# LEARNING PARAMETRIC INTEGRALS FROM INTEGRANDS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Parametric integration requires repeatedly evaluating families of integrals with varying integrands and integration domains. Learning a neural representation of such integrals can amortize this computation, but conventionally requires *integral supervision*, typically obtained by numerically integrating pointwise evaluations of the integrand. We introduce a neural representation that instead learns parametric integrals directly from pointwise *integrand supervision*. Our key idea is to decompose the integrand into a component that is constructed to integrate to zero and a second component that directly represents the integral. We realize the zero-integral component using the divergence of a neural field and handle parameterized integration domains through a change of variables. Once trained, our representation directly evaluates parametric integrals without numerical integration and remains differentiable with respect to its parameters. Across diverse applications such as filtering, rendering, and integral transforms, our approach achieves lower approximation error than supervision on Monte Carlo integral estimates under the same training-time budget, despite using fewer integrand evaluations overall.

## 1 INTRODUCTION

Many computational problems require repeatedly evaluating families of integrals with varying integrands and integration domains. We refer to this class of problems as *parametric integration*. Performing numerical integration from scratch for each parameter configuration can be prohibitively expensive, motivating learned approaches that amortize computation across subsequent queries (Müller et al., 2020; Lindell et al., 2021; Rubab et al., 2025). Let $f(\mathbf{x}; \mathbf{p})$ denote an integrand, where $\mathbf{x}$ is the continuous integration variable and $\mathbf{p}$ ranges over a continuous parameter space, and let $\Omega(\mathbf{q})$ denote an integration domain parameterized by $\mathbf{q}$. Our goal is to learn a representation of

$$I(\mathbf{p}, \mathbf{q}) = \int_{\Omega(\mathbf{q})} f(\mathbf{x}; \mathbf{p}) \, \mathrm{d}\mathbf{x} \tag{1}$$

that can be efficiently evaluated for different integrand and domain parameters $\mathbf{p}$ and $\mathbf{q}$. For this general setting, a straightforward approach is to learn the parametric integral from integral targets. In many settings, however, only pointwise evaluations of the integrand are readily available, so these targets must first be approximated numerically, for example using Monte Carlo integration to manufacture noisy targets for training (Robert et al., 1999; Leitao & Ráfales, 2025) (Fig. 1, left). This raises a natural question: *Can we instead learn a parametric integral directly from pointwise evaluations of its integrand, without ever requiring integral supervision?*

We answer this question in the affirmative. Our key idea is to design a neural representation that decomposes the integrand into two components with distinct roles. For a fixed integration domain, the first depends on both the integration variable $\mathbf{x}$ and the parameters $\mathbf{p}$, while the second depends only on $\mathbf{p}$. Building on the divergence theorem, we obtain the zero-integral component as the divergence of a neural field. Consequently, the second neural component directly represents the integral. Parameterized integration domains are incorporated through a change of variables, reducing them to the same formulation. The resulting representation can be trained entirely through pointwise integrand supervision (Fig. 1, right). Once trained, the model directly evaluates the corresponding integral for different integrand and domain parameters $\mathbf{p}$ and $\mathbf{q}$, without numerical integration at inference time. Moreover, the learned integral is differentiable with respect to $\mathbf{p}$ and $\mathbf{q}$, making these derivatives readily available through automatic differentiation.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

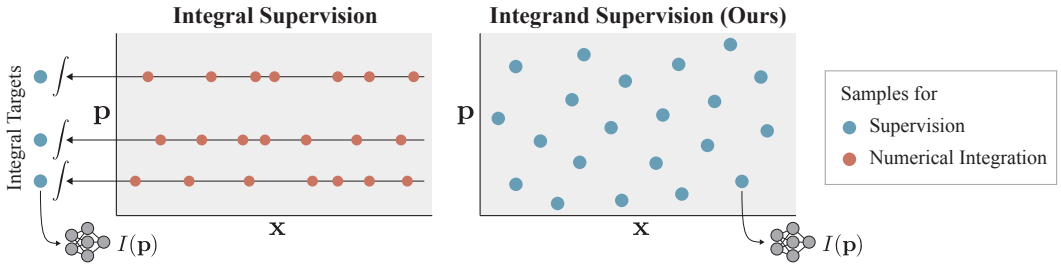



**Figure 1: Integral vs. integrand supervision.** We consider a family of integrands $f(\mathbf{x}; \mathbf{p})$, where $\mathbf{x}$ is the continuous integration variable and $\mathbf{p}$ ranges over a continuous parameter space, and seek to train a neural model of the corresponding parametric integral $I(\mathbf{p})$. Conventional integral supervision first samples the integrand at multiple $\mathbf{x}$ for each $\mathbf{p}$ and numerically integrates these samples to construct approximate integral targets for training the model (left). Our approach instead trains directly on individual integrand evaluations sampled jointly over the continuous $\mathbf{x}$ and $\mathbf{p}$ spaces (right), while still *directly representing the corresponding parametric integral* $I(\mathbf{p})$ *after training*. Our full formulation further supports parameterized integration domains $\Omega(\mathbf{q})$.

