered, such as communicating the receptive field of the full sequence of convolutions computed across the network [Galerne et al., 2024]. This alleviates the need for successive communications phases in each layer, at the cost of a significant increase in the size of messages as the input dimensions and the number of layers increase. Comparing different communication patterns is an interesting prospect for future work.

2.3 Proposed distributed sampler

Following the AXDA approach introduced in Section 1.3.2, the posterior distribution (2.1) is approximated by introducing an auxiliary variable $\boldsymbol{z} \in \mathbb{R}^{M_2}$ as

$$\pi_\rho(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{y}) \propto p(\boldsymbol{x}) \exp\Big(- f_1(\boldsymbol{H}_1\boldsymbol{x}) - f_2(\boldsymbol{z}) - \frac{1}{2\rho}\|\boldsymbol{z} - \boldsymbol{H}_2\boldsymbol{x}\|_2^2 \Big), \tag{2.20}$$

where the parameter $\rho > 0$ controls the coupling between $\boldsymbol{z}$ and $\boldsymbol{H}_2\boldsymbol{x}$. Note that the same technique could be applied to f_1 as well. Samples can be drawn from (1.26) with SGS [Vono et al., 2019], where the associated conditional distributions are

$$\pi_\rho(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) \propto p(\boldsymbol{x}) \exp\big(-f_1(\boldsymbol{H}_1\boldsymbol{x}) - \frac{1}{2\rho}\|\boldsymbol{z}-\boldsymbol{H}_2\boldsymbol{x}\|_2^2\big), \tag{2.21}$$

$$\pi_\rho(\boldsymbol{z}|\boldsymbol{y},\boldsymbol{x}) \propto \exp\big(-f_2(\boldsymbol{z}) - \frac{1}{2\rho}\|\boldsymbol{z}-\boldsymbol{H}_2\boldsymbol{x}\|_2^2\big). \tag{2.22}$$

Under Assumption (A4), PSGLA [Salim et al., 2019] can be used to sample (2.22). Further, to leverage a pre-trained prior, the PnP Unadjusted Langevin Algorithm (PnP-ULA) [Laumont et al., 2022] is employed to update $\boldsymbol{x}^{(t+1)}$ in (2.21). Such a PnP-ULA approach has the advantage that it ensures the existence of a proper prior distribution with pdf p in (2.1) associated with a denoiser D_ϵ (pre-trained at noise level $\epsilon > 0$), assuming $\boldsymbol{I}_N - D_\epsilon$ is Lipschitz continuous with constant $L_D > 0$ [Laumont et al., 2022, Section 3.2].

Hence, for $\kappa \in (0, \rho)$, the transition from iteration $t \in \mathbb{N}^*$ to $t+1$ reads

$$\begin{aligned}\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} &- \gamma \boldsymbol{H}_1^* \nabla f_1(\boldsymbol{H}_1\boldsymbol{x}^{(t)}) \\ &- \frac{\gamma}{\rho}\boldsymbol{H}_2^*(\boldsymbol{H}_2\boldsymbol{x}^{(t)} - \boldsymbol{z}^{(t)}) + \frac{\alpha\gamma}{\epsilon^2}\big(D_\epsilon(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big) \\ &+ \frac{\gamma}{\lambda}\big(\operatorname{proj}_{\mathcal{C}}(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big) + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)},\end{aligned} \tag{2.23}$$

$$\boldsymbol{z}^{(t+1)} = \operatorname{prox}_{\kappa f_2}\left(\boldsymbol{z}^{(t)} - \frac{\kappa}{\rho}\big(\boldsymbol{z}^{(t)} - \boldsymbol{H}_2\boldsymbol{x}^{(t+1)}\big) + \sqrt{2\kappa}\boldsymbol{\zeta}^{(t+1)}\right), \tag{2.24}$$

where $\boldsymbol{\xi}^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_N)$, $\boldsymbol{\zeta}^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_{M_2})$, and $\lambda, \alpha, \gamma > 0$ are such that

$$\begin{cases} 2(L + \|\boldsymbol{H}_2\|_2^2) + \alpha\epsilon^{-2}L_D \leq (2\lambda)^{-1}, \\ 3\gamma(L + \|\boldsymbol{H}_2\|_2^2 + \lambda^{-1} + \alpha\epsilon^{-2}L_D) < 1. \end{cases} \tag{2.25}$$

Note that variants of the proposed sampler (2.23)–(2.24) can be obtained with other combinations of a PnP-based transition kernel, *e.g.* [Faye et al., 2025], with a Langevin kernel relying on a proximal operator, such as MYULA [Durmus et al., 2018], PSGLA [Salim et al., 2019] or SK-ROCK [Pereyra et al., 2020].

The distributed setting of Section 2.1 paves the way to a distributed counterpart of (2.23)–(2.24) inspired by [Thouvenin et al., 2024], where the denoiser D_ϵ is also distributed as described in Section 2.2. The resulting distributed sampler is described in Algorithm 1.

Algorithm 1: Proposed distributed sampler.

Input: $\boldsymbol{y} \in \mathbb{R}^M$ and $\epsilon, \rho > 0$, $\alpha, \lambda > 0$ such that
$2\lambda\big(2(L + \|\boldsymbol{H}_2\|_2^2) + \alpha\epsilon^{-2}L_D\big) \leq 1$,
$\gamma \in \big(0, (L + \|\boldsymbol{H}_2\|_2^2 + \lambda^{-1} + \alpha\epsilon^{-2}L_D)^{-1}/3\big)$, $\kappa \in (0, \rho)$

for *each worker* $b \in \{1, \ldots, B\}$ **do** in parallel

- Load and store observation chunk $\boldsymbol{y}_b \in \mathbb{R}^{M_{1,b}}$;
- Initialize $\boldsymbol{x}_b^{(0)} \in \mathbb{R}^{N_b}$ and $\boldsymbol{z}_b^{(0)} \in \mathbb{R}^{M_{1,b}}$;
- **for** $t = 0$ **to** $T - 1$ **do**
  - `// Update` $\boldsymbol{x}_b$ `with PnP-ULA` [Laumont et al., 2022]
  - Communicate to retrieve $\big(\boldsymbol{S}_{i,b}\boldsymbol{x}^{(t)}\big)_{0\leq i\leq 2}$;
  - $\boldsymbol{u}_{1,b}^{(t)} = \boldsymbol{H}_{1,b}^* \nabla f_{1,b}(\boldsymbol{H}_{1,b}\boldsymbol{S}_{1,b}\boldsymbol{x}^{(t)})$;
  - $\boldsymbol{u}_{2,b}^{(t)} = \frac{1}{\rho}\boldsymbol{H}_{2,b}^*(\boldsymbol{H}_{2,b}\boldsymbol{S}_{2,b}\boldsymbol{x}^{(t)} - \boldsymbol{z}_b^{(t)})$;
  - Communicate to compute $\boldsymbol{v}_b^{(t)} = \sum_{j=1}^{B} \boldsymbol{S}_{1,j}^* \boldsymbol{u}_{1,b}^{(t)} + \boldsymbol{S}_{2,j}^* \boldsymbol{u}_{2,j}^{(t)}$;
  - Communicate and compute $D_\epsilon(\boldsymbol{x}^{(t)})_b$, see Section 2.2;
  - Sample $\boldsymbol{\xi}_b^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_{N_b})$;
  - $\boldsymbol{x}_b^{(t+1)} = \boldsymbol{x}_b^{(t)} - \gamma\boldsymbol{v}_b^{(t)} + \frac{\alpha\gamma}{\epsilon}\big(D_{\epsilon^2}(\boldsymbol{x}^{(t)})_b - \boldsymbol{x}_b^{(t)}\big) + \frac{\gamma}{\lambda}\big(\operatorname{proj}_{\mathcal{C}_b}(\boldsymbol{x}_b^{(t)}) - \boldsymbol{x}_b^{(t)}\big) + \sqrt{2\gamma}\boldsymbol{\xi}_b^{(t+1)}$;
  - `// Update` $\boldsymbol{z}_b$ `with PSGLA` [Salim et al., 2019]
  - Communicate to retrieve $\boldsymbol{S}_{2,b}\boldsymbol{x}^{(t+1)}$;
  - Sample $\boldsymbol{\zeta}_b^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_{M_{2,b}})$;
  - $\boldsymbol{z}_b^{(t+1)} = \operatorname{prox}_{\kappa f_{2,b}}\Big(\boldsymbol{z}_b^{(t)} - \frac{\kappa}{\rho}\big(\boldsymbol{z}_b^{(t)} - \boldsymbol{H}_{2,b}\boldsymbol{S}_{2,b}\boldsymbol{x}^{(t+1)}\big) + \sqrt{2\kappa}\boldsymbol{\zeta}_b^{(t+1)}\Big)$;

Output: $(\boldsymbol{x}^{(t)}, \boldsymbol{z}^{(t)})_{1\leq t\leq T}$

Under (A3)–(A4), all computations associated with f_1, f_2, $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ can be decomposed over $B \geq 2$ workers. In addition, as a direct consequence of [Laumont et al., 2022, Section 3.2], the proposed distributed sampler ensures that there exists a proper prior distribution with pdf p in (2.1) associated with the denoiser D_ϵ. Since D_ϵ is a CNN, its structure ensures that p satisfies Assumption (A5). This allows the PnP-ULA (Algorithm 1 lines 5–11) and PSGLA (lines 12–14) steps to be distributed on an SPMD architecture.

Remark 2. A few remarks can be made on Algorithm 1.

1) Most steps in Algorithm 1 do not require synchronization.

2) Communications are only required between a few workers in lines 5, 8, 9 and 12. The distributed PnP-ULA transition induces communications to compute $\nabla(f_1 \circ \boldsymbol{H}_1)$ (lines 5 and 8) and to apply the denoiser (line 9). Some entries in $\boldsymbol{x}^{(t+1)}$ are exchanged between a few workers in line 12 so that $\boldsymbol{z}$ can be updated locally (line 14).

3) The messages in line 5 can be grouped into a single message to reduce communication costs.

2.4 Application to imaging inverse problems

The general distributed algorithm presented above can be instantiated in various imaging inverse problems by specifying the quantities involved in each case. This section presents its adaptation to three representative problems, each formulated with either a Total Variation (TV) prior (1.8) or a PnP prior based on DDFB, DnCNN or DRUNet. These examples illustrate the versatility of the algorithm, provide the concrete setup used in the implementation example of Section 3.3, and serve as a benchmark for the evaluation of performance reported in Chapter 4.

2.4.1 Inpainting under Gaussian noise

The likelihood is

$$\boldsymbol{y} \mid \boldsymbol{x} \sim \mathcal{N}(\boldsymbol{M}\boldsymbol{x}, \sigma^2 \boldsymbol{I}_N), \tag{2.26}$$

with $\boldsymbol{M} \in \{0,1\}^{N \times N}$ a diagonal operator containing the observation mask, and $\sigma^2 > 0$ the noise variance. The pdf of the posterior distribution is of the form (2.1), with

$$\boldsymbol{H}_1 = \boldsymbol{M}, \quad f_1(\boldsymbol{H}_1 \boldsymbol{x}) = \frac{1}{2\sigma^2} \|\boldsymbol{y} - \boldsymbol{M}\boldsymbol{x}\|_2^2. \tag{2.27}$$

The masking operator $\boldsymbol{M}$ acts element-wise. No communication is needed to evaluate $\boldsymbol{H}_1$ and $\boldsymbol{H}_1^*$ in a distributed computing setting, *i.e.*, $M_{1,b} = N_b$ and $\boldsymbol{S}_{1,b} = \boldsymbol{I}_{N_b}$ in (2.4). Depending on the choice of prior, the functions p and $f_2 \circ \boldsymbol{H}_2$ can be identified as follows.

(i) **TV prior**: in this case, $p = \mathbb{1}_{\mathbb{R}_+^N}$, $f_2 = \beta\|\cdot\|_{2,1}$ and $\boldsymbol{H}_2 = \boldsymbol{D}$ (defined in (1.8)). The AXDA approach is applied as in (1.26). The latent variable $\boldsymbol{z}$ is updated with (2.24) and $\boldsymbol{x}$ with PSGLA [Durmus et al., 2018] as follows

$$\begin{aligned}\boldsymbol{x}^{(t+1)} = \operatorname{proj}_{\mathbb{R}_+^N}\Big(&\boldsymbol{x}^{(t)} - \gamma \boldsymbol{H}_1^* \nabla f_1(\boldsymbol{H}_1 \boldsymbol{x}^{(t)}) \\ &- \frac{\gamma}{\rho}\boldsymbol{H}_2^*(\boldsymbol{H}_2\boldsymbol{x}^{(t)} - \boldsymbol{z}^{(t)}) + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)}\Big),\end{aligned} \tag{2.28}$$

with $\boldsymbol{\xi}^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_N)$.

(ii) **Denoiser-based prior**: in this case, $f_2 \circ \boldsymbol{H}_2 = 0$, and p is taken as in [Laumont et al., 2022]. The posterior distribution can be directly sampled with PnP-ULA [Laumont et al., 2022] without using AXDA.

$$\begin{aligned}\boldsymbol{x}^{(t+1)} = &\boldsymbol{x}^{(t)} + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)} - \gamma \boldsymbol{H}_1^* \nabla f_1(\boldsymbol{H}_1 \boldsymbol{x}^{(t)}) \\ &+ \frac{\alpha\gamma}{\epsilon}\big(D_\epsilon(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big) + \frac{\gamma}{\lambda}\big(\operatorname{proj}_{\mathcal{C}}(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big),\end{aligned} \tag{2.29}$$

with $\boldsymbol{\xi}^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_N)$.

2.4.2 Deconvolution under Gaussian noise

The pdf of the posterior distribution is of the form (2.1), with $\boldsymbol{H}_1 = \boldsymbol{H} \in \mathbb{R}^{M\times N}$ a linear convolution operator. Communications similar to Section 2.2 are required to apply $\boldsymbol{H}_1$ and $\boldsymbol{H}_1^*$ in a distributed computing setting. The same setting and sampling strategy as Section 2.4.1 are considered.

2.4.3 Deconvolution under Poisson noise

The likelihood is

$$\boldsymbol{y} \mid \boldsymbol{x} \sim \mathcal{P}(\eta \boldsymbol{H}\boldsymbol{x}), \tag{2.30}$$

with $\mathcal{P}$ the Poisson distribution, $\boldsymbol{H} \in \mathbb{R}^{M\times N}$ a linear convolution operator, and $\eta > 1$ controlling the variance of the Poisson noise. The pdf of the posterior distribution is of the form (2.1), with $f_1 \circ \boldsymbol{H}_1 = 0$. The expression of f_2, $\boldsymbol{H}_2$ and p in (2.1) depends on the choice of prior (see the beginning of Section 2.4), as

detailed below with the corresponding sampling strategy adapted from Section 2.3. As in [Thouvenin et al., 2024], two blocks are considered in the AXDA auxiliary variable $\boldsymbol{z} = (\boldsymbol{z}_1^T, \boldsymbol{z}_2^T)^T$, with associated coupling parameters $\rho_1, \rho_2 > 0$.

(i) **TV prior**: in this case, $p = \mathbb{1}_{\mathbb{R}_+^N}$, $\boldsymbol{H}_2\boldsymbol{x} = ((\eta\boldsymbol{H}\boldsymbol{x})^T, (\boldsymbol{D}\boldsymbol{x})^T)^T$, with $\boldsymbol{D}$ defined as in (1.8), and $f_2(\boldsymbol{z}) = \mathrm{D}_{\mathrm{KL}}(\boldsymbol{y}\|\boldsymbol{z}_1) + \beta\|\boldsymbol{z}_2\|_{2,1}$. AXDA is applied as in (1.26). The variables are updated with PSGLA [Salim et al., 2019] as in [Thouvenin et al., 2024], *i.e.*,

$$\boldsymbol{\xi}^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_N), \tag{2.31}$$

$$\begin{aligned}\boldsymbol{x}^{(t+1)} = \mathrm{proj}_{\mathbb{R}_+^N}\Big(\boldsymbol{x}^{(t)} - \frac{\gamma\eta^2}{\rho_1}\boldsymbol{H}^*\big(\boldsymbol{H}\boldsymbol{x}^{(t)} - \eta^{-1}\boldsymbol{z}_1^{(t)}\big) \\ - \frac{\gamma}{\rho_2}\boldsymbol{D}^*\big(\boldsymbol{D}\boldsymbol{x}^{(t)} - \boldsymbol{z}_2^{(t)}\big) + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)}\Big),\end{aligned} \tag{2.32}$$

$$\boldsymbol{\zeta}_1^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_M), \tag{2.33}$$

$$\boldsymbol{z}_1^{(t+1)} = \mathrm{prox}_{\kappa_1 \mathrm{D}_{\mathrm{KL}}(\boldsymbol{y}\|\cdot)}\Big(\boldsymbol{z}_1^{(t)} - \frac{\kappa_1}{\rho_1}\big(\boldsymbol{z}_1^{(t)} - \eta\boldsymbol{H}\boldsymbol{x}^{(t+1)}\big) + \sqrt{2\kappa_1}\boldsymbol{\zeta}_1^{(t+1)}\Big), \tag{2.34}$$

$$\boldsymbol{\zeta}_2^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_{2N}), \tag{2.35}$$

$$\boldsymbol{z}_2^{(t+1)} = \mathrm{prox}_{\kappa_2\beta\|\cdot\|_{2,1}}\Big(\boldsymbol{z}_2^{(t)} - \frac{\kappa_2}{\rho_2}\big(\boldsymbol{z}_2^{(t)} - \boldsymbol{D}\boldsymbol{x}^{(t+1)}\big) + \sqrt{2\kappa_2}\boldsymbol{\zeta}_2^{(t+1)}\Big) \tag{2.36}$$

(ii) **Denoiser-based prior**: in this case, p is defined as in [Laumont et al., 2022], $\boldsymbol{H}_2\boldsymbol{x} = ((\eta\boldsymbol{H}\boldsymbol{x})^T, \boldsymbol{x}^T)^T$ and $f_2(\boldsymbol{z}) = \mathrm{D}_{\mathrm{KL}}(\boldsymbol{y}\|\boldsymbol{z}_1) + \iota_{\mathbb{R}_+^N}(\boldsymbol{z}_2)$. AXDA is applied as in (1.26), and the posterior distribution is sampled with Algorithm 1. More precisely, (2.23)–(2.24) become

$$\boldsymbol{\xi}^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_N), \tag{2.37}$$

$$\begin{aligned}\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \frac{\gamma\eta^2}{\rho_1}\boldsymbol{H}^*\big(\boldsymbol{H}\boldsymbol{x}^{(t)} - \eta^{-1}\boldsymbol{z}_1^{(t)}\big) + \frac{\alpha\gamma}{\epsilon}\big(D_\epsilon(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big) \\ + \frac{\gamma}{\lambda}\big(\mathrm{proj}_{\mathcal{C}}(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big) - \frac{\gamma}{\rho_2}(\boldsymbol{z}_2^{(t)} - \boldsymbol{x}^{(t+1)}) + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)}\end{aligned} \tag{2.38}$$

$$\boldsymbol{\zeta}_1^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_M), \tag{2.39}$$

$$\boldsymbol{z}_1^{(t+1)} = \mathrm{prox}_{\kappa_1 \mathrm{D}_{\mathrm{KL}}(\boldsymbol{y}\|\cdot)}\Big(\boldsymbol{z}_1^{(t)} - \frac{\kappa_1}{\rho_1}\big(\boldsymbol{z}_1^{(t)} - \eta\boldsymbol{H}\boldsymbol{x}^{(t+1)}\big) + \sqrt{2\kappa_1}\boldsymbol{\zeta}_1^{(t+1)}\Big), \tag{2.40}$$

$$\boldsymbol{\zeta}_2^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_{2N}), \tag{2.41}$$

$$\boldsymbol{z}_2^{(t+1)} = \mathrm{proj}_{\mathbb{R}_+^N}\Big(\boldsymbol{z}_2^{(t+1)} - \frac{\kappa_2}{\rho_2}\big(\boldsymbol{z}_2^{(t)} - \boldsymbol{x}^{(t+1)}\big) + \sqrt{2\kappa_2}\boldsymbol{\zeta}_2^{(t+1)}\Big), \tag{2.42}$$

Remark 3. All these applications satisfy the assumptions stated in Assumption 1. In particular, the masking operator is element-wise, and the convolution and dis-

crete gradient operators are localized, thus satisfying (A3). In both the Gaussian and Poisson noise cases, the data-fidelity terms are separable, hence (A4) holds. Finally, the denoiser-based prior satisfies (A5) since the denoiser is implemented as a CNN, and the TV prior satisfies the same assumption because the indicator function $\mathbb{1}_{\mathbb{R}_+^N}$ is additively separable.