Across a diverse set of applications, we find that, under the same training-time budget, direct integrand supervision achieves substantially lower approximation error than supervision on Monte Carlo integral estimates. This advantage holds despite our method requiring more computation per training iteration and consequently using fewer integrand evaluations overall. While the quality of Monte Carlo targets, and consequently the resulting fit, strongly depends on the number of samples used to construct each target, our approach is considerably less sensitive to this quantity and maintains strong performance across a broad range of sample counts, particularly when samples are limited. We demonstrate these properties on parametric integration problems spanning linear and nonlinear filtering, rendering, and integral transforms, and further showcase the ability to differentiate the learned integrals with respect to their parameters.

Our main contributions are:

- We introduce a neural representation for parametric integrals that can be trained directly from pointwise *integrand supervision*, without requiring *integral supervision*.
- We devise a neural decomposition for parametric integration into zero-integral and integral components that accommodates parameterized integration domains.
- We demonstrate across diverse applications that, under matched training time, our approach achieves lower approximation error than Monte Carlo supervision while using fewer integrand evaluations overall.

All models, data, and source code will be released upon publication.

## 2 Related Work

**Neural representations.** Neural fields provide continuous, differentiable representations of signals and have become a general tool for representing complicated functions (Xie et al., 2022; Sitzmann et al., 2020). Since their derivatives are readily available through automatic differentiation, they are particularly well suited to problems that impose differential constraints or directly operate on signal derivatives, including PDE solving (Raissi et al., 2019), signal processing (Xu et al., 2022), and rendering (Nsampi et al., 2026). This convenience does not generally extend to integration: applications such as neural volume rendering still require numerical integration of the learned field at inference time (Mildenhall et al., 2020). Our work instead uses a neural field to represent the integral itself.

**Learning integrals.** Neural representations can be restricted to specially designed, analytically integrable function classes (*Analytic* in Tab. 1), at the cost of reduced expressivity that can limit their ability to represent complex integrands (Zhe-Zhao et al., 2006; Yan et al., 2013; Lloyd et al., 2020; Subr, 2021; Meister & Harada, 2025). Alternatively, neural representations of parametric integrals can be learned from numerically estimated integral targets (*Numerical Supervision* in Tab. 1 and Fig. 1, left) (Leitao & Ráfales, 2025; Yaldiz et al., 2025). However, the accuracy of these targets—and consequently the achievable fitting quality—depends on the number of samples used to construct

**Table 1:** Comparison of classes of learning-based approaches to integration. Our problem setting requires general integrand representations together with support for parametric integrals and general parameterized integration domains; methods that do not satisfy all three requirements are shown in gray. Derivative Order denotes the highest-order derivative of the representation evaluated during training, where $d_{\mathbf{x}}$ denotes the dimensionality of the integration variable.

| Method | Representation | Supervision | Param. Integral | Param. Domain | Derivative Order |
|---|---|---|---|---|---|
| Analytic | Restricted | Integrand | ✓ | ✓ | 0 |
| Antiderivative | General | Integrand | ✓ | Axis-aligned | $\propto d_{\mathbf{x}}$ |
| Zero-Integral | General | Integrand | × | × | 1 |
| Numerical Supervision | General | Integral | ✓ | ✓ | 0 |
| **Ours** | General | Integrand | ✓ | ✓ | 1 |

them, with accurate estimates generally requiring high sample counts (Robert et al., 1999). A third approach is to learn antiderivatives (*Antiderivative* in Tab. 1) (Rubab et al., 2025), allowing definite integrals to be evaluated efficiently from their values at the integration boundaries. A common strategy is to parameterize the antiderivative, differentiate it with respect to its inputs, and fit the resulting derivative model to function observations (Teichert et al., 2019; Lindell et al., 2021; Maître & Santos-Mateos, 2023; Nsampi et al., 2023; Li et al., 2024). In multiple dimensions, however, this construction naturally applies to axis-aligned integration domains and requires successive differentiation along each integration dimension, resulting in increasingly high-order derivatives that make optimization increasingly challenging as the integration dimensionality grows. In contrast, our approach retains the expressivity of general neural fields while requiring neither numerically estimated integral targets nor successive differentiation, making it applicable to complex integrands and higher-dimensional integration problems.

More generally, neural operators can learn integral operators by mapping observations of an input function to the corresponding operator output (Lu et al., 2021). This allows a learned integration operator to generalize across input functions, whereas our setting learns the integrals of a parameterized family directly from pointwise integrand supervision.