2.5 Conclusion

This chapter investigated how a PnP-based sampler can be efficiently adapted to larger datasets using distributed computing techniques. By exploiting the structure of *localized* operators, the distributed approach enables an efficient SPMD implementation and ensures numerical equivalence with its sequential counterpart. The proposed methodology remains applicable to a broader class of PnP-based MCMC algorithms beyond PnP-ULA, provided that they satisfy similar locality properties. Applications to several standard inverse problems illustrate the interest of the proposed algorithm. Chapter 4 shows the experimental results.

The next chapter is devoted to the design of a software library that implements the proposed distributed sampler in a modular and extensible way. Particular emphasis will be placed on providing users with direct low-level control over communication strategies, a feature often difficult to access in state-of-the-art Bayesian inference software. This implementation effort complements the theoretical contributions of the present chapter with practical tools for efficient computation.

Chapter 3

From distributed algorithms to their implementation

Contents

3.1 Bayesian inference software development 46
3.1.1 Review on Bayesian inference software 46
3.1.2 Design specifications and development choices 49
3.2 Library design principles 52
3.2.1 High-level orchestration 53
3.2.2 Model specification 54
3.2.3 Transition kernels 55
3.2.4 Distributed computing components 55
3.3 Complete example 57
3.3.1 Implementing transition kernels 57
3.3.2 Implementing distributed denoisers 59
3.3.3 Implementing distributed operators 61
3.3.4 Main script 62
3.4 Conclusion 63

This chapter connects the distributed algorithm developed in the previous chapter with the proposed software implementation. The goal is to provide a modular and flexible environment in which a wide variety of samplers can be implemented in an efficient, yet deeply customizable, distributed way.

The chapter begins by situating this work within the broader landscape of Bayesian inference, highlighting the limitations of existing software libraries and

motivating the need for a new framework. The following sections then detail the design principles of the proposed library and illustrate its use for an application from Chapter 2. The development of this library has greatly benefited from the contributions of Stéphane Despierres, engineer at the CRIStAL laboratory from Oct. 2022 to Oct. 2025. A special thanks goes to Damien Marchal and Emmanuel Leguy, research engineers at CRIStAL, who have accompanied the development and provided precious feedback over the implementation. The source code of the proposed software library is available at:

`https://github.com/maxime-bouton/cards`

3.1 Bayesian inference software development

This section aims to motivate the development of the proposed distributed MCMC software framework. By reviewing established libraries across different programming languages, this section highlights both the strengths and the limitations of current solutions, with a particular focus on the following aspects:

- **distributed computing capabilities**: support for execution across multiple devices (CPU, GPU, TPU, ...), with an explicit control over communications.
- **extensibility**: ease of adding new components with limited prior knowledge of the internal code structure or excessive development time;
- **maturity**: time since release and degree of adoption within the community;

Building on these observations, the section then formulates specifications for a distributed MCMC library designed to fill the identified gaps and outlines the resulting development choices.

3.1.1 Review on Bayesian inference software

A wide range of software frameworks has been developed to implement MCMC algorithms in practice [Štrumbelj et al., 2024]. These libraries differ in performance, extensibility, available features, and ease of use. In this overview, which is not meant to be exhaustive, the main tools are grouped by programming language. Frameworks based on low-level languages such as C++ are generally built for high-performance computing (HPC), prioritizing efficiency, scalability, and stability for demanding workloads. In comparison, higher-level languages such as Python, R, or Julia are more tightly integrated into data science and statistical analysis workflows. They prioritize ease of prototyping, user-friendly interfaces,

and flexible integration with machine learning ecosystems, sometimes at a performance cost. A synthesis of the most relevant features, particularly in the context of distributed MCMC algorithms, is provided in Table 3.1.

C++ remains a reference language for performance and backward-compatible stability, but its low-level nature makes it more verbose and demands greater programming expertise. Among its most established statistical modeling tools, `Stan` [Carpenter et al., 2017] provides robust implementations of Hamiltonian Monte Carlo (HMC) [Neal, 2011] and the No-U-Turn Sampler (NUTS) [Homan et al., 2014], together with automatic differentiation, extensive diagnostics, and partial GPU support. Alongside Stan, the **BUGS family** (`OpenBUGS` [Lunn et al., 2009], `JAGS` [Plummer, 2003]) has long been central to applied Bayesian statistics, offering accessible Gibbs samplers for hierarchical models. Their maturity and stability have made them standard choices in many domains, but their limited extensibility restricts algorithmic innovation. Extending this line, `MultiBUGS` [Goudie et al., 2020] addresses scalability by distributing Gibbs updates across multiple processors, making it particularly suited to HPC environments.

Python has become the dominant ecosystem for modern Bayesian inference, thanks to its flexibility, strong community, and integration with machine learning algorithms. General-purpose frameworks such as `PyMC` [Salvatier et al., 2016] offer user-friendly interfaces, mature workflows, and a large ecosystem of visualization and diagnostic tools, making them attractive to a wide audience. Other frameworks place stronger emphasis on deep learning integration: `Pyro` [Bingham et al., 2019] builds directly on PyTorch, while `TensorFlow Probability` (TFP) [Dillon et al., 2017] leverages the TensorFlow ecosystem. Both enable deep probabilistic models and hybrid inference methods combining MCMC with variational inference (VI). Among frameworks optimized for large-scale computation, `NumPyro` [Phan et al., 2019] and `BlackJAX` [Cabezas et al., 2024] exploit JAX's just-in-time compilation and vectorization, achieving scalable sampling across CPUs, GPUs, and TPUs. The former provides a complete probabilistic programming framework, while the latter focuses on modular, reusable samplers. Beyond these general-purpose tools, the Python ecosystem also hosts specialized libraries. For example, `emcee` [Foreman-Mackey et al., 2013] is widely adopted in astrophysics for its efficient ensemble samplers, and `Mici` [Graham, 2025] explores advanced geometric MCMC methods on Riemannian manifolds.

Language	Library	Maturity	Key Algorithms	Parallelization	GPU	Backend
C++	`Stan`	High	HMC, NUTS	Multi-chain	✓	OpenCL
	BUGS family	High	Gibbs	Multi-chain	×	-
Python	`PyMC`	High	HMC, NUTS, VI	✓	✓	PyTensor
	`Pyro`	High	HMC, NUTS, VI	✓	✓	PyTorch
	`NumPyro`	Growing	HMC, NUTS, SMC	✓	✓	JAX
	`BlackJAX`	Research	Modular	✓	✓	JAX
	TFP	High	HMC, VI	✓	✓	TensorFlow
	Specialized	Niche	-	-	-	-
R	`NIMBLE`	Moderate	Gibbs, Custom	Multi-chain	×	C++
Julia	`Turing.jl`	Growing	HMC, NUTS, VI	✓	✓	Julia

Table 3.1: Major Bayesian inference libraries. BUGS family includes OpenBUGS, JAGS, MultiBUGS. Specialized includes emcee (astronomy), Mici (Riemannian HMC). SMC corresponds to Sequential Monte Carlo. **Parallelization** indicates the support for single-MCMC-chain vectorization and multi-chain parallelism.

R remains popular among applied statisticians due to its accessibility and integration with traditional statistical workflows. The most notable framework, `NIMBLE` [de Valpine et al., 2017], extends the BUGS [Gilks et al., 1994] philosophy while offering greater flexibility. It allows users to write custom samplers in a domain-specific language that compiles to C++, combining the accessibility of R with the efficiency of low-level execution.

Julia aims to deliver the performance of low-level languages while maintaining the usability of the high-level ones. Its flagship framework, `Turing.jl` [Ge et al., 2018], provides a flexible and performant probabilistic programming environment that supports both classical and modern inference methods, with growing support for accelerators. Thanks to Julia's design, it offers an attractive compromise between performance and usability, although its ecosystem is still younger than that of Python or R.

In summary, C++ libraries emphasize robustness and long-term adoption, while Python frameworks dominate in terms of flexibility and ecosystem integration. R remains attractive for traditional statistics, and Julia offers a promising compromise between performance and usability. This diversity in libraries reflects a trade-off between maturity, computational capabilities, and extensibility.

In the context of modular, distributed MCMC algorithms, existing frameworks generally lack straightforward and explicit control over two key aspects: (i) the internal structure of the sampler, such as easily composable transition kernels and block decompositions, and (ii) the communication mechanisms needed to run a single MCMC chain across multiple devices. Typically, distributed execution

is limited to independent multi-chain runs, or relies on automatic device-level parallelization provided by the underlying library, *e.g.*, `PyMC`, `NumPyro`. As a result, implementing custom communication schemes for single-chain updates usually requires complex workarounds or deep modifications of the library code. Notably, `BlackJAX` stands out by offering explicit control over communication schemes. However, it lacks high-level structural abstractions that would allow non-experts to build custom distributed workflows without delving into the library internals. This limitation motivates the design of a framework in which both the sampler structure and the communication schemes are explicit and easy to customize, and where `JAX` could serve as a promising backend.

3.1.2 Design specifications and development choices

Despite the wide range of available Bayesian inference software libraries, none simultaneously meet all the requirements needed for the seamless implementation of modular, distributed MCMC algorithms such as those introduced in Chapter 2. The following paragraphs establish some specifications that the new library should meet to facilitate this integration and translate them into concrete implementation choices leveraging high-performance computing. These specifications and design choices aim to make distributed sampling accessible to anyone, especially researchers and practitioners without HPC expertise.

Specifications for a modular distributed MCMC library

Three fundamental capabilities define the requirements for addressing current software limitations: modular model construction, extensibility towards advanced methodologies, and scalability through distributed computation.

Modularity: The library should enable users to compose complex inference schemes from elementary building blocks such as transition kernels, operators, and variable splitting schemes. This modular approach ensures that algorithms can be constructed, adapted, and compared without being constrained by overly rigid model specifications, allowing researchers to focus on algorithmic innovation instead of implementation details.

Extensibility: The framework must enable the addition of diverse modeling components, including traditional parametric priors and modern plug-and-play techniques. It should also allow the creation of multiple algorithmic strategies for

the same model, such as data augmentation schemes or transition kernel composition for different Gibbs sampler variants. New components should be easy to integrate, requiring minimal knowledge of the internal code structure and little development effort. By combining flexibility with accessibility, the framework should consolidate capabilities that are currently scattered across specialized libraries, facilitating rapid experimentation with new models and algorithmic variants.

Scalability: The library should natively support running single MCMC chains across multiple workers, typically using GPUs when leveraging deep learning frameworks. At the same time, it should remain compatible with sequential execution modes. This design principle is crucial for scaling inference to high-dimensional problems such as imaging applications.

These three guidelines define the specifications for a library that would allow for an efficient and user-friendly implementation of a wide range of distributed MCMC algorithms.

Development choices

Given the specifications outlined above, an appropriate programming environment should combine ease of prototyping, access to advanced numerical and machine learning tools, and the ability to scale efficiently across heterogeneous hardware. Among the available options, Python stands out as a natural candidate. Its popularity in the data science and machine learning communities has fostered a large and active ecosystem, where libraries are well maintained, extensively documented, and widely adopted. In the signal processing community, Python has become the standard environment for research and prototyping, gradually replacing earlier MATLAB-based workflows.

Python's high level syntax allows for concise expression of complex workflows, while compatible with low-level languages (C, C++, CUDA) for performance-critical components. This dual capacity of rapid prototyping and seamless access to high-performance routines matches the requirements for a modular and scalable library. Several mature tools already exist in the Python language ecosystem to meet these requirements. Those selected for the proposed library are detailed in the following paragraphs.

`CuPy` for GPU computing: `CuPy` [Okuta et al., 2017] is a drop-in replacement for NumPy that runs computations on CUDA-enabled GPUs. By simply replacing `numpy` with `cupy`, existing NumPy code can be executed on the GPU without

further modification. The library mirrors most of the NumPy API, including array creation, broadcasting, indexing, and mathematical operations, which makes the transition from CPU to GPU seamless (see example in Listing 1). Internally, `CuPy` executes operations through *CUDA kernels* and NVIDIA's optimized libraries such as cuBLAS and cuFFT, while keeping the same interface as NumPy.

```python
import numpy as np
import cupy as cp

# NumPy: CPU arrays
x_np = np.random.rand((2,3,64,64))
l21_np = np.sum(np.sqrt(np.sum(x_np**2, axis=0)))

# CuPy: GPU arrays (same API)
x_cp = cp.random.rand((2,3,64,64))
l21_cp = cp.sum(cp.sqrt(cp.sum(x_cp**2, axis=0)))

# from CPU to GPU
x_cp = cp.asarray(x_np)

# from GPU to CPU, `cp.asnumpy(x_cp)` also works
x_np = x_cp.get()
```

Listing 1: Comparison example of NumPy and CuPy APIs.

`mpi4py` for distributed computing: `mpi4py` [Dalcin et al., 2021] is a Python wrapper to the *Message Passing Interface* (MPI) standard, enabling communication between processes in a distributed system. Several MPI implementations exist, such as `MPICH` and `OpenMPI`, each providing its own runtime and optimizations. Notably, `OpenMPI` supports CUDA-aware communication through the `UCX` framework [Shamis et al., 2015], enabling processes to transfer data directly between GPUs without intermediate copies on host memory. This makes `mpi4py` particularly relevant for multi-GPU or multi-node setups, where minimizing communication overhead is essential to scale MCMC across large computing infrastructures.

`PyTorch` as a framework for Deep Learning and auto-differentiation: Although primarily known as a deep learning framework, `PyTorch` [Ansel et al., 2024] provides a versatile set of tools for GPU-accelerated tensor computation and *automatic differentiation* [Baydin et al., 2017], which are highly relevant for MCMC methods that integrate learning-based priors or gradient-informed proposals (e.g.

based on Langevin-derived transition kernels). It offers native CUDA support for tensor operations, neural network layers, and custom CUDA kernels, enabling efficient execution on multi-GPU systems. Through interoperability with CuPy, specialized numerical routines can be seamlessly integrated into deep learning workflows. PyTorch tensors and CuPy arrays can be shared via implicit CUDA array interfaces without copying data between CPU and GPU memory (see Listing 2). This allows custom numerical code (CuPy) and deep learning models (PyTorch) to coexist in a unified pipeline while avoiding costly memory transfers.

```python
import torch
import cupy as cp
cp.cuda.Device(1).use() # bind cupy to second GPU
x_cp = cp.ones(10, dtype=cp.float32) # created directly on GPU 1

# Zero-copy conversion to a PyTorch tensor (still on GPU 1)
x_torch = torch.as_tensor(x_cp)

# Convert back to CuPy with zero-copy
x_cp_back = cp.asarray(x_torch)
```

Listing 2: Interoperability between CuPy and PyTorch. The example demonstrates zero-copy conversion between CuPy arrays and PyTorch tensors on the same GPU, avoiding unnecessary data transfers.

These libraries demonstrate that Python already provides the technological backbone required for an efficient distributed MCMC framework: `CuPy` ensures GPU-accelerated numerical computations, `mpi4py` enables distributed execution across multiple processes and nodes, `PyTorch` integrates seamlessly with deep learning approaches. Practical considerations to run Python scripts on multi-GPU systems are discussed in Appendix A. Building on these tools and these library specifications, the next section presents the design principles that guided the development of the proposed library.

3.2 Library design principles

The library design is guided by three principles: *modularity*, so that the sampling of any posterior distribution of the form (1.3) can be handled by combining elementary components, including when splitting mechanisms are involved; *extensibility*, to allow for new forward models, regularization techniques, or sampling strategies to be incorporated with minimal effort; and *scalability*, with a particular

emphasis on distributed execution. The software architecture directly implements the mathematical structure of MCMC algorithms, as illustrated in Algorithm 2. In this algorithm, $\boldsymbol{X}$ represents possibly one or more latent variables being sampled and the transition kernel K_θ may depend on fixed parameters θ.

Algorithm 2: Generic MCMC Sampler.

1 Load observations $\boldsymbol{y}$;
2 Initialize sampling variables $\boldsymbol{X}^{(0)} \sim p(\boldsymbol{X})$;
3 **for** $t \leftarrow 0$ **to** $T-1$ **do**
4 $\quad \boldsymbol{X}^{(t+1)} \sim K_\theta(\boldsymbol{X}^{(t)}, \cdot)$;
Output: samples $\left(\boldsymbol{X}^{(t)}\right)_{t=1}^{T}$;

The design, as outlined in Figure 3.1, organizes the library into four distinct architectural layers to address the mathematical components of Algorithm 2. The role of the layers is described in the following paragraphs.

3.2.1 High-level orchestration

The mathematical structure of the MCMC sampling loop (lines 3–4 in Algorithm 2) requires orchestration of data loading (line 1), variable initialization (line 2), and sample collection (line 4). This orchestration is implemented through three interconnected classes that manage the execution flow.

The `Sampler` class directly implements the generic sampling loop structure from Algorithm 2, coordinating timing and checkpointing (i.e. saving intermediate results periodically) while delegating problem-specific updates to an associated `Model`. It gathers all attributes common to any MCMC algorithm (e.g., number of iterations T, burn-in period, random seed) and defines the sampling process structure corresponding to lines 1–4.

The `DataManager` centralizes and abstracts file input/output operations, such as in line 1. In a distributed setting, it loads the observation blocks $\boldsymbol{y}_b$ from disk and saves the corresponding estimate blocks $\hat{\boldsymbol{x}}_b$ to disk. This component ensures proper data flow throughout the sampling process.

Together with the `Model` detailed in the next section, these components constitute the orchestration layer that transforms the abstract mathematical algorithm into executable code, while maintaining a clear separation between execution management, inference logic, and data handling.

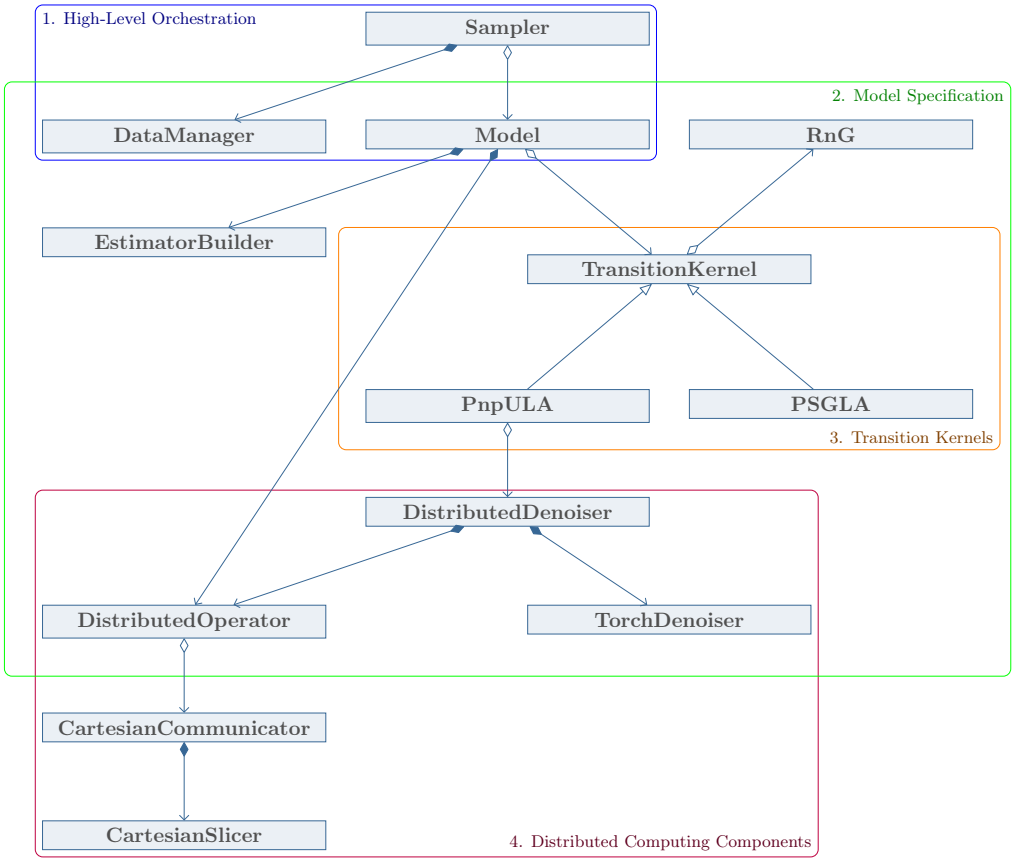

Figure 3.1: Simplified class diagram of the proposed library. A solid diamond indicates composition, meaning that one class is made up of another and cannot exist without it. An open diamond represents aggregation, where one class references another but both can exist independently. An arrow denotes inheritance, showing that one class extends another.

3.2.2 Model specification

The core of any MCMC algorithm lies in the definition of the transition kernel $K_\theta(\boldsymbol{X}^{(t)}, \cdot)$ from line 4 and the underlying probabilistic model. The `Model` class encapsulates this mathematical structure by combining three fundamental components that directly correspond to the mathematical objects in Bayesian inference:

- The `TransitionKernel` is the building block to encode K_θ, governing the proposal mechanism in line 4. Multiple instances of this class can be combined to define the sampling dynamics of complex models involving multiple latent variables, e.g. SGS models. If applicable, this object can also implement an accept/reject step.

- The `DistributedOperator` represents any operator within the `Model`. It is especially needed to compute the generic transition encoded in the `TransitionKernel` instances and its implementation is distributed following a pattern specified by the user. For instance, the linear operators $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ defined in Chapter 2 are represented as instances of this class.
- The `EstimatorBuilder` aggregates the sample sequence $\left(\boldsymbol{X}^{(t)}\right)_{t=1}^{T}$ from line 4 and creates auxiliary quantities to form meaningful posterior estimators. This component implements several methods for aggregating statistical information. One approach updates estimates continuously as new samples are generated, which can be used for MMSE or pixel variance computation. Alternatively, it can retain all samples, enabling the offline construction of credibility intervals that cannot be computed incrementally.

3.2.3 Transition kernels

The mathematical foundation of MCMC sampling relies on transition kernels $K_\theta(\boldsymbol{X}^{(t)}, \cdot)$ that define the update step in line 4. The `TransitionKernel` interface provides a generic implementation of this mathematical concept, with specializations for different algorithmic approaches.

For example, the library currently includes implementations of Langevin-based kernels such as `PnpULA` (Plug-and-Play Unadjusted Langevin Algorithm) and `PSGLA` (Proximal Stochastic Gradient Langevin Algorithm). Each kernel type corresponds to different mathematical formulations of the transition operator, either based on plug-and-play priors or on proximal-gradient techniques.

Each `TransitionKernel` instance stores the current state $\boldsymbol{X}^{(t)}$ and implements the update state to obtain $\boldsymbol{X}^{(t+1)}$ from $\boldsymbol{X}^{(t)}$ according to line 4.

The `RNG` class for random number generation ensures that the stochastic aspects of the mathematical kernel K_θ are properly handled, maintaining reproducibility and statistical validity of the sampling process. Further details related to reproducibility and random number generation are reported in Appendix B.

3.2.4 Distributed computing components

The distributed computing layer implements the decomposition strategies that enable parallel computation. These elements are key to the distributed implementation of operations involved in the algorithm.

The `DistributedOperator` abstracts mathematical operators that can be decomposed across multiple processes, combining local computations with inter-

process communication. This abstraction covers both operators requiring communication (e.g., distributed convolutions), and embarrassingly parallel operations (e.g., pointwise transformations). Its implementation relies on two auxiliary classes: `CartesianCommunicator` and `CartesianSlicer`.

- The `CartesianCommunicator` manages *how* data is exchanged between processes. It handles the mathematical topology of the MPI process grid and allocates the appropriate buffers and MPI types. In particular, it implements the communication patterns required for applying the underlying `DistributedOperator` and stores the buffer associated with the extended variable $\boldsymbol{S}_b\boldsymbol{x}$ (see Section 2.2 and Figure 2.2 for details).
- The `CartesianSlicer` defines *what* portions of the data needs to be exchanged. It implements the domain decomposition strategy, identifying the precise data slices and index ranges that each process must handle when applying the `DistributedOperator`.

These distributed components collectively ensure that the mathematical structure of the MCMC algorithm from Algorithm 2 can be executed efficiently in a distributed manner. In particular, all spatial indexing and communication modalities are precomputed during the initialization of the `DistributedOperator` object, which contains `CartesianSlicer` and `CartesianCommunicator` instances. Concretely, the `CartesianSlicer` computes and exposes the slice objects and sizes that describe the placement of the local block $\boldsymbol{x}_b$ within the extended buffer $\boldsymbol{S}_b\boldsymbol{x}$, and the indices of the valid sub-block of the output $\boldsymbol{W}_b\boldsymbol{S}_b\boldsymbol{x}$. These slice descriptors are purely geometric (no MPI logic) and are then consumed by the `CartesianCommunicator`, which builds the actual MPI subarray datatypes, determines sender/receiver ranks, and preallocates the send/receive requests used at runtime.

Complex operators can be composed from multiple `DistributedOperator` instances. For example, the `DistributedDenoiser` combines several such operators to provide a distributed implementation of a denoising neural network.

Remark 4. Although the entire `Sampler` implementation relies on a single partition of a given latent variable (e.g. $\boldsymbol{x} = (\boldsymbol{x}_b)_b$), the need for applying different distributed operators to this variable may require different overlapping regions (i.e. different families of operators $\boldsymbol{S}_b$). In other words, the overlapping buffer (e.g. $\boldsymbol{S}_b\boldsymbol{x}$) generally depends on the specific operator being applied. As a result, multiple temporary buffers may coexist, one for each distinct overlap pattern,

which raises important considerations for memory efficiency. This is something one wants to avoid.

To illustrate, with B workers, the local block $\boldsymbol{x}_b$ is of size $N_b \approx N/B$, while its extended version $\boldsymbol{S}_b\boldsymbol{x}$ has a size $\widetilde{N}_b$ slightly larger than N_b, to account for the overlap with neighboring processes. When B is small (typically $B = 2$), the cumulative memory on a worker can potentially exceed N, meaning that each process effectively manipulates more data than in a non-distributed setting. This overhead is the price paid for the modularity of each distributed operator to be applied independently.

A more memory-efficient alternative, when multiple operators with different overlaps are involved, is to allocate a single buffer large enough to accommodate the union of all required overlapping regions. This avoids maintaining several copies of buffers meant only for communication purposes and thereby reduce redundant memory usage.

Taken together, these four layers illustrate how the library translates the mathematical structure of Algorithm 2 into a modular and scalable software architecture. From high-level orchestration managing the sampling loop to distributed components enabling parallel execution, each class plays a well-defined role corresponding to a specific functionality in a generic MCMC algorithm. The following section presents a concrete example that demonstrates how these architectural concepts are instantiated in practice.

3.3 Complete example

Using this library first calls for the definition of a class derived from the generic `Model`, which encodes the transition kernels and their interactions. This section illustrates the approach through the distributed implementation of the Split-Gibbs-Sampling (SGS) algorithm, Algorithm 1, for image deconvolution under Poisson noise with a plug-and-play prior, as described in Section 2.4.

3.3.1 Implementing transition kernels

This section details the particular implementations of the transition kernels used in the given application: PSGLA and PnP-ULA.

PSGLA updates

```
X.prox = prox_nonegativity # function projecting onto R_+
X.grad = lambda x: eta**2 * H.adjoint(Hx - Z1.state / eta) / rho_1 + (x
↪  - Z2.state) / rho_2

Z1.prox = lambda z1: prox_KL(z1, y, lam=Z1.step_size)
Z1.grad = lambda z1: (z1 - eta * Hx) / rho_1
```

Listing 3: Definition of the proximal and gradient operators for the PSGLA transition kernels within a `Model` instance. Note that the `TransitionKernel` instances `X` and `Z1`, and all the parameters are attributes of the `Model` class.

In the considered application, the variables $\boldsymbol{x}$ and $\boldsymbol{z}_1$ are updated following

$$\begin{aligned}\boldsymbol{x}^{(t+1)} = \mathrm{proj}_{\mathbb{R}_+}\Bigg(&\boldsymbol{x}^{(t)} + \sqrt{2\gamma}\,\boldsymbol{\xi}^{(t+1)} \\ &- \gamma\left[\frac{\eta^2}{\rho_1}\boldsymbol{H}^*\left(\boldsymbol{H}\boldsymbol{x}^{(t)} - \frac{1}{\eta}{\boldsymbol{z}_1}^{(t)}\right) + \frac{1}{\rho_2}\left(\boldsymbol{x}^{(t)} - {\boldsymbol{z}_2}^{(t)}\right)\right]\Bigg),\end{aligned} \tag{3.1}$$

$${\boldsymbol{z}_1}^{(t+1)} = \mathrm{prox}_{\kappa\,\mathrm{D_{KL}}}\left({\boldsymbol{z}_1}^{(t)} - \frac{\kappa}{\rho_1}\left({\boldsymbol{z}_1}^{(t)} - \eta\,\boldsymbol{H}\boldsymbol{x}^{(t+1)}\right) + \sqrt{2\kappa}\,\boldsymbol{\zeta}_1^{(t+1)}\right). \tag{3.2}$$

The kernel relies on a core method, `mc_update`, based on two auxiliary routines: `prox` and `grad`, which realize the proximal and gradient operators of PSGLA. The corresponding implementation is provided in Listing 3.

PnP-ULA updates

```
Z2.denoise = lambda z2: denoiser(z2, Z2.epsilon**0.5)
Z2.grad = lambda z2: (z2 - X.state) / rho_2
Z2.project = lambda z2: z2.clip(-1, 2)
```

Listing 4: Definition of the proximal and gradient operators for the PnP-ULA transition kernel related to $\boldsymbol{z}_2$.

The PnP-ULA kernel involves the update of $\boldsymbol{z}_2$ given by

$$\begin{aligned}{\boldsymbol{z}_2}^{(t+1)} = {\boldsymbol{z}_2}^{(t)} - \frac{\gamma}{\rho_2}\left({\boldsymbol{z}_2}^{(t)} - \boldsymbol{x}^{(t)}\right) + \sqrt{2\gamma}\,\boldsymbol{\zeta}_2^{(t+1)} \\ + \frac{\alpha\gamma}{\epsilon^2}\left(D_\epsilon({\boldsymbol{z}_2}^{(t)}) - {\boldsymbol{z}_2}^{(t)}\right) + \frac{\gamma}{\lambda}\left(\Pi_{\mathcal{C}}({\boldsymbol{z}_2}^{(t)}) - {\boldsymbol{z}_2}^{(t)}\right).\end{aligned} \tag{3.3}$$

Its implementation requires defining `grad`, as well as the `denoise` and `proj`

```
1  class DnCNN(torch.nn.Module):
2      def __init__(self, C, P):
3          super().__init__()
4
5          m_head = conv(C, P, mode="CR", bias=True)
6          m_body = [conv(P, P, mode="CR", bias=True) for _ in range(18)]
7          m_tail = conv(P, C, mode="C", bias=True)
8
9          self.model = sequential(m_head, *m_body, m_tail)
10
11     def forward(self, x):
12         n = self.model(x)
13         return x - n
```

Listing 5: Standard DnCNN implementation. Functions `conv` and `sequential` facilitate the creation of custom convolutional layers and sequential models, respectively. The mode `CR` indicates a convolutional layer followed by a ReLU activation.

methods corresponding to the denoising and projection steps. The `denoise` routine makes use of a `DistributedDenoiser`, described in the following section. The implementation is given in Listing 4.

3.3.2 Implementing distributed denoisers

In the proposed implementation, distributing a pre-trained denoiser amounts to decomposing its architecture and layers into elementary distributed operators. This decomposition not only makes the approach conceptually transparent, but also facilitates its extension to other neural architectures. For illustrative purposes, the methodology is demonstrated on the architecture of DnCNN, see Example 2, which is the simplest convolutional denoiser considered in this work.

The specific DnCNN configuration employed is provided in Listing 5. Its structure is organized into three submodules: `m_head`, `m_body`, and `m_tail` (lines 5 to 7). The head consists of the initial convolutional layer and a ReLU activation, which maps the input image into a higher-dimensional feature representation. The body is composed of a sequence of convolutional layers, eighteen in total, each followed by a ReLU activation, progressively refining the feature maps. Finally, the tail convolutional layer projects the processed features back into the image domain to recover the denoised output. In total, the model comprises twenty convolution steps, with the last layer deliberately left without activation to allow direct reconstruction in the pixel space. This aggregated `model` object outputs

```python
class MpiDnCNN(DistributedDenoiser):
    def __init__(self, comm, grid_size, N):
        self.dncnn = load_pretrained_dncnn()
        kernel_size = self.dncnn.model[0].kernel_size
        padding = self.dncnn.model[0].padding
        P = self.dncnn.model[0].out_channels
        # define size of full data in the feature space
        core_size = ...
        # define input partitioning of the corresponding layer(s)
        tile_range = ...

        self.edge_conv = MpiTorchConvolution(
            N,
            kernel_size,
            padding,
            comm,
            grid_size,
            backward=False,
            Cout=P,
        )

        self.core_conv = MpiTorchConvolution(
            core_size,
            kernel_size,
            padding,
            comm,
            grid_size,
            backward=True,
            tile_range=tile_range,
        )

    def _apply_2_layers(self, u_b, W_k: Conv2d, W_l: Conv2d):
        u_b = self.core_conv.forward(u_b, W_k).clip(min=0)
        return self.core_conv.backward(u_b, W_l).clip(min=0)

    def __call__(self, x_b, sigma):
        W = self.dncnn.model[0]
        u_b = self.edge_conv.forward(x_b, W).clip(min=0)
        for i in range(len(self.dncnn.model[2:-2:4])):
            conv_forward = self.dncnn.model[2 + 4 * i]
            conv_backward = self.dncnn.model[2 + 4 * i + 2]
            u_b = self._apply_2_layers(u_b, conv_forward, conv_backward)

        return x_b - self.edge_conv.backward(u_b, self.dncnn.model[-1])
```

Listing 6: Proposed Distributed DnCNN implementation.

the noise contained in the input and subtracts it to produce the denoised image.

The distributed version of this model is provided in Listing 6. In addition to the pre-trained DnCNN instance (a standard `torch.nn.Module`), two `MpiTorchConvolution` instances, `edge_conv` and `core_conv` (lines 12 and 22), handle the distributed execution of convolutional layers. These operators manage the buffers corresponding to the overlapping regions $\boldsymbol{S}_b\boldsymbol{x}$ and orchestrate the MPI communication required to exchange messages between workers. A detailed explanation about how these objects operate is provided in Section 3.3.3. The method `_apply_2_layers`, in line 32, applies two consecutive convolutional layers, and the method `__call__`, in line 36, is the equivalent of the forward pass in standard PyTorch models, *i.e.*, it processes the input through the entire network.