**Neural functions with known integrals.** Several learning-based approaches to numerical integration rely on neural functions whose integrals are known by construction. Normalizing flows (Rezende & Mohamed, 2015) provide expressive normalized functions with unit integral, a property exploited for importance sampling and control variates (Müller et al., 2019; Müller et al., 2020). In contrast, Stein (1972) operators provide a differential construction of functions with zero expectation under a given distribution. This property has been used for neural numerical integration by decomposing an integrand into a learned zero-expectation component and a scalar representing its integral (*Zero-Integral* in Tab. 1) (Ott et al., 2023). Specializing this principle to a uniform reference measure yields the divergence-based zero-integral construction underlying our method. We generalize the resulting decomposition from individual integrals to parametric families, turning the scalar integral estimate into a learned function over the parameter space, while accommodating parameterized integration domains.

## 3 METHOD

In this section, we first formalize the problem setting and notation in Sec. 3.1. We then develop our method gradually, starting from the simplest setting and successively introducing the components required for general parametric integration. In Sec. 3.2, we introduce our core decomposition for a single, non-parametric integral. We extend this formulation to parametric families of integrands with fixed integration domains in Sec. 3.3, before incorporating parameterized integration domains in Sec. 3.4 to obtain the complete formulation. We provide implementation details in Sec. 3.5.

### 3.1 PROBLEM FORMULATION

We consider parametric integrands $f \in (\mathbb{R}^{d_{\mathbf{x}}} \times \mathbb{R}^{d_{\mathbf{p}}}) \to \mathbb{R}^d$, where $\mathbf{x} \in \mathbb{R}^{d_{\mathbf{x}}}$ denotes the integration variable and $\mathbf{p} \in \mathbb{R}^{d_{\mathbf{p}}}$ the integrand parameters. We further let $\mathbf{q} \in \mathbb{R}^{d_{\mathbf{q}}}$ parameterize the integration domain $\Omega(\mathbf{q}) \subseteq \mathbb{R}^{d_{\mathbf{x}}}$. We assume that each $\Omega(\mathbf{q})$ is given by a known mapping $T_{\mathbf{q}} : [0,1]^{d_{\mathbf{x}}} \to \Omega(\mathbf{q})$

from the unit hypercube; we formalize the required properties of this mapping in Sec. 3.4. Together, these quantities define the parametric integral $I \in (\mathbb{R}^{d_\mathbf{p}} \times \mathbb{R}^{d_\mathbf{q}}) \to \mathbb{R}^d$ introduced in Eq. 1.

Our goal is to learn a neural field $I_\phi \in (\mathbb{R}^{d_\mathbf{p}} \times \mathbb{R}^{d_\mathbf{q}}) \to \mathbb{R}^d$ with trainable parameters $\phi$, such that $I_\phi(\mathbf{p}, \mathbf{q}) \approx I(\mathbf{p}, \mathbf{q})$. Crucially, we consider the common practical setting in which the integrand $f$ can be evaluated pointwise, while values of $I$ are not directly available. Training supervision is therefore limited to evaluations $f(\mathbf{x}; \mathbf{p})$ of the integrand. Our objective is to learn $I_\phi$ directly from these pointwise observations, avoiding the need to first approximate $I$ numerically to obtain training targets.For ease of exposition, we assume scalar-valued outputs in the following derivations; vector-valued outputs are handled independently per dimension by using multiple output channels in our models.

### 3.2 Learning a Single Integral

We begin with the simplest setting of a single integrand $f : \Omega \to \mathbb{R}$ over the unit hypercube $\Omega = [0, 1]^{d_\mathbf{x}}$. Given pointwise evaluations of $f$, we seek to recover its integral $I = \int_\Omega f(\mathbf{x}) \, \mathrm{d}\mathbf{x}$. This setting allows us to introduce the core construction underlying our method before extending it to parametric integrands and integration domains. Rather than estimating $I$ through numerical integration, we seek a decomposition from which $I$ can be read off directly. We achieve this by separating the integral from the spatial variation of $f$. Specifically, we decompose $f$ into a function whose integral vanishes and a constant carrying the integral,

$$f(\mathbf{x}) = f_0(\mathbf{x}) + I, \qquad \int_\Omega f_0(\mathbf{x}) \, \mathrm{d}\mathbf{x} = 0. \tag{2}$$

Since $\Omega$ has unit volume, integrating the decomposition leaves only the constant $I$. Hence, constructing the first component such that its integral vanishes isolates the desired integral in the second component. To construct such a zero-integral function, we build on the divergence theorem. For any differentiable vector field $\mathbf{g} : \Omega \to \mathbb{R}^{d_\mathbf{x}}$ with vanishing normal component on the boundary $\partial\Omega$ of $\Omega$, the divergence theorem gives

$$\int_\Omega \nabla_\mathbf{x} \cdot \mathbf{g}(\mathbf{x}) \, \mathrm{d}\mathbf{x} = 0. \tag{3}$$

We enforce this boundary condition by tapering $\mathbf{g}$ with a fixed function $w : \Omega \to \mathbb{R}$ that vanishes on $\partial\Omega$. The divergence of the resulting field,

$$f_0(\mathbf{x}) = \nabla_\mathbf{x} \cdot [w(\mathbf{x})\mathbf{g}(\mathbf{x})] \, , \tag{4}$$

consequently provides the desired zero-integral component (Ott et al., 2023).