Each `DistributedOperator` provides both a forward method and a so-called `backward` method. The latter is primarily intended to implement the mathematical adjoint of a linear operator, but in practice, it can also be used to reverse communication direction to maintain balanced workloads between workers (see the end of Section 2.2). This convention guarantees that after two consecutive convolution steps, the partitioning of the data remains unchanged. Since all convolutions in DnCNN share the same characteristics (kernel size 3×3, stride 1, and padding 1), once the buffers and communication schemes are defined, they can be reused across all `core_conv` layers by simply providing the appropriate kernel weights.

3.3.3 Implementing distributed operators

This section describes the interaction of the core libraries (`CuPy`, `PyTorch`, and `mpi4py`) in the implementation of distributed operators. It also details the internal dataflow, with a focus on buffer management and the storage of intermediate results. As an example, consider the application of the first convolution layer in the distributed DnCNN (see Listing 6, line 38), illustrated in Figure 3.2. The following explanation builds upon notations and concepts introduced in Chapter 2.

Each process begins with its local portion of the input image $\boldsymbol{x}_b$, stored as a `CuPy` array named `x_b` (step (a)). Inside the `MpiTorchConvolution` object, this local block is copied into a larger buffer `S_bx`, which includes placeholders for overlapping regions shared with neighboring processes (step (b)). At this stage, inter-process communication exchanges the overlapping data using `mpi4py` routines, so that each process has all the pixels needed by the convolution.

After communication, `S_bx` is converted into a `PyTorch` tensor and passed

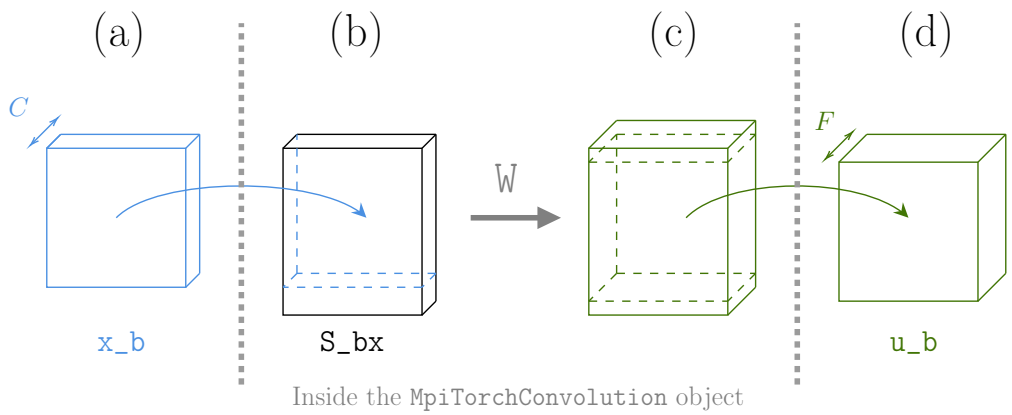

Figure 3.2: Illustration of the distributed convolution in DnCNN. The left part shows the input buffer handled by the current process, the right part displays the corresponding output, and the center part details the forward pass within the `MpiTorchConvolution`, including buffer extension, inter-process communication, and local convolution. Solid shapes represent actual data buffers while dashed inner lines indicate slices or views of these buffers.

through the convolution layer `W`, an instance of `torch.nn.Conv2d` (step (c)). The result is then converted back into a `CuPy` array. The valid sub-block corresponding to the partition $[\boldsymbol{W}\boldsymbol{x}]_b$ is extracted and stored in `u_b` (step (d)). This sub-block serves as the input for the next layer, after applying the ReLU nonlinearity.

3.3.4 Main script

Listing 7 provides a simplified script for running the proposed distributed sampling algorithm. The MPI environment is initialized at line 1, and the global image size and local block dimensions are defined at lines 2-6. Observations are loaded in a distributed manner at line 5, and a distributed DnCNN denoiser is instantiated at line 8.

All kernels involved in the Split-Gibbs Sampler are then created: `PSGLA` for $\boldsymbol{x}$ and $\boldsymbol{z}_1$ (lines 10 and 11), and `PnP-ULA` for $\boldsymbol{z}_2$ (line 12), each configured with its respective local block size and step size. These components are combined into a `MpiPoissonDeconvolutionPnpModel` at lines 15-21, which encapsulates the distributed operators, kernels, and denoiser. Inside this model, the code from Listings 3 and 4 is executed during its initialization.

Finally, a `Sampler` is initialized at line 24 with the model and sampling parameters, and the distributed sampling procedure is executed with the `sample()` method at line 25.

```python
comm = MPI.COMM_WORLD
grid_size = ... # define MPI grid dimensions
N = ... # define global image size
N_b = ... # define local image block size
y = load_observations(comm)
M_b = ... # define local observation block size

denoiser = MpiDnCNN(N, comm, grid_size)

X = PSGLA(N_b, step_size_X)
Z1 = PSGLA(M_b, step_size_Z1)
Z2 = PnpULA(N_b, step_size_Z2, params_Z2)

model = MpiPoissonDeconvolutionPnpModel(
    N,
    comm,
    grid_size,
    X,
    Z1,
    Z2,
    denoiser
)

sampler = Sampler(model, comm, params_sampler)
sampler.sample()
```

Listing 7: Simplified script to run the proposed distributed sampling algorithm.

This main script serves as a concrete demonstration of the modular design of the library. It shows how all the previously introduced building blocks are assembled into a single, operational distributed sampler. The example highlights the clear separation of concerns: each component manages its own data, buffers, and communication, while the `Model` and `Sampler` orchestrate their interactions. In this way, the abstractions developed throughout the chapter are instantiated in practice, providing a scalable and flexible framework for distributed inference.

3.4 Conclusion

Overall, this chapter related the theoretical aspects of Chapter 2 to their practical implementation. The review of existing MCMC frameworks shows that none of the software libraries provides straightforward and explicit control over both the sampler's structure and the communication scheme required to distribute a single MCMC chain, particularly in multi-GPU settings.

This chapter has presented a library that directly addresses this limitation

by mapping building blocks of MCMC algorithms onto modular software classes. By organizing the library into a small set of well-defined layers that reflects the mathematical structure of an MCMC algorithm, such as transition kernels, model specification, orchestration, and distributed operators, the design achieves both computational efficiency and conceptual clarity. A complete implementation illustrated how these components interact in practice, demonstrating that complex distributed computations can be handled in a consistent and scalable manner. This example highlights the flexibility of the framework, showing how abstract MCMC principles can be instantiated into practical and reproducible workflows.

The next chapter evaluates the practical relevance of the proposed approach on a range of imaging inverse problems, with an emphasis on both performance and scalability.

Chapter 4

Application to imaging using distributed deep denoisers

Contents

4.1 Experimental setup 66
4.1.1 Computational Environment and Resources 66
4.1.2 Evaluation metrics 67
4.2 Experimental settings 68
4.2.1 Denoiser benchmarking 68
4.2.2 Distributed sampler assessment 69
4.3 Experimental results 70
4.3.1 Denoising performance 71
4.3.2 Reconstruction quality 72
4.3.3 Scalability experiments 73
4.4 Conclusion 79

This chapter presents a comprehensive evaluation of the proposed distributed methodology introduced in Chapter 2 and implemented in Chapter 3. The analysis is conducted in two stages. First, the denoisers are benchmarked independently to assess their intrinsic restoration quality, computational requirements, and scalability. Second, the distributed sampler Algorithm 1 is evaluated across several large-scale imaging applications, with a focus on both reconstruction accuracy and parallel efficiency. Together, these experiments provide a thorough assessment of the trade-offs between denoising quality and scalability of the proposed distributed approach.

4.1 Experimental setup

This section provides the preliminary information required to describe and assess the experiments presented in the following sections. It details the computational environment and introduces the different image datasets involved, the denoising models considered, as well as the metrics used to quantify reconstruction quality and parallel performance.

4.1.1 Computational Environment and Resources

Hardware and software environment: All the experiments have been conducted on the CNRS supercomputer Jean Zay, hosted by GENCI at IDRIS. Algorithms were run on computing nodes equipped with two 2.5 GHz, 20-core Intel Cascade Lake 6248 processors, and four Nvidia Tesla V100 SXM2 GPUs with 32 GB of memory each. The corresponding implementation is available at `https://github.com/maxime-bouton/cards`, and the environment relies on Python 3.13 with `mpi4py` 4.1.*, `cupy` 13.5.*, and `torch` 2.7.*.

Datasets details: The experiments rely on several image datasets. High-resolution natural images are taken from `https://pixabay.com`. The DIV2K dataset [Agustsson et al., 2017] provides a large collection of high-quality images, while the ImageNet test set [Russakovsky et al., 2015] offers a diverse source of natural image content. Pixel values are normalized to the range $[0, 1]$.

Denoisers presentation: Three different architectures of deep denoisers have been considered in these experiments: DnCNN [K. Zhang et al., 2017], DRUNet [K. Zhang et al., 2021] and DDFB [Le et al., 2022, 2024; Repetti et al., 2022]. Their nature and structures are detailed in Section 2.2. Both DnCNN and DRUNet are on-the-shelf pre-trained denoisers, whose weights and implementation are available at `https://github.com/cszn/KAIR`. The DnCNN network contains $K = 20$ layers, each with $F = 64$ filters. The DRUNet network considered has been trained with $I = 3$ downsampling/upsampling layers, $J = 4$ blocks per residual layer, and $F = 64$ features. Two instances of the DDFB network have been used, containing respectively $K = 4$ and $K = 20$ layers, with $F = 64$ latent features. As in [Le et al., 2024], DDFB has been trained on random patches of size 50×50 from the ImageNet test set [Russakovsky et al., 2015], corrupted by additive white Gaussian noise with standard deviation ϵ ranging from 0 to 0.1. The ℓ_2-norm loss function in [Le et al., 2024] has been replaced by an ℓ_1-norm

Table 4.1: Denoisers' characteristics: number of parameters, memory footprint and Lipschitz constant L_D for $\boldsymbol{I}_n - D_\epsilon$.

Denoiser	**#Params**	**Mem. (MiB)**	L_D
DDFB ($K = 4$)	6 912	0.03	2
DDFB ($K = 20$)	34 560	0.13	2
DnCNN	668 227	2.55	3
DRUNet	32 640 960	124.52	7

for better training stability. Training has been carried out over 100 epochs using the Adam optimizer [Kingma et al., 2017], with a learning rate of 10^{-3}, a weight decay of 10^{-4} and a batch size of 1000. The Lipschitz constant L_D of $\boldsymbol{I}_N - D_\epsilon$ has been estimated for each denoiser as in [Le et al., 2024]. A summary of the denoisers' main characteristics is provided in Table 4.1.

4.1.2 Evaluation metrics

Reconstruction quality metrics: Reconstruction quality is evaluated using three complementary metrics that compare an estimate $\widehat{\boldsymbol{x}}$ to the ground truth $\overline{\boldsymbol{x}}$.

The peak signal-to-noise ratio (PSNR) [Zhou Wang et al., 2009] measures distortion based on the mean squared error (MSE), and is defined as

$$\mathrm{PSNR}(\overline{\boldsymbol{x}}, \widehat{\boldsymbol{x}}) = 10 \log_{10} \left(\frac{I^2}{\mathrm{MSE}(\overline{\boldsymbol{x}}, \widehat{\boldsymbol{x}})} \right). \tag{4.1}$$

where I is the maximum intensity a pixel can take (e.g., 255 for 8-bit images).

The reconstruction signal-to-noise ratio (rSNR) normalizes the error with respect to the signal energy and is given by

$$\mathrm{rSNR}(\overline{\boldsymbol{x}}, \widehat{\boldsymbol{x}}) = 10 \log_{10} \left(\frac{\|\overline{\boldsymbol{x}}\|_2^2}{\|\overline{\boldsymbol{x}} - \widehat{\boldsymbol{x}}\|_2^2} \right), \tag{4.2}$$

The structural similarity index (SSIM) [Z. Wang et al., 2004] accounts for perceptual quality through luminance, contrast, and structural comparison. It ranges from -1 to 1, with higher values indicating greater similarity.

Together, these metrics assess reconstruction quality in terms of pixel fidelity, relative error, and perceptual similarity.

Scalability metrics: Scalability is evaluated through standard parallel performance metrics [Grama et al., 2003]. *Strong scaling* measures how the computation

time decreases as the number of workers B increases for a fixed problem size, while *weak scaling* measures how the computation time changes when both the problem size and the number of workers increase proportionally.

For *strong scaling*, the *speed-up* for B workers is defined as

$$\text{Speedup}(B) = \frac{T_1^s}{T_B^s}, \tag{4.3}$$

where T_1^s and T_B^s denote the execution times with one and B units for a fixed problem size, respectively. Perfect strong scaling corresponds to $\text{Speedup}(B) = B$, when the execution time decreases linearly with the number of workers.

For *weak scaling*, the *efficiency* is defined as

$$\text{Efficiency}(B) = \frac{T_1^w}{T_B^w}, \tag{4.4}$$

where T_1^w is the execution time for the base problem size with a single worker, and T_B^w is the execution time for a problem that is B times larger using B workers. Perfect weak scaling corresponds to $\text{Efficiency}(B) = 1$, indicating that the execution time remains constant as both the problem size and number of workers scale proportionally.

4.2 Experimental settings

Two sets of experiments are conducted. The first set benchmarks the denoisers independently of the sampler in Section 4.2.1. The second one, described in Section 4.2.2, evaluates the proposed distributed sampling Algorithm 1 across the imaging applications presented in Section 2.4. In both cases, reconstruction quality and scalability are assessed.

In all experiments, each MPI process, or worker, is assigned to a dedicated GPU. In the remainder, the terms *worker* and *GPU* are used interchangeably.

4.2.1 Denoiser benchmarking

The denoisers are first evaluated independently of the sampling algorithm.

Denoising performance: Intrinsic restoration quality is assessed on 1000 randomly selected 50×50 patches from the DIV2K dataset [Agustsson et al., 2017], degraded with Gaussian noise corresponding to an input SNR of 20 dB. Corresponding results are reported in Section 4.3.1.

Scalability: Scalability is analyzed from both a theoretical and an empirical perspective. Theoretical figures include the approximate number of floating-point operations (FLOPs) and the total communication volume required by each denoiser, reported in Table 4.4. These quantities provide a first estimate of the computation-to-communication trade-off.

Empirical runtimes are measured for both the standard implementation ($B = 1$) and the distributed versions of the networks ($B \in \{2, 4\}$) described in Section 2.2. Each configuration is tested on color images of size $N = 3 \times s^2$ with $s \in \{128, 256, 512, 1024, 2048, 4096\}$.

For the reasons outlined in Remark 4, the memory required to apply DRUNet in the configuration ($B = 2, N = 3 \times 4096^2$) was too large to fit within the same hardware configurations. Hence, this specific configuration was not tested. This highlights the need for smaller CNNs, such as DDFB, for large-scale applications.

4.2.2 Distributed sampler assessment

Algorithm 1 is evaluated on the three imaging applications described in Section 2.4: image inpainting under additive white Gaussian noise, image deconvolution under Gaussian noise, and image deconvolution under Poisson noise. Both TV regularization and denoiser-based priors are considered. Synthetic observations are generated from ground truth images cropped to predefined sizes N. In experiments involving DDFB, only the instance containing $K = 4$ layers is used.

Restoration quality: Estimation quality is evaluated using $B \in \{1, 2, 4\}$ GPUs. Results are presented in Section 4.3.2.

Scalability: Strong and weak scaling are assessed. Strong scaling is measured with $B \in \{1, 2, 4\}$ GPUs for images of size $N = 3 \times 2048 \times 2048$. Weak scaling is evaluated by increasing both image size and number of GPUs proportionally, with $C = 3$ and $(N_y N_x, B) \in \{(2048^2, 1), (2896^2, 2), (4096^2, 4)\}$. Results are reported in Section 4.3.3.

Sampling configuration

Specific sampling parameters and output metrics are defined for consistent evaluation across all applications. Reconstruction quality is measured using the minimum mean square error (MMSE) estimator. Uncertainty quantification is re-

ported as the pixel-wise variance of the Markov chain after burn-in.[1] This approach enables online computation without additional storage cost. Scalability is quantified by the average runtime per iteration. Unless stated otherwise, all experiments use $T = 10{,}000$ samples, including 1,000 burn-in iterations. Estimates and pixel-wise variances are computed from the resulting chains.

Application-specific parameters

In all applications, the parameters $\alpha, \epsilon, \lambda$ are tuned following the guidelines provided in Laumont et al., 2022, Section 4.1.

- **Inpainting under Gaussian noise** (Section 2.4.1): Observations $\boldsymbol{y}$ are generated from $\overline{\boldsymbol{x}}$ by randomly hiding 70% of its pixels and polluted with Gaussian noise of variance σ^2, corresponding to an input SNR of 15 dB. The sampler is initialized with bicubic spline interpolation of the observations.
  - *TV prior:* $\rho = 10^{-5}$, $\beta = 40$
  - *Denoiser prior:* $\alpha = 1$, $\epsilon = \sigma$, $\lambda = 0.99/(4L + 2\alpha\epsilon^{-2}L_D)$

- **Deconvolution under Gaussian noise** (Section 2.4.2): The operator $\boldsymbol{H}$ is a motion blur with kernel size $L_x \times L_y$ proportional to image size:

  $$(N, L_x L_y) \in \{(3 \times 2048^2, 65^2), (3 \times 2896^2, 91^2), (3 \times 4096^2, 129^2)\}.$$

  Observations are corrupted with Gaussian noise of variance σ^2, with an input SNR of 25 dB. Parameters $\rho, \beta, \alpha, \epsilon$ are the same as for the inpainting case. The parameter λ is valued using the same strategy as well.

- **Deconvolution under Poisson noise** (Section 2.4.3): Observations are generated with $\eta = 250$ and the same convolution kernel as in the Gaussian deconvolution setting. In these experiments, $\rho_1 = 10$, $\rho_2 = 10^{-3}$.
  - *TV prior:* $\beta = 13$
  - *Denoiser prior:* $\alpha = 1$, $\epsilon = 0.05$, $\lambda = 0.99/(4\rho_2^{-1} + 2\alpha\epsilon^{-2}L_D)$

4.3 Experimental results

Table 4.2: Denoiser comparison in terms of restoration quality ($B = 1$).