We now parameterize $\mathbf{g}$ using a neural field $\mathbf{g}_\theta : \Omega \to \mathbb{R}^{d_\mathbf{x}}$ with trainable parameters $\theta$, and parameterize the unknown integral by a separate learned scalar $c$. Collectively, we denote the trainable parameters by $\Theta = (\theta, c)$. This yields

$$f_\Theta(\mathbf{x}) = \nabla_\mathbf{x} \cdot [w(\mathbf{x})\mathbf{g}_\theta(\mathbf{x})] + c. \tag{5}$$

The divergence is a sum of first-order derivatives, so the required derivative order remains fixed as the integration dimensionality increases, unlike the successive differentiation required by antiderivative-based approaches (last column in Tab. 1). In practice, we approximate these derivatives using central finite differences (Li et al., 2023; Nsampi et al., 2023), which we empirically find to perform better than automatic differentiation in most cases.

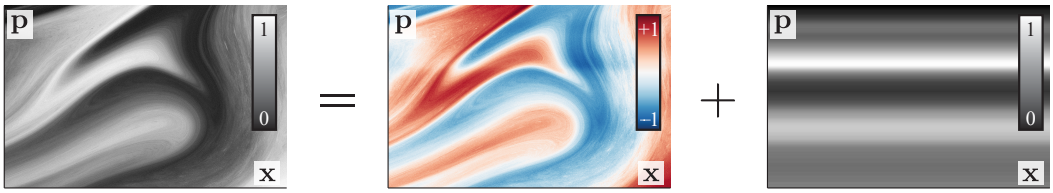
We fit this representation directly to pointwise evaluations of the integrand using

$$\mathcal{L}(\Theta) = \mathbb{E}_{\mathbf{x} \sim \mathcal{U}(\Omega)} \left[ |f_\Theta(\mathbf{x}) - f(\mathbf{x})|^2 \right]. \tag{6}$$

Importantly, this objective requires only evaluations of $f$ and no knowledge of $I$. Since the first component integrates to zero by construction, fitting $f_\Theta$ to $f$ directly learns $c \approx I$, which can be read off after training without numerical integration.

### 3.3 Learning Parametric Integrals

We now extend the previous construction from a single integral to a parametric family of integrands $f : (\Omega \times \mathbb{R}^{d_\mathbf{p}}) \to \mathbb{R}$, while keeping the integration domain $\Omega$ fixed to the unit hypercube. Our goal




$f(\mathbf{x};\mathbf{p})$ $f_0(\mathbf{x};\mathbf{p})$ $I(\mathbf{p})$

**Figure 2: Parametric zero-integral decomposition.** We decompose a parametric integrand $f(\mathbf{x};\mathbf{p})$ into a spatially varying component $f_0(\mathbf{x};\mathbf{p})$ whose integral over $\mathbf{x}$ vanishes for every $\mathbf{p}$, and a component $I(\mathbf{p})$ representing the corresponding integral. The first component captures the variation of the integrand while integrating to zero along the integration dimension, whereas the second is constant along $\mathbf{x}$ and varies only with $\mathbf{p}$, directly representing the parametric integral of interest. Both components emerge jointly by fitting our neural representation to pointwise evaluations of the integrand.

is no longer to recover a single scalar $I$, but the function $I(\mathbf{p})$ that maps each integrand parameter $\mathbf{p} \in \mathbb{R}^{d_\mathbf{p}}$ to its corresponding integral. The key step is to make both components of the decomposition functions of the integrand parameters $\mathbf{p}$. We therefore write

$$f(\mathbf{x};\mathbf{p}) = f_0(\mathbf{x};\mathbf{p}) + I(\mathbf{p}), \qquad \int_\Omega f_0(\mathbf{x};\mathbf{p})\,\mathrm{d}\mathbf{x} = 0. \tag{7}$$

For every $\mathbf{p}$, the first component captures the spatial variation of the corresponding integrand while integrating to zero, leaving the second component to represent its integral (Fig. 2).