Denoiser	**SNR**	**PSNR**	**SSIM**
Noisy input	20.00 ± 0.07	27.80 ± 3.95	0.689 ± 0.196
DDFB ($K = 4$)	27.24 ± 3.09	35.04 ± 4.18	0.928 ± 0.041
DDFB ($K = 20$)	27.61 ± 3.27	35.41 ± 4.24	0.935 ± 0.038
DnCNN	28.54 ± 3.40	36.10 ± 3.90	0.948 ± 0.031
DRUNet	29.13 ± 3.74	36.69 ± 4.19	0.953 ± 0.028

4.3.1 Denoising performance

The intrinsic restoration quality of the considered denoisers is reported in Table 4.2. All networks provide a substantial improvement over the noisy input.

Among the learned priors, DRUNet consistently achieves the best performance across all metrics, reaching an average SNR of 29.13 dB, PSNR of 36.69 dB, and SSIM of 0.953. DnCNN follows closely with slightly inferior results, while the two DDFB configurations yield competitive performance given their limited number of parameters. Increasing the number of layers in DDFB from $K = 4$ to $K = 20$ provides a marginal gain in restoration quality, with respect to the performance of DnCNN and DRUNet.

These performance differences reflect the characteristics of the networks summarized in Table 4.1. In particular, DRUNet's superior quality comes at the cost of a very large model: with over 32 million parameters, it is nearly three orders of magnitude larger than DDFB ($K = 4$), whose parameter count remains below 10^4. A similar observation can be made on memory usage, with DRUNet requiring more than 120 MiB compared to only 0.03 MiB for DDFB ($K = 4$). DnCNN, positioned between these extremes, combines moderate resource requirements with competitive restoration quality. On modern GPUs, which typically come with more than 8 GiB of memory, duplicating CNN denoisers across workers is generally not a problem, even for large models such as DRUNet. The main limitation arises when these networks process large images. For instance, processing a single 4096×4096 color image with DRUNet requires about 24 GiB of memory for inference, and substantially more during training, as gradients must also be stored. Because the amount of temporary data scales with both the image size and the network depth, large models like DRUNet are far more likely to run into memory

[1] Pixel-wise variance can be accumulated during sampling, whereas the computation of credible intervals requires storing every sample. For instance, the storage of 10,000 samples of size 3×2048^2 for a single experiment represents about 469 GiB.

Table 4.3: Reconstruction quality for all problems with $B \in \{1, 2, 4\}$.

Task	Prior	rSNR (↑)	PSNR (↑)	SSIM (↑)
Inpainting	TV	21.12	27.42	0.68
	DDFB	22.88	29.19	0.79
	DnCNN	20.76	27.07	0.74
	DRUNet	23.59	29.90	0.81
Gaussian Deconv.	TV	17.56	23.86	0.63
	DDFB	21.05	27.35	0.79
	DnCNN	18.93	25.23	0.70
	DRUNet	20.47	26.78	0.74
Poisson Deconv.	TV	19.47	25.78	0.66
	DDFB	20.31	26.61	0.75
	DnCNN	18.78	25.08	0.71
	DRUNet	18.52	24.83	0.61

issues when applied to high-resolution images than when being duplicated across workers.

Overall, the results highlight a clear trade-off: larger networks such as DRUNet offer the best denoising accuracy but incur significantly higher computational and memory costs, whereas lightweight architectures such as DDFB provide satisfactory performance while consuming far fewer resources.

4.3.2 Reconstruction quality

The quality of the MMSE estimates obtained with Algorithm 1 is summarized in Table 4.3. Among the priors tested, DDFB consistently gives estimates with satisfactory and stable metrics, outperforming the TV baseline.

Despite superior performance in pure denoising tasks (see Section 4.3.1), the DnCNN denoiser yields poorer estimates. This degradation stems primarily from the denoiser's design: it performs blind denoising, *i.e.*, without explicit regularization parameter ϵ. While DnCNN was trained across a range of noise levels, the PnP-ULA framework assumes a fixed denoising strength ϵ throughout the sampling process. The mismatch between the network's implicit behavior and the fixed parameter required by the sampler complicates tuning and may lead to instability, ultimately degrading the reconstruction quality.

DRUNet achieves the best results for the inpainting task, but its performance deteriorates sharply for the deconvolution problem in presence of Poisson noise. These poor metrics come from spurious details and high-frequency artifacts in

Table 4.4: Number of floating point operations, total communication cost, number of communications and average message size to distribute the different priors.

Prior	FLOPs	Total Comm.	♯ Comms.	Message size
TV	$6N_x(N_y/B+1)$	$6N_x$	1	$6N_x$
DDFB	$27\,648N_x(N_y/B+2)$	$536N_x$	8	$67N_x$
DnCNN	$1\,336\,451N_x(N_y/B+2)$	$2\,438N_x$	20	$122N_x$
DRUNet	$4\,236\,054N_x(N_y/B+2)$	$7\,304N_x$	58	$126N_x$

pixels where uncertainty in the observations is the highest. These artifacts suggest instability during sampling, likely caused by an underestimation of L_D. A more conservative value could stabilize the sampling at the cost of smaller step sizes, reducing the exploration capability of the Markov chain.

For both the inpainting and Gaussian deconvolution tasks, the best results were obtained when the denoising level ϵ matched the noise level in the observations, *i.e.*, $\epsilon = \sigma$. In this configuration, the denoiser effectively aligns with the degradation model. For deconvolution under Poisson noise, determining an appropriate denoising level is considerably more challenging. In this case, the lack of a clear choice for ϵ can also explain the limited performance for DRUNet.

Quality metrics are further confirmed visually, as illustrated in Figure 4.1. The MMSE and pixel-wise variance estimates are displayed for three tasks: inpainting of an $N = 3 \times 2048^2$ image with $B = 4$ workers (see Section 2.4.1), Gaussian deconvolution with $N = 3 \times 4096^2$ and $B = 4$ (see Section 2.4.2), and Poisson deconvolution with $N = 3 \times 2896^2$ and $B = 2$ (see Section 2.4.3). Note that Algorithm 1 produces estimates equivalent to its sequential implementation up to the sequence of pseudo-random numbers. This is also confirmed by zooms on communication boundaries in Figure 4.1, showing no tessellation artifacts.

4.3.3 Scalability experiments

Scalability is analyzed at two levels: the denoisers isolated from the sampler, and the complete distributed sampler. At both levels, the balance between computation and communication costs determines achievable speed-ups and efficiency.

Scalability of deep denoisers

Table 4.4 compares the computation and communication costs of the different denoisers using the proposed distributed implementation introduced in Section 2.2.

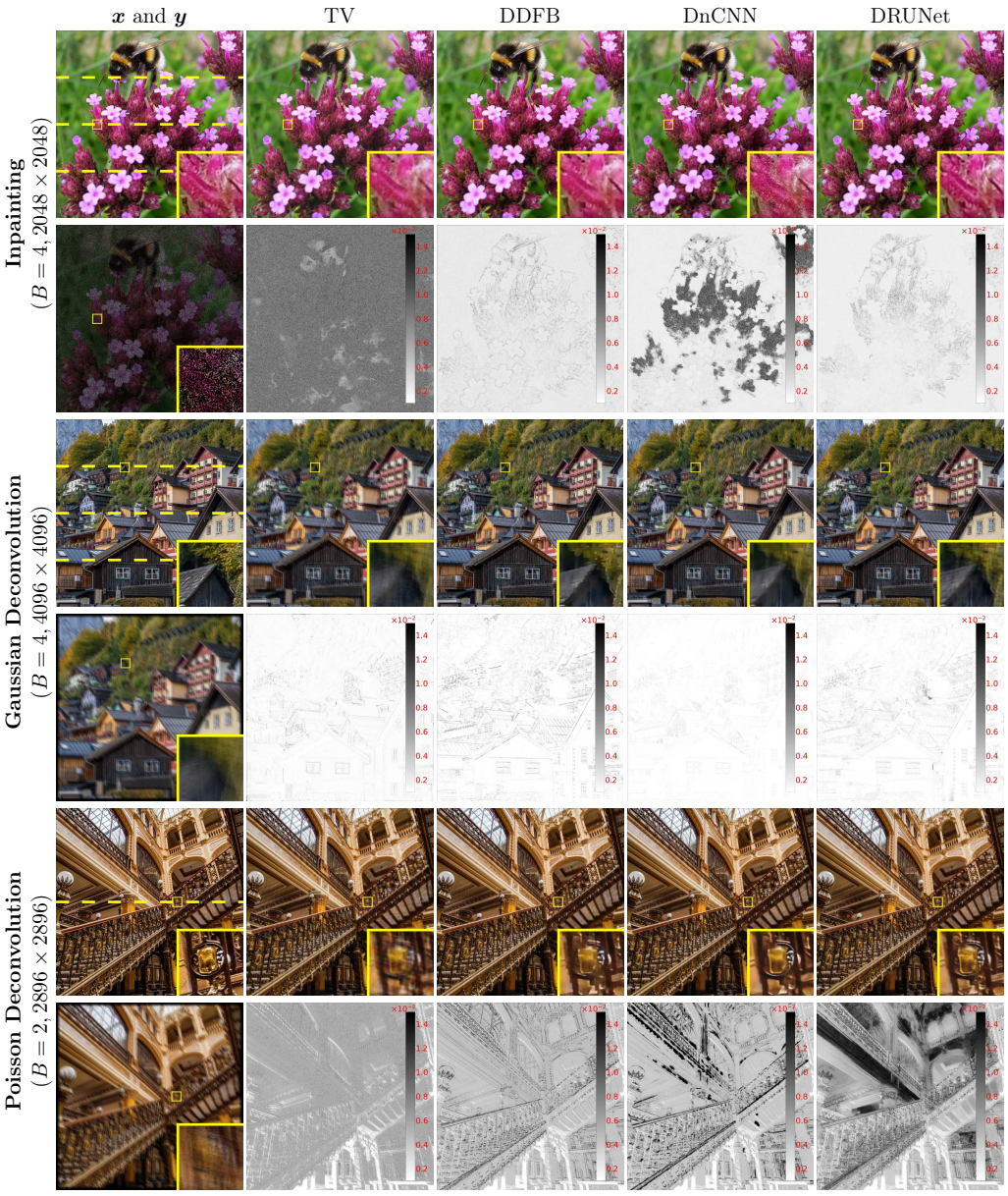

Figure 4.1: Restoration results with Algorithm 1 using different configurations. First column: ground truth $\boldsymbol{x}$ and observations $\boldsymbol{y}$. Last 4 columns, left to right: MMSE and pixel variance of the red channel using TV, DDFB, DnCNN, and DRUNet. Dashed lines on ground truth images show partitions considered for $\boldsymbol{x}$.

Computation load is reported as the maximum amount of floating-point operations (FLOPs) carried out by a worker, estimated numerically with the `calflops` library[2]. Communication costs are reported as the total number of elements a worker needs to send in the distributed setting from Section 2.2. The number of communication phases corresponds to the number of times workers need to send messages. The average message size is the total number of elements sent divided by the number of communication phases. Note that splitting along both spatial dimensions N_x and N_y would lead to comparable computing loads and message sizes, with differences mainly in the number of communications (up to 4 neighbour workers, instead of 2 when partitioning along a single axis).

Although DDFB requires far fewer computations and parameters than the other deep denoisers, its average message size costs remain similar. In fact, only narrow border regions need to be exchanged between workers (1 pixel wide for TV, against $L_y - 1 = 2$ for the CNNs). Dominant factors in communication costs are the number of channels (for message sizes), and the number of convolutional layers for the number of communication phases.

For DRUNet, although the number of channels doubles after each downsampling layer, both spatial dimensions N_x and N_y are divided by 2: the resulting computation load is thus divided by 2. For N_x, N_y and B fixed, the ratio of FLOPs to communication volume per worker is similar for DnCNN and DRUNet, and approximately 10 times larger than DDFB. This suggests DnCNN and DRUNet are more computationally intensive relative to their communication cost, indicating these can benefit the most from a distributed implementation. However, both DnCNN and DRUNet require significantly more communication phases (20 and 58) than DDFB (8 phases). This reduces the practical scalability of DnCNN and DRUNet, with large overheads incurred to establish communications.

Figure 4.2 presents the empirical runtime results that demonstrate the scaling behavior of the deep denoisers across increasing image sizes and varying numbers of workers. The first column compares performance across different denoising architectures (DDFB-4, DDFB-20, DnCNN, and DRUNet) for each distributed configuration, while the second one illustrates the scaling behavior of individual denoisers across different worker counts. For smaller sizes $N_x N_y$ (e.g. 128^2 and 256^2), all denoising models exhibit relatively low runtimes around 10 ms, with distributed execution providing counterproductive overhead. As $N_x N_y$ increase beyond 2048^2 (or 1024^2 for DnCNN), distributed processing achieves substantial speed-ups. The bottom row clearly demonstrates that all denoisers exhibit the

[2]Available at https://github.com/MrYxJ/calculate-flops.pytorch

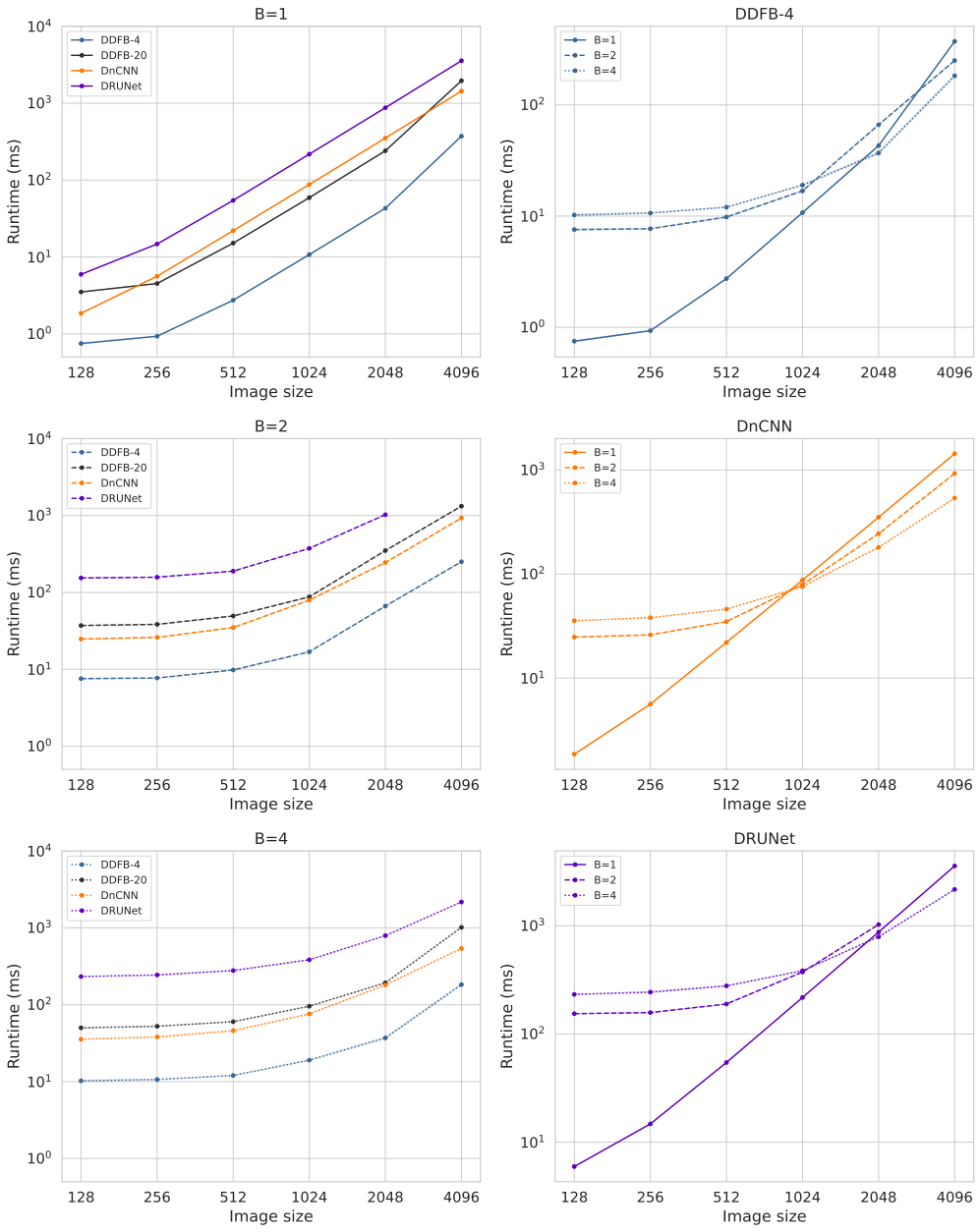

Figure 4.2: Runtime of denoisers applied to square color images of increasing size with $B \in \{1, 2, 4\}$ workers. The x-axis represents the full image width. Results highlight the different scaling behaviors of lightweight networks (DDFB) compared to heavier models (DnCNN, DRUNet).

Table 4.5: Strong scaling experiments: runtime (in ms) and speed-up.

Task	**Prior**	**Time in ms (speed-up w.r.t $B = 1$)**		
		$B = 1$	$B = 2$	$B = 4$
Inpainting	TV	7.09	7.18 (0.99)	4.17 (1.40)
	DDFB	88.97	65.09 (1.37)	36.80 (2.42)
	DnCNN	342.13	241.40 (1.42)	176.29 (1.94)
	DRUNet	1505.90	998.22 (1.51)	771.46 (1.95)
Gaussian Deconv.	TV	11.71	21.16 (0.55)	24.32 (0.48)
	DDFB	94.84	80.75 (1.17)	57.08 (1.66)
	DnCNN	353.00	251.72 (1.40)	196.05 (1.80)
	DRUNet	1499.61	1001.38 (1.50)	791.41 (1.89)
Poisson Deconv.	TV	14.04	24.24 (0.58)	25.11 (0.56)
	DDFB	97.56	84.05 (1.16)	59.82 (1.63)
	DnCNN	356.75	255.86 (1.39)	197.11 (1.81)
	DRUNet	1514.14	1122.49 (1.35)	793.18 (1.91)

most significant runtime reductions for size 4096^2 with DnCNN and DRUNet scaling respectively from 1.4 s and 3.6 s, to 0.5 s and 2.2 s when increasing from $B = 1$ to $B = 4$ GPUs, corresponding to a speed-up of approximately 2.8× for DnCNN and 1.6× for DRUNet. As expected, this speed-up is not close to the ideal 4× due to communication overheads, but it remains significant and increases past a certain image size.