We realize this decomposition by conditioning the neural vector field on the integrand parameters, $\mathbf{g}_\theta : (\Omega \times \mathbb{R}^{d_\mathbf{p}}) \to \mathbb{R}^{d_\mathbf{x}}$, and, crucially, replacing the learned scalar $c$ from Sec. 3.2 by a second neural field $I_\phi : \mathbb{R}^{d_\mathbf{p}} \to \mathbb{R}$ with trainable parameters $\phi$. Collecting the parameters of both neural fields as $\Theta = (\theta, \phi)$ yields

$$f_\Theta(\mathbf{x};\mathbf{p}) = \nabla_\mathbf{x} \cdot [w(\mathbf{x})\mathbf{g}_\theta(\mathbf{x};\mathbf{p})] + I_\phi(\mathbf{p}). \tag{8}$$

Importantly, the divergence is taken only with respect to the integration variable $\mathbf{x}$, so the first component integrates to zero for every fixed $\mathbf{p}$, while the neural field $I_\phi$ directly represents the resulting family of integrals. As before, the first component thus serves as a vehicle for learning the second component, which is the quantity of interest.

We jointly optimize both neural fields from pointwise integrand evaluations using

$$\mathcal{L}(\Theta) = \mathbb{E}_{\mathbf{p},\mathbf{x}}\left[\left|f_\Theta(\mathbf{x};\mathbf{p}) - f(\mathbf{x};\mathbf{p})\right|^2\right]. \tag{9}$$

As before, supervision consists only of pointwise integrand evaluations, now sampled over the product of the integration and parameter spaces, without requiring integral targets. After training, the complete parametric integral is directly available as $I_\phi(\mathbf{p}) \approx I(\mathbf{p})$ without numerical integration.

### 3.4 Parameterized Integration Domains

So far, we have assumed that the integration domain $\Omega$ is fixed to the unit hypercube. We now consider the parameterized domains $\Omega(\mathbf{q})$ introduced in Sec. 3.1. Recall that $T_\mathbf{q} : [0,1]^{d_\mathbf{x}} \to \Omega(\mathbf{q})$ maps the unit hypercube to the integration domain associated with $\mathbf{q}$. We require $T_\mathbf{q}$ to be a diffeomorphism with a tractable Jacobian determinant, a setting that covers the diverse integration domains considered in our evaluation. The key idea is to use this mapping to pull every integral back to the same unit hypercube, absorbing the varying domain into the integrand and thereby reducing the problem to the fixed-domain setting of Sec. 3.3. Specifically, a change of variables gives

$$I(\mathbf{p},\mathbf{q}) = \int_{\Omega(\mathbf{q})} f(\mathbf{x}';\mathbf{p})\,\mathrm{d}\mathbf{x}' = \int_{[0,1]^{d_\mathbf{x}}} \underbrace{f(T_\mathbf{q}(\mathbf{x});\mathbf{p})\,\left|\det \nabla_\mathbf{x} T_\mathbf{q}(\mathbf{x})\right|}_{\tilde{f}(\mathbf{x};\mathbf{p},\mathbf{q})}\,\mathrm{d}\mathbf{x}. \tag{10}$$

The Jacobian determinant accounts for the local change in volume induced by $T_\mathbf{q}$. The integration domain is now fixed, with all dependence on $\mathbf{q}$ absorbed into the transformed integrand $\tilde{f}$. We can

therefore treat $\mathbf{p}$ and $\mathbf{q}$ jointly as parameters of $\tilde{f}$ and directly apply the construction from Sec. 3.3. Conditioning both neural fields on these parameters yields

$$\tilde{f}_\Theta(\mathbf{x}; \mathbf{p}, \mathbf{q}) = \nabla_\mathbf{x} \cdot [w(\mathbf{x}) \mathbf{g}_\theta(\mathbf{x}; \mathbf{p}, \mathbf{q})] + I_\phi(\mathbf{p}, \mathbf{q}). \tag{11}$$

As before, the first component integrates to zero over the unit hypercube for every fixed $(\mathbf{p}, \mathbf{q})$, while the second component directly represents the full parametric integral introduced in Eq. 1, with $I_\phi(\mathbf{p}, \mathbf{q}) \approx I(\mathbf{p}, \mathbf{q})$.

Since $T_\mathbf{q}$ and its Jacobian determinant are known, $\tilde{f}$ remains available through pointwise evaluations of $f$. We jointly optimize both neural fields using

$$\mathcal{L}(\Theta) = \mathbb{E}_{\mathbf{q},\mathbf{p},\mathbf{x}} \left[ \left| \tilde{f}_\Theta(\mathbf{x}; \mathbf{p}, \mathbf{q}) - \tilde{f}(\mathbf{x}; \mathbf{p}, \mathbf{q}) \right|^2 \right]. \tag{12}$$

As before, supervision consists only of pointwise integrand evaluations, now sampled jointly over the integration, integrand-parameter, and domain-parameter spaces, without requiring integral targets. The fixed-domain formulation of Sec. 3.3 is recovered when $T_\mathbf{q}$ is the identity mapping. Thus, even for parameterized integration domains, the complete parametric integral is learned entirely from pointwise integrand supervision.