Scalability of the distributed sampler

Strong scaling: Table 4.5 reports strong scaling results for the applications described in section 2.4. For the considered image size, samplers based on deep priors exhibit good scaling behaviour due to a favorable ratio between computation and communication costs, with speed-ups between 1.16 and 1.51 for $B = 2$, and 1.63 and 2.42 for $B = 4$. This behavior aligns with the results obtained when denoisers are assessed outside the sampler, as the denoising step dominates the runtime and benefits most from parallelization.

Results for the TV prior illustrate scalability limits in low-computation regimes. Speed-ups lower than 1 occur when computational loads are too small relative to communication overheads, indicating that using several GPUs is not justified for such tasks. These observations highlight the importance of having a sufficiently large computational load before considering distributed sampling for speed-up.

Table 4.6: Weak scaling experiments: runtime (in ms) and efficiency for $(B, N_y N_x) \in \{(1, 2048^2), (2, 2896^2), (4, 4096^2)\}$.

Task	**Prior**	**Time in ms (efficiency w.r.t. setting $B = 1$)**		
		$(1, 2048^2)$	$(2, 2896^2)$	$(4, 4096^2)$
Inpainting	TV	7.09	13.87 (0.51)	14.47 (0.49)
	DDFB	88.97	127.30 (0.70)	186.37 (0.47)
	DnCNN	342.13	468.51 (0.73)	540.66 (0.63)
	DRUNet	1505.90	1891.94 (0.80)	2180.04 (0.69)
Gaussian Deconv.	TV	11.71	43.86 (0.27)	92.73 (0.13)
	DDFB	94.84	159.76 (0.59)	268.02 (0.35)
	DnCNN	353.00	500.53 (0.71)	620.98 (0.57)
	DRUNet	1499.61	1931.7 (0.78)	2249.29 (0.67)
Poisson Deconv.	TV	14.04	48.12 (0.29)	98.24 (0.14)
	DDFB	97.56	173.99 (0.56)	277.35 (0.35)
	DnCNN	356.75	509.76 (0.70)	628.69 (0.57)
	DRUNet	1514.14	1958.7 (0.77)	2260.03 (0.67)

Weak scaling: Table 4.6 presents weak scaling results, where both the number of workers and problem size are increased proportionally. Ideally, runtime should remain constant if communication costs scale favorably. In practice, a gradual degradation in efficiency is observed with B due to increasing communication overheads. This behavior is expected, as larger images require exchanging more data at each iteration. The main interest lies in determining up to which problem size and number of workers the algorithm maintains satisfactory efficiency.

As in strong scaling, samplers based on deep priors yield better efficiency than TV. With TV, local computation costs do not sufficiently outweigh communication overheads. For inpainting, all denoisers maintain acceptable efficiency levels (between 70% and 80%), while TV drops to 51% for $B = 2$. For deconvolution, communication costs associated with the likelihood grow with kernel size, resulting in sharp efficiency drops: below 60% for DnCNN and DDFB at 2 workers, and below 40% at 4 workers. The TV prior suffers the most, reaching only 13% efficiency with 4 workers.

Overall, these results confirm that the proposed distributed approach is effective when the computational load per worker is sufficiently large to justify the incurred communication costs, as observed in other distributed computing contexts [Eijkhout, 2023].

4.4 Conclusion

This chapter has presented an extensive evaluation of both the intrinsic properties of the considered deep denoisers and the performance of the proposed distributed sampler, Algorithm 1, across a range of imaging applications.

The benchmarking of denoisers highlighted a clear trade-off between restoration quality and computational efficiency. DRUNet achieved the highest reconstruction metrics in pure denoising tasks but required substantially more computational and memory resources. DnCNN offered a balance between performance and resource consumption. The lightweight DDFB models demonstrated competitive results while requiring significantly less memory allocation, making them attractive candidates for large-scale applications. These differences directly impacted the suitability of each denoiser as a prior within the sampling framework.

The sampling experiments confirmed that the proposed distributed algorithm, completely equivalent to the sequential version, successfully leveraged multi-GPU environments. Estimators' quality depended strongly on the compatibility between the denoiser behavior and the fixed parameters required by the PnP-ULA framework. DDFB provided the most stable and reliable results across applications. In contrast, DnCNN and DRUNet occasionally exhibited instability, particularly in challenging scenarios such as deconvolution under Poisson noise.

Finally, the scalability analysis confirmed that the efficiency of the methodology depends on the balance between computation and communication costs. Deep priors exhibited favorable scaling, achieving significant speed-ups in both strong and weak scaling regimes. In contrast, the TV prior did not scale across GPUs, as the problem sizes resulted in too little computation per worker to offset communication overheads. Overall, the results confirmed that the proposed distributed strategy is most effective when applied to computationally demanding settings, where the communication overhead is outweighed by local processing.

Chapter 5

Conclusions and perspectives

5.1 Conclusions

This thesis has developed a comprehensive framework for distributed Plug-and-Play MCMC sampling, addressing the computational challenges of Bayesian inference in large-scale imaging problems. The work spans methodological foundations, practical implementation, and experimental validation, demonstrating how distributed computing can help scaling PnP samplers while maintaining equivalence with their serial counterparts. The following synthesis highlights the main achievements across the different contributions of this research.

Chapter 2 established a methodology for developing distributed PnP-inspired MCMC algorithms, which was illustrated through a distributed implementation of the Plug-and-Play unadjusted Langevin algorithm (PnP-ULA) [Laumont et al., 2022]. The approach exploits conditional independence properties and operator locality to spread computations within a Single Program Multiple Data (SPMD) framework. The approach maintains numerical equivalence with the serial counterpart while enabling significant computational speedups through distributed processing.

Chapter 3 identifies practical challenges to implement the methodological contributions of Chapter 2 into an ergonomic and efficient distributed MCMC framework. The chapter demonstrated that existing MCMC software libraries lack the necessary abstractions for explicit control over both sampler structure and communication schemes in distributed settings, particularly for multi-GPU environments. This motivated the design of a fully fledged Python library, that directly

maps the mathematical building blocks of MCMC algorithms onto modular software classes organized into well-defined layers reflecting the algorithmic structure. This design philosophy achieves both computational efficiency and conceptual clarity by exploiting the generic nature of MCMC methods. The complete implementation demonstrated how abstract MCMC algorithms can be instantiated into practical, reproducible workflows that handle complex distributed computations in a consistent and scalable manner. The engineering contributions include deployment strategies for multi-GPU systems and extendable communication designs, while the modular architecture ensures adaptability to future algorithmic developments across different hardware configurations.

Chapter 4 provided comprehensive experimental validation through extensive evaluation of both deep denoiser properties and the performance of the proposed distributed sampler across three common imaging applications. The denoiser benchmarking revealed fundamental trade-offs between restoration quality and computational efficiency. The sampling experiments confirmed that the distributed algorithm can successfully leverage multi-GPU environments, with DDFB providing the most stable results while DnCNN and DRUNet occasionally exhibited instability in challenging scenarios. The scalability analysis confirmed that distributed efficiency depends on the balance between computation and communication costs. For the problem sizes and hardware considered, deep priors scale favorably while simpler priors like TV can fail to amortize communication overhead due to insufficient local computation. Notably, the distributed approach enabled the processing of 4096^2 color images, a scale that was previously intractable with sequential implementations on the available hardware.

The following section outlines key perspectives for extending and building upon this framework.

5.2 Perspectives

The distributed PnP-MCMC framework established in this work opens several promising research directions. These perspectives range from algorithmic optimizations and architectural extensions to broader applications beyond imaging problems.

Alternative communication strategies: The current implementation performs communication at each operator level within deep denoisers, ensuring min-

imal message sizes regardless of network depth. An alternative approach, such as the receptive field-based strategy proposed by [Galerne et al., 2024], would communicate only once per denoiser application by exchanging larger data blocks based on the network's receptive field. For typical CNN architectures with small 3×3 kernels, this could significantly reduce overall communication costs at the expense of slightly increased computational overhead and memory footprint. However, the memory requirements would scale with network depth as receptive fields grow, potentially limiting applicability to very deep architectures. A comprehensive theoretical and experimental comparison of these communication paradigms would provide valuable insights into their respective trade-offs.

Integration of modern generative architectures: Extending the framework to incorporate transformer-based models, diffusion architectures, and other state-of-the-art generative priors represents a significant research opportunity. These models exhibit fundamentally different structural properties compared to CNN-based denoisers, particularly through non-local attention mechanisms that challenge traditional domain decomposition strategies. The distributed implementation would require different approaches to handle the global dependencies inherent in transformer architectures while preserving computational efficiency. This extension would necessitate developing approximation techniques and communication patterns tailored to the unique features of modern generative models.

Extension to non-Cartesian data structures: While this work focused on imaging problems with natural Cartesian grid structures, many applications involve data defined on graphs, manifolds, or other complex topologies. Adapting the distributed methodology to these settings requires addressing broader challenges [Meng et al., 2024]. For instance, it can be difficult to define locality and neighborhood relationships for irregular structures while still ensuring computational efficiency. The proposed approach could be applied provided partitioning algorithms that balance computational loads while minimizing communication overhead across irregular connectivity patterns.

Asynchronous computing extensions: Relaxing synchronization constraints in the current framework could yield significant runtime improvements. Allowing computational nodes to skip communication phases at selected iterations or denoiser layers could substantially reduce communication bottlenecks. However, this flexibility introduces critical challenges in maintaining the theoretical guaran-

tees of MCMC methods, as stale information and delayed updates can introduce bias and affect convergence properties [De Sa et al., 2016; Terenin et al., 2020]. Future research must establish rigorous theoretical foundations for asynchronous PnP-MCMC methods while developing practical strategies to monitor and control the trade-offs between computational efficiency and statistical accuracy.

These research directions collectively represent a roadmap toward more flexible, efficient, and broadly applicable distributed MCMC algorithms. By addressing the outlined challenges, future work can build upon the foundation established in this thesis to improve the scalability of Bayesian inference methods leveraging deep priors across diverse applications and data structures.

References

Agustsson, Eirikur and Radu Timofte [July 2017]. "NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1122–1131. DOI: 10.1109/CVPRW.2017.150 [cit. on pp. 66, 68].

Ahn, Sungjin, Babak Shahbaba, and Max Welling [June 18, 2014]. "Distributed Stochastic Gradient MCMC". In: *Proceedings of the 31st International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 1044–1052. URL: https://proceedings.mlr.press/v32/ahn14.html [cit. on p. 23].

Alzubaidi, Laith et al. [2021]. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". In: *J. Big Data* 8, pp. 1–74 [cit. on p. 7].

Angelino, Elaine, Matthew James Johnson, and Ryan P. Adams [2016]. "Patterns of Scalable Bayesian Inference". In: *Foundations and Trends® in Machine Learning* 9.2-3, pp. 119–247. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000052 [cit. on p. 20].

Ansel, Jason et al. [Apr. 2024]. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation". In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM. DOI: 10.1145/3620665.3640366. URL: https://pytorch.org/assets/pytorch2-2.pdf [cit. on p. 51].

Atchadé, Yves and Liwei Wang [Nov. 1, 2023]. "A Fast Asynchronous Markov Chain Monte Carlo Sampler for Sparse Bayesian Inference". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 85.5, pp. 1492–1516. ISSN: 1369-7412. DOI: 10.1093/jrsssb/qkad078 [cit. on p. 23].

Baydin, Atılım Günes et al. [Jan. 1, 2017]. "Automatic Differentiation in Machine Learning: A Survey". In: *J. Mach. Learn. Res.* 18.1, pp. 5595–5637. ISSN: 1532-4435 [cit. on p. 51].

Beck, Amir [Oct. 3, 2017]. *First-Order Methods in Optimization*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics. 477 pp. ISBN: 978-1-61197-498-0. DOI: 10.1137/1.9781611974997. URL: https://epubs.siam.org/doi/book/10.1137/1.9781611974997 [cit. on p. 9].

Benning, Martin and Martin Burger [2018]. "Modern regularization methods for inverse problems". In: *Acta numerica* 27, pp. 1–111 [cit. on pp. vii, 5].

Bingham, Eli et al. [2019]. "Pyro: Deep Universal Probabilistic Programming". In: *Journal of Machine Learning Research* 20.28, pp. 1–6. ISSN: 1533-7928. URL: http://jmlr.org/papers/v20/18-403.html [cit. on p. 47].

Bouton, Maxime et al. [2025a]. "A Distributed Plug-and-Play MCMC Algorithm for High-Dimensional Inverse Problems". Submitted to IEEE Transactions on Computational Imaging. [cit. on pp. 25, 28].

— [June 2025b]. "Multi-GPU Distributed PnP-ULA for High-Dimensional Imaging Inverse Problems". In: *23rd IEEE Statistical Signal Processing Workshop (IEEE SSP 2025)*. Winner for Best Student Paper Award. Edimbourg, United Kingdom. URL: https://hal.science/hal-05143098 [cit. on pp. 25, 28].

— [Aug. 2025c]. "Un Échantillonneur Plug and Play Distribué Pour La Résolution de Problèmes Inverses de Très Grande Dimension". In: *Colloque Gretsi 2025*. Strasbourg, France. URL: https://hal.science/hal-05143079 [cit. on pp. 25, 28].

— [Mar. 27, 2025d]. *Une approche PnP distribuée pour léchantillonnage de problèmes inverses en grande dimension*. Journée GdR IASIS - "Evènements rares et réduction de modèles". IHP, Paris, France [cit. on p. 26].

— [May 13, 2025e]. *Une approche PnP distribuée pour léchantillonnage de problèmes inverses en grande dimension*. Journée GdR IASIS - "Unrolling and un/self/supervised learning for inverse problems". INRIA, Paris, France [cit. on p. 26].

Cabezas, Alberto et al. [Feb. 2024]. *BlackJAX: Composable Bayesian Inference in JAX*. DOI: 10.48550/arXiv.2402.10797. arXiv: 2402.10797 [cs] [cit. on p. 47].

Cai, Xiaohao, Marcelo Pereyra, and Jason D McEwen [July 2018]. "Uncertainty quantification for radio interferometric imaging – I. Proximal MCMC methods". In: *Mon. Not. R. Astron. Soc.* 480.3, pp. 4154–4169. ISSN: 0035-8711 [cit. on p. 3].

Cai, Ziruo et al. [June 30, 2024]. "NF-ULA: Normalizing Flow-Based Unadjusted Langevin Algorithm for Imaging Inverse Problems". In: *SIAM J. Imaging Sci.* 17.2, pp. 820–860. DOI: 10.1137/23M1581807. URL: https://epubs.siam.org/doi/10.1137/23M1581807 [cit. on p. 7].

Carpenter, Bob et al. [Jan. 2017]. "Stan: A Probabilistic Programming Language". In: *Journal of Statistical Software* 76, pp. 1–32. ISSN: 1548-7660. DOI: 10.18637/jss.v076.i01 [cit. on p. 47].

Carvalho, Carlos M., Nicholas G. Polson, and James G. Scott [2010]. "The Horseshoe Estimator for Sparse Signals". In: *Biometrika* 97.2, pp. 465–480. ISSN: 0006-3444. JSTOR: 25734098. URL: https://www.jstor.org/stable/25734098 [cit. on p. 5].

Coeurdoux, Florentin, Nicolas Dobigeon, and Pierre Chainais [Sept. 25, 2024a]. "Normalizing Flow Sampling with Langevin Dynamics in the Latent Space". In: *Mach. Learn.* ISSN: 1573-0565. DOI: 10.1007/s10994-024-06623-x. URL: https://doi.org/10.1007/s10994-024-06623-x [cit. on p. 7].

— [2024b]. "Plug-and-Play Split Gibbs Sampler: Embedding Deep Generative Priors in Bayesian Inference". In: *IEEE Trans. Image Process.* 33, pp. 3496–3507. ISSN: 1941-0042. DOI: 10.1109/TIP.2024.3404338. URL: https://ieeexplore.ieee.org/document/10541919?source=authoralert [cit. on p. 8].

Combettes, Patrick L., Dinh Dũng, and Bang Công Vũ [Dec. 1, 2010]. "Dualization of Signal Recovery Problems". In: *Set-Valued Anal* 18.3, pp. 373–404. ISSN: 1877-0541. DOI: 10.1007/s11228-010-0147-7. URL: https://doi.org/10.1007/s11228-010-0147-7 [visited on 07/25/2025] [cit. on p. 34].

Combettes, Patrick L. and Jean-Christophe Pesquet [May 2011]. "Proximal splitting methods in signal processing". In: *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*. Vol. 49. Springer, pp. 185–212. URL: http://link.springer.com/10.1007/978-1-4419-9569-8_10 [cit. on pp. 6, 12, 34].

Dalcin, Lisandro and Yao-Lung L. Fang [2021]. "mpi4py: Status Update After 12 Years of Development". In: *IEEE Comput. Sci. Eng.* 23.4, pp. 47–54 [cit. on p. 51].

Daras, Giannis et al. [Sept. 30, 2024]. *A Survey on Diffusion Models for Inverse Problems*. DOI: 10.48550/arXiv.2410.00083. arXiv: 2410.00083. URL: http://arxiv.org/abs/2410.00083. Pre-published [cit. on p. 7].

De Sa, Christopher, Kunle Olukotun, and Christopher Ré [2016]. "Ensuring Rapid Mixing and Low Bias for Asynchronous Gibbs Sampling". In: *JMLR work-*

shop and conference proceedings 48, pp. 1567–1576. ISSN: 1938-7288. PMID: 28344730. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5360990/ [cit. on pp. 22, 83].

De Valpine, Perry et al. [Apr. 3, 2017]. "Programming With Models: Writing Statistical Algorithms for General Model Structures With NIMBLE". In: *Journal of Computational and Graphical Statistics* 26.2, pp. 403–413. ISSN: 1061-8600. DOI: 10.1080/10618600.2016.1172487 [cit. on p. 48].

Dillon, Joshua V. et al. [Nov. 28, 2017]. *TensorFlow Distributions*. DOI: 10.48550/arXiv.1711.10604. arXiv: 1711.10604 [cs]. Pre-published [cit. on p. 47].

Durmus, Alain and Éric Moulines [June 2017]. "Nonasymptotic Convergence Analysis for the Unadjusted Langevin Algorithm". In: *Ann. Appl. Probab.* 27.3, pp. 1551–1587. ISSN: 1050-5164. DOI: 10.1214/16-AAP1238. URL: http://projecteuclid.org/euclid.aoap/1500451235 [cit. on p. 12].