### 3.5 Implementation Details

For the boundary taper, we use the normalized polynomial window $w(\mathbf{x}) = \prod_{i=1}^{d_\mathbf{x}} \max\big(4\, x_i\, (1 - x_i),\ 0\big)$, which is smooth on the unit hypercube and vanishes on its boundary. We parameterize the fields $\mathbf{g}_\theta$ and $I_\phi$ as coordinate-based MLPs (widths 256 and 128 respectively, two hidden layers each) with Swish activations (Ramachandran et al., 2017) and an axis-aligned Fourier-feature positional encoding with eight log-spaced frequency bands (Mildenhall et al., 2020; Tancik et al., 2020). The central finite differences used to estimate the divergence (Sec. 3.2) use a step $\epsilon$ tuned per application. During training, the integrand is sampled uniformly at random. We optimize with Adam (Kingma & Ba, 2015) at a base learning rate of $2 \times 10^{-3}$, cosine-annealed to $10^{-5}$ over the training budget.

## 4 Evaluation

We first introduce our evaluation protocol and experimental problems in Sec. 4.1. We then present quantitative and qualitative results across these problems in Sec. 4.2, followed by the analysis of the key components and design choices of our method in Sec. 4.3.

### 4.1 Experimental Setup

**Evaluation Protocol.** We consider learning models for a broad range of parametric integration problems and compare our approach against numerical integral supervision, the only baseline in Tab. 1 that supports the full problem setting considered here, including general integrands, parametric integrals, and parameterized integration domains.

**Table 2:** Dimensionalities of the parametric integration problems considered in our evaluation.

| Application | $d$ | $d_\mathbf{x}$ | $d_\mathbf{p}$ | $d_\mathbf{q}$ |
|---|---|---|---|---|
| Gaussian Filtering | 3 | 2 | 3 | — |
| Motion Blur | 3 | 1 | 4 | — |
| Bilateral Filtering | 3 | 2 | 4 | — |
| Rendering | 3 | 2 | 4 | — |
| Fourier Transform | 2 | 1 | 1 | — |
| Radon Transform | 1 | 1 | — | 2 |

For a controlled comparison, we impose the same wall-clock training-time limit on integral supervision and our approach. Within this limit, we systematically vary the number of integrand evaluations performed in each training iteration. Our approach uses all of these evaluations directly as pointwise supervision (Fig. 1, right). Integral supervision instead first samples a batch of parameter configurations $(\mathbf{p}, \mathbf{q})$. For each configuration, it samples multiple integration variables $\mathbf{x}$ and combines the corresponding integrand evaluations into a Monte Carlo estimate of $I(\mathbf{p}, \mathbf{q})$, yielding one approximate integral target (Fig. 1, left). Thus, for a fixed number of integrand evaluations per training iteration, integral supervision trades off more integral targets with fewer samples per target against fewer, more accurately estimated targets. For every problem

and per-iteration sample count, we sweep this allocation and report the best-performing configuration for integral supervision. To ensure a fair comparison of the learned integrals, we match the capacity of the neural field trained using the integral-supervision baseline to that of our integral neural field $I_\phi$. Our approach additionally learns the zero-integral component, which serves as a vehicle for learning $I_\phi$ from pointwise integrand supervision and is discarded after training. This auxiliary neural field and the evaluation of its divergence increase the computational cost of each training iteration, but do not increase the capacity of the integral representation being compared. Consequently, within the same wall-clock training time, our approach completes fewer training iterations and therefore performs fewer integrand evaluations in total.

For each application, we perform our experiments across 20 different parametric integrands and evaluate approximation quality on a set of parameter configurations against high-accuracy reference integrals computed using dense numerical sampling. We report PSNR, SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018) for image outputs, and MSE for all other outputs. More details on the experimental setup are provided in the appendix (Sec. A.1).

**Filtering.** Many filtering operations can be expressed as weighted integrals of a signal $s$,

$$I(\mathbf{p}) = \int_\Omega \underbrace{s(\mathbf{x})\, k(\mathbf{x};\mathbf{p})}_{f(\mathbf{x};\mathbf{p})}\, \mathrm{d}\mathbf{x}, \tag{13}$$

where $k$ denotes a filter kernel and $\mathbf{p}$ jointly parameterizes the output location and filter properties. We consider three instances spanning linear and nonlinear filtering.

*Gaussian filtering.* We take a fixed image as the input and learn Gaussian filtering jointly over $\mathbf{p}\in\mathbb{R}^3$, the spatial location and kernel variance. Integration is over $\Omega=\mathbb{R}^2$.

*Motion blur.* We treat a video as a 3D space-time signal and filter it temporally using a box kernel centered at the queried time. The queried space-time location and the filter width form the parameter $\mathbf{p} \in \mathbb{R}^4$, and integration is over time, $\Omega=\mathbb{R}$.