Durmus, Alain, Éric Moulines, and Marcelo Pereyra [Jan. 1, 2018]. "Efficient Bayesian Computation by Proximal Markov Chain Monte Carlo: When Langevin Meets Moreau". In: *SIAM J. Imaging Sci.* 11.1, pp. 473–506. DOI: 10.1137/16M1108340 [cit. on pp. viii, 12, 13, 31, 39, 42].

Efron, Bradley [Dec. 1, 2011]. "Tweedie's Formula and Selection Bias". In: *Journal of the American Statistical Association* 106.496, pp. 1602–1614. ISSN: 0162-1459. DOI: 10.1198/jasa.2011.tm11181. PMID: 22505788 [cit. on p. 10].

Eijkhout, Victor [2023]. *The Science of Computing*. URL: https://github.com/VictorEijkhout/TheArtofHPC_pdfs [cit. on pp. viii, 17–19, 30, 37, 78].

Faye, Elhadji C., Mame Diarra Fall, and Nicolas Dobigeon [2025]. "Regularization by Denoising: Bayesian Model and Langevin-Within-Split Gibbs Sampling". In: *IEEE Trans. Image Process.* 34, pp. 221–234. DOI: 10.1109/TIP.2024.3520012 [cit. on pp. 3, 11, 16, 39].

Fleuret, François [2024]. *The little book of deep learning*. URL: https://fleuret.org/public/lbdl.pdf [cit. on p. 7].

Foreman-Mackey, Daniel et al. [2013]. "Emcee: The MCMC Hammer". In: *Publications of the Astronomical Society of the Pacific* 125.925, pp. 306–312. ISSN: 0004-6280. DOI: 10.1086/670067. JSTOR: 10.1086/670067 [cit. on p. 47].

Gagneux, Anne et al. [June 19, 2025]. "Convexity in ReLU Neural Networks: Beyond ICNNs?" In: *Journal of Mathematical Imaging and Vision* 67.4, p. 40. ISSN: 1573-7683. DOI: 10.1007/s10851-025-01253-x [cit. on p. 9].

Galerne, Bruno et al. [2024]. "Scaling Painting Style Transfer". In: *Computer Graphics Forum* 43.4, e15155. DOI: https://doi.org/10.1111/cgf.15155. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.15155.

URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.15155 [cit. on pp. 38, 82].

Ge, Hong, Kai Xu, and Zoubin Ghahramani [Mar. 31, 2018]. "Turing: A Language for Flexible Probabilistic Inference". In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. International Conference on Artificial Intelligence and Statistics. PMLR, pp. 1682–1690. URL: https://proceedings.mlr.press/v84/ge18b.html [cit. on p. 48].

Gelfand, Alan E. and Adrian F. M. Smith [June 1, 1990]. "Sampling-Based Approaches to Calculating Marginal Densities". In: *Journal of the American Statistical Association* 85.410, pp. 398–409. ISSN: 0162-1459. DOI: 10.1080/01621459.1990.10476213 [cit. on p. 5].

Gilks, W. R., A. Thomas, and D. J. Spiegelhalter [1994]. "A Language and Program for Complex Bayesian Modelling". In: *Journal of the Royal Statistical Society: Series D (The Statistician)* 43.1, pp. 169–177. ISSN: 1467-9884. DOI: 10.2307/2348941 [cit. on p. 48].

Girolami, Mark and Ben Calderhead [2011]. "Riemann Manifold Langevin and Hamiltonian Monte Carlo Methods". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73.2, pp. 123–214. ISSN: 1467-9868. DOI: 10.1111/j.1467-9868.2010.00765.x [cit. on p. 13].

Goudie, Robert J. B. et al. [Oct. 7, 2020]. "MultiBUGS: A Parallel Implementation of the BUGS Modeling Framework for Faster Bayesian Inference". In: *Journal of Statistical Software* 95, pp. 1–20. ISSN: 1548-7660. DOI: 10.18637/jss.v095.i07 [cit. on p. 47].

Goujon, Alexis et al. [2023]. "A Neural-Network-Based Convex Regularizer for Inverse Problems". In: *IEEE Trans. Comput. Imag.* 9, pp. 781–795. ISSN: 2333-9403. DOI: 10.1109/TCI.2023.3306100 [cit. on p. 9].

Graham, Matthew M. [Aug. 15, 2025]. *Mici: Manifold Markov Chain Monte Carlo Methods in Python*. DOI: 10.5281/zenodo.3517301 [cit. on p. 47].

Grama, Ananth et al. [Jan. 2003]. *Introduction to Parallel Computing (2nd Edition)*. ISBN: 0201648652 [cit. on pp. 17, 67].

Gropp, William et al. [Nov. 7, 2014]. *Using Advanced MPI: Modern Features of the Message-Passing Interface*. Red. by William Gropp. Scientific and Engineering Computation. Cambridge, MA, USA: MIT Press. 392 pp. ISBN: 978-0-262-52763-7 [cit. on p. 17].

Hadamard, J. [1902]. "Sur les problèmes aux dérivés partielles et leur signification physique". In: *Princeton University Bulletin* 13, pp. 49–52 [cit. on p. 2].

Hafych, Vasyl et al. [June 27, 2022]. "Parallelizing MCMC Sampling via Space Partitioning". In: *Statistics and Computing* 32.4, p. 56. ISSN: 1573-1375. DOI: 10.1007/s11222-022-10116-z [cit. on p. 22].

Hastings, W. K. [1970]. "Monte Carlo Sampling Methods Using Markov Chains and Their Applications". In: *Biometrika* 57.1, pp. 97–109. ISSN: 0006-3444. DOI: 10.2307/2334940. JSTOR: 2334940 [cit. on p. 5].

He, Kaiming et al. [June 2016]. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778. DOI: 10.1109/CVPR.2016.90 [cit. on p. 7].

Holden, Matthew, Marcelo Pereyra, and Konstantinos C. Zygalakis [June 30, 2022]. "Bayesian Imaging with Data-Driven Priors Encoded by Neural Networks". In: *SIAM J. Imaging Sci.* 15.2, pp. 892–924. DOI: 10.1137/21M1406313. URL: https://epubs.siam.org/doi/10.1137/21M1406313 [cit. on p. 3].

Homan, Matthew D. and Andrew Gelman [Jan. 1, 2014]. "The No-U-turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo". In: *J. Mach. Learn. Res.* 15.1, pp. 1593–1623. ISSN: 1532-4435 [cit. on p. 47].

Jordan, Michael I., Jason D. Lee, and Yun Yang [Apr. 3, 2019]. "Communication-Efficient Distributed Statistical Inference". In: *Journal of the American Statistical Association* 114.526, pp. 668–681. ISSN: 0162-1459. DOI: 10.1080/01621459.2018.1429274 [cit. on p. 21].

Kamilov, Ulugbek S. et al. [Jan. 2023]. "Plug-and-Play Methods for Integrating Physical and Learned Models in Computational Imaging: Theory, Algorithms, and Applications". In: *IEEE Signal Process. Mag.* 40.1, pp. 85–97. ISSN: 1053-5888, 1558-0792. DOI: 10.1109/MSP.2022.3199595. URL: https://ieeexplore.ieee.org/document/10004791/ [cit. on pp. vii, 9].

Kawar, Bahjat et al. [Dec. 6, 2022]. "Denoising Diffusion Restoration Models". In: *Adv. in Neural Information Processing Systems* 35, pp. 23593–23606. URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/95504595b6169131b6ed6cd72eb05616-Abstract-Conference.html [visited on 11/07/2024] [cit. on p. 7].

Kingma, Diederik P. and Jimmy Ba [Jan. 30, 2017]. *Adam: A Method for Stochastic Optimization*. Version 9. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980 [cs]. URL: https://arxiv.org/abs/1412.6980 [visited on 06/04/2025]. Pre-published [cit. on p. 67].

Kobyzev, Ivan, Simon J. D. Prince, and Marcus A. Brubaker [Nov. 1, 2021]. "Normalizing Flows: An Introduction and Review of Current Methods". In:

IEEE Trans. Pattern Anal. Mach. Intell. 43.11, pp. 3964–3979. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2020.2992934. arXiv: 1908.09257 [cs, stat]. URL: http://arxiv.org/abs/1908.09257 [visited on 03/12/2023] [cit. on p. 7].

Laumont, Rémi et al. [June 2022]. "Bayesian Imaging Using Plug & Play Priors: When Langevin Meets Tweedie". In: *SIAM J. Imaging Sci.* 15.2, pp. 701–737. ISSN: 1936-4954. DOI: 10.1137/21M1406349. arXiv: 2103.04715 [cs, eess, math, stat] [cit. on pp. viii, 3, 10, 11, 16, 24, 27, 31, 39–43, 70, 80].

Le, Hoang Trieu Vy, Nelly Pustelnik, and Marion Foare [Aug. 2022]. "The Faster Proximal Algorithm, the Better Unfolded Deep Learning Architecture ? The Study Case of Image Denoising". In: *Proc. European Signal Process. Conf. (EUSIPCO)*. 2022 30th European Signal Processing Conference (EUSIPCO), pp. 947–951. DOI: 10.23919/EUSIPCO55093.2022.9909592. URL: https://ieeexplore.ieee.org/document/9909592 [visited on 06/03/2025] [cit. on pp. 9, 66].

Le, Hoang Trieu Vy, Audrey Repetti, and Nelly Pustelnik [2024]. "Unfolded Proximal Neural Networks for Robust Image Gaussian Denoising". In: *IEEE Trans. Image Process.* 33, pp. 4475–4487. ISSN: 1941-0042. DOI: 10.1109/TIP.2024.3437219. URL: https://ieeexplore.ieee.org/document/10630640/?arnumber=10630640 [cit. on pp. 9, 34, 66, 67].

Leclaire, Arthur, Eliot Guez, and Bruno Galerne [Aug. 2025]. "Backward Diffusion Iterates Noising-Relaxed Denoising". In: *Colloque Gretsi 2025*. Strasbourg, France. URL: https://hal.science/hal-05027888 [cit. on p. 8].

Lunn, David et al. [2009]. "The BUGS Project: Evolution, Critique and Future Directions". In: *Statistics in Medicine* 28.25, pp. 3049–3067. ISSN: 1097-0258. DOI: 10.1002/sim.3680 [cit. on p. 47].

Luo, Guanxiong et al. [July 2023]. "Bayesian MRI Reconstruction with Joint Uncertainty Estimation Using Diffusion Models". In: *Magnetic Resonance in Medicine* 90.1, pp. 295–311. ISSN: 1522-2594. DOI: 10.1002/mrm.29624. PMID: 36912453 [cit. on p. 2].

Mallat, S. G. [2009]. *A Wavelet Tour of Signal Processing: The Sparse Way*. 3rd ed. Amsterdam ; Boston: Elsevier/Academic Press. 805 pp. ISBN: 978-0-12-374370-1 [cit. on p. 6].

Mbakam, Charlesquin Kemajou, Jean-Francois Giovannelli, and Marcelo Pereyra [Sept. 6, 2024]. *Empirical Bayesian Image Restoration by Langevin Sampling with a Denoising Diffusion Implicit Prior*. DOI: 10.48550/arXiv.2409.04384.

arXiv: 2409.04384 [cs]. URL: http://arxiv.org/abs/2409.04384 [visited on 05/26/2025]. Pre-published [cit. on p. 8].

Meng, Lingkai et al. [Oct. 10, 2024]. "A Survey of Distributed Graph Algorithms on Massive Graphs". In: *ACM Comput. Surv.* 57.2, 27:1–27:39. ISSN: 0360-0300. DOI: 10.1145/3694966 [cit. on p. 82].

Meyn, Sean and Richard L. Tweedie [2009]. *Markov Chains and Stochastic Stability*. 2nd ed. Cambridge Mathematical Library. Cambridge: Cambridge University Press. ISBN: 978-0-521-73182-9. DOI: 10.1017/CBO9780511626630 [cit. on pp. vii, 5].

Mitchell, T. J. and J. J. Beauchamp [Dec. 1, 1988]. "Bayesian Variable Selection in Linear Regression". In: *Journal of the American Statistical Association* 83.404, pp. 1023–1032. ISSN: 0162-1459. DOI: 10.1080/01621459.1988.10478694 [cit. on p. 5].

Monga, Vishal, Yuelong Li, and Yonina C. Eldar [Mar. 2021]. "Algorithm Unrolling: Interpretable, Efficient Deep Learning for Signal and Image Processing". In: *IEEE Signal Process. Mag.* 38.2, pp. 18–44. ISSN: 1558-0792. DOI: 10.1109/MSP.2020.3016905. URL: https://ieeexplore.ieee.org/document/9363511 [cit. on pp. 8, 9].

Moreau, Jean Jacques [1965]. "Proximité et dualité dans un espace hilbertien". In: *Bulletin de la société mathématique de France* 93, p. 273. DOI: 10.24033/bsmf.1625. URL: https://hal.science/hal-01740635 [visited on 06/16/2025] [cit. on pp. xvii, 10, 12].

Neal, Radford M. [2011]. "MCMC Using Hamiltonian Dynamics". In: *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC [cit. on pp. 12, 47].

Okuta, Ryosuke et al. [2017]. "CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations". In: *Adv. in Neural Information Processing Systems*. URL: http://learningsys.org/nips17/assets/papers/paper_16.pdf [cit. on p. 50].

Ongie, Gregory et al. [May 2020]. "Deep Learning Techniques for Inverse Problems in Imaging". In: *IEEE J. Sel. Inf. Theory* 1.1, pp. 39–56. ISSN: 2641-8770. DOI: 10.1109/JSAIT.2020.2991563. URL: https://ieeexplore.ieee.org/document/9084378/?arnumber=9084378 [cit. on pp. vii, 3, 6].

Park, Trevor and George Casella [June 1, 2008]. "The Bayesian Lasso". In: *Journal of the American Statistical Association* 103.482, pp. 681–686. ISSN: 0162-1459, 1537-274X. DOI: 10.1198/016214508000000337 [cit. on p. 14].

Pereyra, Marcelo [July 2016]. "Proximal Markov Chain Monte Carlo Algorithms". In: *Stat. Comput.* 26.4, pp. 745–760. ISSN: 1573-1375 [cit. on pp. 12, 13].

Pereyra, Marcelo, Luis Vargas Mieles, and Konstantinos C. Zygalakis [Jan. 2020]. "Accelerating Proximal Markov Chain Monte Carlo by Using an Explicit Stabilized Method". In: *SIAM J. Imaging Sci.* 13.2, pp. 905–935. DOI: 10.1137/19M1283719. URL: https://epubs.siam.org/doi/abs/10.1137/19M1283719 [visited on 05/29/2025] [cit. on p. 39].

Pereyra, Marcelo, Luis A. Vargas-Mieles, and Konstantinos C. Zygalakis [Dec. 31, 2023]. "The Split Gibbs Sampler Revisited: Improvements to Its Algorithmic Structure and Augmented Target Distribution". In: *SIAM J. Imaging Sci.* 16.4, pp. 2040–2071. DOI: 10.1137/22M1506122. URL: https://epubs.siam.org/doi/abs/10.1137/22M1506122 [visited on 05/29/2025] [cit. on pp. 13, 15].

Pereyra, Marcelo et al. [Mar. 2016]. "A Survey of Stochastic Simulation and Optimization Methods in Signal Processing". In: *IEEE J. Sel. Topics Signal Process.* 10.2, pp. 224–241. ISSN: 1941-0484. DOI: 10.1109/JSTSP.2015.2496908. eprint: 1505.00273 [cit. on pp. 2, 3, 32].

Phan, Du, Neeraj Pradhan, and Martin Jankowiak [Dec. 24, 2019]. *Composable Effects for Flexible and Accelerated Probabilistic Programming in NumPyro.* DOI: 10.48550/arXiv.1912.11554. arXiv: 1912.11554 [stat]. Pre-published [cit. on p. 47].

Plummer, Martyn [2003]. "JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling". In: *Working Papers* [cit. on p. 47].

Polson, Nicholas G., James G. Scott, and Jesse Windle [Dec. 1, 2013]. "Bayesian Inference for Logistic Models Using Pólya–Gamma Latent Variables". In: *Journal of the American Statistical Association* 108.504, pp. 1339–1349. ISSN: 0162-1459. DOI: 10.1080/01621459.2013.829001 [cit. on p. 14].

Pustelnik, Nelly et al. [2016]. "Wavelet-Based Image Deconvolution and Reconstruction". In: *Wiley Encyclopedia of Electrical and Electronics Engineering.* John Wiley & Sons, Ltd, pp. 1–34. ISBN: 978-0-471-34608-1. DOI: 10.1002/047134608X.W8294 [cit. on p. 5].

Renaud, Marien et al. [May 20, 2025]. *From Stability of Langevin Diffusion to Convergence of Proximal MCMC for Non-Log-Concave Sampling.* DOI: 10.48550/arXiv.2505.14177. arXiv: 2505.14177 [stat]. Pre-published [cit. on pp. viii, 11, 13, 16].

Rendell, Lewis J. et al. [Sept. 8, 2020]. "Global Consensus Monte Carlo". In: *J. Comput. and Graph. Stat.* 30.2, pp. 249–259. DOI: 10.1080/10618600.2020.1811105 [cit. on pp. 14, 15].

Repetti, Audrey et al. [Aug. 29, 2022]. "Dual Forward-Backward Unfolded Network for Flexible Plug-and-Play". In: *Proc. European Signal Process. Conf.*

(EUSIPCO). Proc. European Signal Process. Conf. (EUSIPCO). Belgrade, Serbia: IEEE, pp. 957–961. ISBN: 978-90-827970-9-1. DOI: 10.23919/EUSIPCO55093.2022.9909564 [cit. on pp. ix, 9, 33, 34, 66].

Robert, Christian P. [2007]. *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*. 2nd ed. Springer Texts in Statistics. New York: Springer. 602 pp. ISBN: 978-0-387-71598-8 [cit. on pp. vii, 3, 4].

Robert, Christian P. and George Casella [2004]. *Monte Carlo Statistical Methods*. Springer Texts in Statistics. Springer. ISBN: 978-1-4419-1939-7 978-1-4757-4145-2. DOI: 10.1007/978-1-4757-4145-2 [cit. on p. 4].

Roberts, Gareth O. and Richard L. Tweedie [1996]. "Exponential Convergence of Langevin Distributions and Their Discrete Approximations". In: *Bernoulli* 2.4, pp. 341–363. ISSN: 1350-7265. DOI: 10.2307/3318418. JSTOR: 3318418. URL: https://www.jstor.org/stable/3318418 [visited on 11/05/2024] [cit. on p. 12].