*Bilateral filtering.* We further consider nonlinear bilateral filtering, where the filter weights depend on both spatial distance and signal similarity, so the parameter $\mathbf{p} \in \mathbb{R}^4$ includes 2D query location, 1D filter scale, and 1D range scale. Integration is again over $\Omega=\mathbb{R}^2$. We account for normalization by filtering in homogeneous coordinates.

**Rendering.** We consider scene-independent image-based lighting: given an environment map $L$, we learn its continuous pre-filtered representation over reflection direction $\mathbf{r}$ and surface shininess $\alpha$. In contrast to the filtering integrals over the spatial domain, rendering integrates over directions. Using a Phong reflectance lobe, the shading integral is

$$I(\mathbf{r},\alpha) = \int_{\mathbb{S}^2} L(\omega)\, \max(0,\, \mathbf{r}\cdot\omega)^\alpha\, \mathrm{d}\omega, \tag{14}$$

where the integration of the incident direction $\omega$ is over the fixed domain $\Omega=\mathbb{S}^2$, the unit sphere of directions in $\mathbb{R}^3$. The reflection direction and shininess act as the integrand parameters $\mathbf{p}=(\mathbf{r},\alpha)$. Our model is trained per environment map, and can then be used to shade arbitrary scenes.

**Integral Transforms.** We further consider integral transforms, which map an input function to a new representation through integration. We evaluate two common transforms that exercise complementary aspects of our formulation: the Fourier transform parameterizes the integrand, while the Radon transform parameterizes the integration domain.

*Fourier Transform.* For a one-dimensional signal $s$, we learn its continuous Fourier transform

$$I(\xi) = \int_{\mathbb{R}} s(\mathbf{x})\, e^{-2\pi \mathrm{i}\, \xi \mathbf{x}}\, \mathrm{d}\mathbf{x}, \tag{15}$$

where $\mathbf{x}$ is in the fixed 1D domain $\Omega=\mathbb{R}$, and the integrand parameter $\mathbf{p} = \xi \in \mathbb{R}$ is the continuous frequency. We represent the complex-valued transform using two real output channels.

*Radon Transform.* For a two-dimensional signal $s$, we learn its continuous Radon transform which integrates the signal along a line parameterized by its orientation $\theta$ and offset $\tau$,

$$I(\theta,\tau) = \int_{\Omega(\theta,\tau)} s\big(\mathbf{x}\big)\, \mathrm{d}l(\mathbf{x}). \tag{16}$$

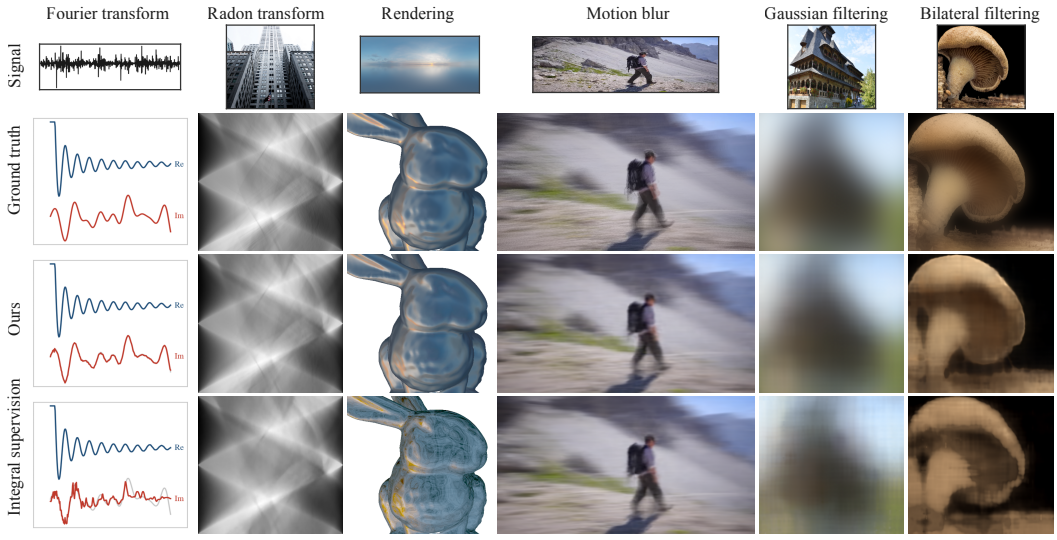



**Figure 3:** Qualitative results at a budget of 2048 integrand evaluations per training iteration. For each application (columns), we show the input signal, the ground-truth integral, and the reconstructions from our method and from integral supervision (rows).