Romano, Yaniv, Michael Elad, and Peyman Milanfar [Jan. 2017]. "The Little Engine That Could: Regularization by Denoising (RED)". In: *SIAM Journal on Imaging Sciences* 10.4, pp. 1804–1844. DOI: 10.1137/16M1102884 [cit. on p. 10].

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox [2015]. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, pp. 234–241. ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28 [cit. on p. 7].

Rudin, Leonid I., Stanley Osher, and Emad Fatemi [Nov. 1992]. "Nonlinear total variation based noise removal algorithms". In: *Phys. D* 60.1-4, pp. 259–268 [cit. on pp. vii, 5, 32].

Russakovsky, Olga et al. [Dec. 2015]. "ImageNet Large Scale Visual Recognition Challenge". In: *Int. J. Comput. Vis.* 115.3, pp. 211–252. ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y [cit. on p. 66].

Salim, Adil, Dmitry Koralev, and Peter Richtarik [2019]. "Stochastic Proximal Langevin Algorithm: Potential Splitting and Nonasymptotic Rates". In: *Adv. in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., pp. 6649–6661. URL: http://papers.nips.cc/paper/8891-stochastic-proximal-langevin-algorithm-potential-splitting-and-nonasymptotic-rates.pdf [cit. on pp. 13, 31, 39, 40, 43].

Salvatier, John, Thomas V. Wiecki, and Christopher Fonnesbeck [Apr. 6, 2016]. "Probabilistic Programming in Python Using PyMC3". In: *PeerJ Computer Science* 2, e55. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.55 [cit. on p. 47].

Shamis, Pavel et al. [Aug. 2015]. "UCX: An Open Source Framework for HPC Network APIs and Beyond". In: *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, pp. 40–43. DOI: 10.1109/HOTI.2015.13 [cit. on pp. 18, 51].

Simsekli, Umut et al. [July 3, 2018]. "Asynchronous Stochastic Quasi-Newton MCMC for Non-Convex Optimization". In: *Proceedings of the 35th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 4674–4683. URL: https://proceedings.mlr.press/v80/simsekli18a.html [cit. on p. 23].

Sohl-Dickstein, Jascha et al. [July 2015]. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *Proc. Int. Conf. Mach. Learning (ICML)*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 2256–2265. URL: https://proceedings.mlr.press/v37/sohl-dickstein15.html [cit. on p. 7].

Štrumbelj, Erik et al. [Feb. 2024]. "Past, Present and Future of Software for Bayesian Inference". In: *Statistical Science* 39.1, pp. 46–61. ISSN: 0883-4237, 2168-8745. DOI: 10.1214/23-STS907 [cit. on p. 46].

Terenin, Alexander, Daniel Simpson, and David Draper [Feb. 29, 2020]. *Asynchronous Gibbs Sampling*. arXiv: 1509.08999 [stat]. URL: http://arxiv.org/abs/1509.08999 [cit. on pp. 22, 83].

Thouvenin, Pierre-Antoine, Audrey Repetti, and Pierre Chainais [Oct. 2024]. "A Distributed Split-Gibbs Sampler with Hypergraph Structure for High-Dimensional Inverse Problems". In: *J. Comput. and Graph. Stat.* 33.3, pp. 814–832. DOI: 10.1080/10618600.2023.2282501 [cit. on pp. viii, 12, 22, 24, 27, 29, 32, 39, 43].

Tzikas, Alexandros E. et al. [Jan. 29, 2024]. *Distributed Markov Chain Monte Carlo Sampling Based on the Alternating Direction Method of Multipliers*. arXiv: 2401.15838 [stat]. URL: http://arxiv.org/abs/2401.15838. Pre-published [cit. on p. 21].

Ulyanov, Dmitry, Andrea Vedaldi, and Victor Lempitsky [July 2020]. "Deep Image Prior". In: *Int J Comput Vis* 128.7, pp. 1867–1888. ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-020-01303-4. arXiv: 1711.10925 [cs, stat]. URL: http://arxiv.org/abs/1711.10925 [visited on 03/30/2023] [cit. on p. 7].

Van Dyk, David A and Xiao-Li Meng [Mar. 1, 2001]. "The Art of Data Augmentation". In: *Journal of Computational and Graphical Statistics* 10.1, pp. 1–50. ISSN: 1061-8600. DOI: 10.1198/10618600152418584 [cit. on p. 14].

Venkatakrishnan, Singanallur V., Charles A. Bouman, and Brendt Wohlberg [Dec. 2013]. "Plug-and-Play Priors for Model Based Reconstruction". In: *Proc. IEEE Glob. Conf. on Sig. and Inf. Processing (GlobalSIP)*. 2013 IEEE Global Conference on Signal and Information Processing, pp. 945–948. DOI: 10.1109/GlobalSIP.2013.6737048. URL: https://ieeexplore.ieee.org/document/6737048/?arnumber=6737048 [visited on 11/12/2024] [cit. on p. 9].

Vetterli, Martin, Jelena Kovačević, and Vivek K. Goyal [2014]. *Foundations of Signal Processing*. Cambridge: Cambridge University Press. ISBN: 978-1-107-03860-8 [cit. on pp. 36, 37].

Vono, Maxime, Nicolas Dobigeon, and Pierre Chainais [Mar. 15, 2019]. "Split-and-Augmented Gibbs Sampler–Application to Large-Scale Inference Problems". In: *IEEE Trans. Signal Process.* 67.6, pp. 1648–1661. ISSN: 1053-587X, 1941-0476. DOI: 10.1109/TSP.2019.2894825. arXiv: 1804.05809 [eess, stat] [cit. on pp. 15, 39].

— [2021]. "Asymptotically Exact Data Augmentation: Models, Properties, and Algorithms". In: *J. Comput. and Graph. Stat.* 30.2, pp. 335–348. ISSN: 1061-8600. DOI: 10.1080/10618600.2020.1826954 [cit. on pp. viii, 14, 15, 27].

Vono, Maxime, Daniel Paulin, and Arnaud Doucet [2022]. "Efficient MCMC Sampling with Dimension-Free Convergence Rate using ADMM-type Splitting". In: *J. Mach. Learning Research* 23.25, pp. 1–69. URL: http://jmlr.org/papers/v23/20-357.html [cit. on p. 15].

Wang, Z. et al. [Apr. 2004]. "Image Quality Assessment: From Error Visibility to Structural Similarity". In: *IEEE Trans. Image Process.* 13.4, pp. 600–612. ISSN: 1057-7149. DOI: 10.1109/TIP.2003.819861. URL: http://ieeexplore.ieee.org/document/1284395/ [visited on 11/20/2024] [cit. on p. 67].

Wang, Zhou and Alan C. Bovik [Jan. 2009]. "Mean Squared Error: Love It or Leave It? A New Look at Signal Fidelity Measures". In: *IEEE Signal Processing Magazine* 26.1, pp. 98–117. ISSN: 1558-0792. DOI: 10.1109/MSP.2008.930649 [cit. on p. 67].

Welling, Max and Yee Whye Teh [2011]. "Bayesian learning via stochastic gradient langevin dynamics". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, pp. 681–688. ISBN: 9781450306195 [cit. on p. 13].

Wen, Bihan et al. [Jan. 2023a]. "Physics-Driven Machine Learning for Computational Imaging". In: *IEEE Signal Process. Mag.* 40.1, pp. 28–30. ISSN: 1558-0792. DOI: 10.1109/MSP.2022.3222888. URL: https://ieeexplore.ieee.org/document/10004832 [visited on 06/04/2025] [cit. on p. 3].

— [Mar. 2023b]. "Physics-Driven Machine Learning for Computational Imaging: Part 2". In: *IEEE Signal Process. Mag.* 40.2, pp. 13–15. ISSN: 1558-0792. DOI: 10.1109/MSP.2023.3236492. URL: https://ieeexplore.ieee.org/document/10054197 [visited on 06/04/2025] [cit. on p. 3].

Wu, Zihui et al. [Dec. 16, 2024]. "Principled Probabilistic Imaging Using Diffusion Models as Plug-and-Play Priors". In: *Adv. in Neural Information Processing Systems* 37, pp. 118389–118427. URL: https://proceedings.neurips.cc/paper_files/paper/2024/hash/d65c4ce22241138c1784ff753d4c746c-Abstract-Conference.html [visited on 06/01/2025] [cit. on p. 8].

Xifara, T. et al. [Aug. 1, 2014]. "Langevin Diffusions and the Metropolis-adjusted Langevin Algorithm". In: *Statistics & Probability Letters* 91, pp. 14–19. ISSN: 0167-7152. DOI: 10.1016/j.spl.2014.04.002 [cit. on p. 13].

Xu, Xingyu and Yuejie Chi [2024]. "Provably Robust Score-Based Diffusion Posterior Sampling for Plug-and-Play Image Reconstruction". In: *Adv. in Neural Information Processing Systems*. URL: https://openreview.net/forum?id=SLnsoaY4u1 [cit. on p. 8].

Zhang, Juan, Junping Yin, and Ruili Wang [2020]. "Basic Framework and Main Methods of Uncertainty Quantification". In: *Mathematical Problems in Engineering* 2020.1, p. 6068203. ISSN: 1563-5147. DOI: 10.1155/2020/6068203 [cit. on p. 4].

Zhang, Kai et al. [July 2017]. "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising". In: *IEEE Trans. Image Process.* 26.7, pp. 3142–3155. ISSN: 1941-0042. DOI: 10.1109/TIP.2017.2662206. URL: https://ieeexplore.ieee.org/document/7839189 [cit. on pp. ix, 33, 34, 66].

Zhang, Kai et al. [2021]. "Plug-and-Play Image Restoration With Deep Denoiser Prior". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 44.10, pp. 6360–6376. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2021.3088914. URL: https://ieeexplore.ieee.org/abstract/document/9454311 [cit. on pp. ix, 9, 33, 35, 66].

Zheng, Hongkai et al. [2025]. *InverseBench: Benchmarking Plug-and-Play Diffusion Priors for Inverse Problems in Physical Sciences*. URL: https://arxiv.org/abs/2503.11043. preprint [cit. on p. 8].

Appendices

A Running Multi-GPU scripts in Python

In the simulations, each process is bound to a dedicated GPU for the entire run. Although not strictly required for execution, but essential for distributed efficiency, multiple processes are not assigned to the same GPU. This prevents *contention*, where processes would compete for the GPU's limited memory and computational resources, potentially slowing performance. By granting exclusive access to each GPU, context switching and memory conflicts are eliminated. As a result, the number of processes is typically limited to the number of available CUDA-capable GPUs. An example of a multi-GPU initialization script using `mpi4py` and `CuPy` is shown in Listing 8.

To enable CUDA-aware communication and ensure optimal performance in multi-GPU configurations, the simulations are launched with `mpirun` using `UCX` as the communication layer and disabling certain caching mechanisms that may interfere with GPU memory transfers. For example, to run a simulation on two MPI processes with `OpenMPI` and GPU-direct support:

```
mpirun
-x OMPI_MCA_pml=ucx
-x OMPI_MCA_osc=ucx
-x OMPI_MCA_opal_cuda_support=true
-x UCX_MEMTYPE_CACHE=n
-np 2 python script.py
```

In this command:

- `OMPI_MCA_pml=ucx`, `OMPI_MCA_osc=ucx`: select UCX for point-to-point and one-sided communications.
- `OMPI_MCA_opal_cuda_support=true`: enable direct transfers from/to GPU

```python
from mpi4py import MPI
import cupy as cp

comm = MPI.COMM_WORLD  # MPI communicator
size = comm.Get_size() # total number of processes
rank = comm.Get_rank() # rank of the current process

cp.cuda.Device(rank).use() # bind process to GPU of index `rank`
x = cp.empty((32, 32))     # array created on this GPU
```

Listing 8: Example of a multi-GPU Python script initialization. Each process binds to a dedicated GPU using its rank as the index, so that process 0 uses GPU 0, process 1 uses GPU 1, and so on. All array creations and computations are performed on the assigned GPU.

memory.

- `UCX_MEMTYPE_CACHE=n`: disable UCX memory type caching to avoid stale GPU pointer issues.

- `-np <B>`: number of MPI processes to launch.

B Reproducibility and RnG Handling on GPU

Reproducibility in a multi-GPU, multi-process environment hinges on two properties: (i) processes must draw from *uncorrelated* pseudo-random sequences, and (ii) all RNG states must be *serializable* for checkpointing and exact resumption of long MCMC runs. In our implementation we rely exclusively on `torch.Generator` for GPU-side randomness. Two pragmatic reasons motivated this choice:

- While CuPy can draw random numbers on the GPU, we did not find a straightforward, well-documented pathway to persist and restore per-process GPU RNG states across runs.

- PyTorch exposes generators with convenient APIs, enabling compact checkpointing. In current PyTorch releases, CUDA RNGs are counter-based (Philox) and can be resumed from a seed and an offset, represented by two integers, which are easier to store than serializing the entire random generator state.

```python
combined = f"{rank}{base_seed}" # combine rank and base seed
hashed_combined = hashlib.sha256(combined.encode()).hexdigest()
local_seed = int(hashed_combined, 16) % (2**32)
rng = torch.Generator(device=f"cuda:{gpu_id}")
rng.manual_seed(local_seed)
```

Listing 9: GPU-specific random number generator initialization.

B.1 Generating Uncorrelated Random Sequences

Concretely, each MPI rank is bound to a single CUDA device, and we create a dedicated `torch.Generator` for that device. To ensure that different processes draw from disjoint pseudo-random sequences, we deterministically derive each device's seed from a shared global base seed and the process's rank. The two values are concatenated, hashed with SHA-256, and reduced to a 32-bit integer suitable for `manual_seed()` (see Listing 9). This hashing scheme should yield (with high probability) collision-free seeds and emulates the safer `spawn` feature provided by `numpy.random.Generator.spawn`, not currently available in CuPy.

B.2 Saving and Loading Random States

In our distributed setting, instead of persisting RNG states which can be more complex to serialize and manage, only the *seed* and *offset* are stored. These are sufficient to resume a run exactly, provided that the same generator configuration is re-created on the same device. The seed sets the starting point of the pseudo-random sequence. The offset marks the advancement along that sequence.

Because the offset is defined relative to the seed, both values must come from the same checkpoint: mixing a seed from one run with an offset from another yields an unrelated point in the random sequence. This representation is compact (a pair of integers) which makes it easy to save and load.

Remark: Even with fixed RNG states, CUDA kernel-level non-determinism (e.g., atomic reductions, convolution algorithms) can cause bitwise differences. When bitwise repeatability matters, some deterministic guards can be applied:

```python
torch.use_deterministic_algorithms(True)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

Note that these settings may reduce performance and do not cover every operator. However, no noticeable impact on run time was observed in our experiments, but this may vary depending on the GPU architecture and the specific operations.

Distributed Plug-and-Play MCMC algorithms for high-dimensional Bayesian inference

Abstract: Markov Chain Monte Carlo (MCMC) algorithms are standard approaches to solve imaging inverse problems and quantify estimation uncertainties, which are essential in absence of ground-truth data. To improve estimation quality, Plug-and-Play MCMC algorithms, such as PnP-ULA, have been recently developed to accommodate priors encoded by a denoising neural network. Designing scalable samplers for high-dimensional imaging inverse problems remains a challenge: drawing and storing high-dimensional samples can be prohibitive, especially for high-resolution images. To address this issue, the work presented in this thesis proposes a distributed sampler based on approximate data augmentation and Plug-and-Play sampling algorithms to solve very large problems. The proposed sampler uses a denoising convolutional neural network encoded by few parameters combined with Langevin-based transition kernels, to efficiently exploit multiple GPUs on a Single Program Multiple Data architecture. A modular library has been introduced to ease the development of distributed MCMC algorithms. It leverages the generic structure of MCMC to allow seamless integration of different samplers, while emphasizing communication efficiency and adaptability to multi-GPU environments. Reconstruction performance and scalability are evaluated on several imaging problems. Communication and computation overheads due to the denoiser are carefully discussed. The proposed distributed approach noticeably combines three very precious qualities: it improves scalability, without altering the estimation quality obtained from rich deep priors, while allowing uncertainty quantification.

Keywords: Bayesian inference, Markov Chain Monte Carlo, Plug-and-Play, Denoising neural networks, Distributed computing, Imaging inverse problems, Uncertainty quantification.

Algorithmes MCMC Plug-and-Play distribués pour l'inférence bayésienne en grande dimension

Résumé : Les algorithmes de Monte Carlo par chaînes de Markov (MCMC) sont des approches standard pour résoudre les problèmes inverses d'imagerie et quantifier les incertitudes d'estimation, qui sont essentielles en l'absence de données de validation. Afin d'améliorer la qualité des estimations, des algorithmes MCMC Plug-and-Play, tels que PnP-ULA, ont récemment été développés pour prendre en compte les a priori encodés par un réseau neuronal de débruitage. La conception d'échantillonneurs pouvant passer à l'échelle pour les problèmes inverses d'imagerie à haute dimension reste un défi : la génération et le stockage d'échantillons en grande dimension peuvent être prohibitifs, en particulier pour les images à haute résolution. Pour remédier à ce problème, les travaux présentés dans cette thèse proposent un échantillonneur distribué basé sur l'augmentation de données approchée et des algorithmes d'échantillonnage Plug-and-Play afin de résoudre des problèmes de très grande taille. L'échantillonneur proposé utilise un réseau neuronal convolutif de débruitage encodé par un faible nombre de paramètres, combiné à des noyaux de transition basés sur Langevin, afin d'exploiter efficacement plusieurs GPU sur une architecture Single Program Multiple Data. Une librairie modulaire a été introduite pour faciliter le développement des algorithmes MCMC distribués. Elle tire parti de la structure générique des algorithmes MCMC pour permettre une intégration plus simple de différents échantillonneurs, tout en mettant l'accent sur l'efficacité de la communication et l'adaptabilité à des environnements multi-GPU. Les performances de reconstruction et de passage à l'échelle sont évaluées sur plusieurs problèmes d'imagerie. Les coûts de communication et de calcul liés au débruiteur sont comparés et discutés. L'approche distribuée proposée combine de manière notable trois qualités très précieuses : elle permet de passer à l'échelle, sans altérer la qualité d'estimation obtenue grâce à des a priori profonds riches, et d'en quantifier l'incertitude.

Mots-clés : Inférence bayésienne, Monte Carlo par chaînes de Markov, Plug-and-Play, Réseaux de neurones débruiteurs, Calcul distribué, Problèmes inverses d'imagerie, Quantification d'incertitude.