Here, the integration of $\mathbf{x}$ is over the domain $\Omega(\theta, \tau) = \{\tau\,\hat{\mathbf{n}}(\theta) + t\,\hat{\mathbf{d}}(\theta)\}$, which is the line with unit normal $\hat{\mathbf{n}}(\theta)$ and unit direction $\hat{\mathbf{d}}(\theta)$. Each integral is along the line with 1D measure $\mathrm{d}l(\mathbf{x})$ and the domain parameters $\mathbf{q} = (\theta, \tau) \in \mathbb{R}^2$ are the line orientation and offset. In contrast to the Fourier transform, the integrand remains fixed while the integration domain (the line through the 2D signal) varies with $\mathbf{q}$.

### 4.2 Results

Fig. 4 reports quality as a function of the number of integrand evaluations per training iteration. The central finding is that our approach is *highly sample efficient*: it reaches high quality with very few evaluations, while integral supervision which spends the same evaluations forming noisy Monte Carlo targets, needs substantially more to reach comparable quality. On rendering, the most striking case, our method's LPIPS at just $128$ evaluations is already better than integral supervision achieves at any budget we test. The size of this advantage depends on the evaluation budget and the nature of the integrand. For simple, low-variance integrands a few Monte Carlo samples already yield an accurate target, so integral supervision catches up quickly and can even overtake our approach. Motion blur, a 1D temporal integral with little variance, crosses over around $512$ evaluations, and the 2D Radon transform converges to our quality by the largest budgets. For harder integrands the gap is wider and more persistent, as with the oscillatory Fourier transform and bilateral filtering. Our quality is non-monotone in the per-iteration budget, peaking around $1,024$–$2,048$ evaluations before declining. It follows from the fixed wall-clock protocol and two competing effects: a larger per-iteration budget leaves proportionally fewer optimization iterations (Tab. 4), and because our construction already recovers the integral from few samples, additional samples bring little benefit while the lost iterations hurt. Integral supervision, which relies on many samples to denoise its targets, keeps improving instead. Our method appears to be at its strongest in the sample-limited regime that motivates learning parametric integrals in the first place. Fig. 3 shows these differences qualitatively: our reconstructions recover sharper filtered detail and cleaner specular highlights, whereas integral supervision retains residual Monte Carlo noise, most visibly in the imaginary Fourier spectrum and the specular renderings. Additional qualitative results can be found in the appendix.

### 4.3 Analysis

Our decomposition enforces a vanishing integral by construction. To test whether this exact structure is needed, we remove the divergence operator and instead add a loss that drives the training sample mean of $\mathbf{g}_\theta$ to zero, encouraging its integral to vanish. As Fig. 5 shows, this regularization is consistently worse across budgets, indicating that our exact zero-integral construction drives the accuracy gains. We also compare our finite-difference divergence estimate against exact automatic differentiation (AD). The two perform comparably, with finite differences slightly ahead on most

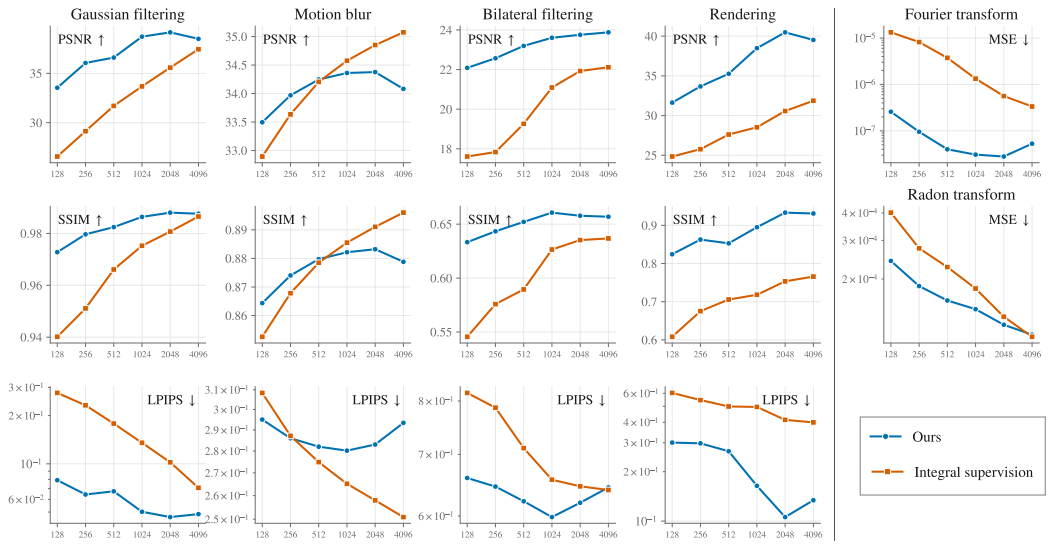



**Figure 4:** Quality as a function of integrand evaluations per training iteration, across all applications (columns).

applications and cheaper, as it avoids differentiating through the input-space derivatives required by the exact AD divergence.

Another way to view our construction is as a learned control variate. Recall from Eq. 8 that our model is the sum of a divergence term $\nabla_{\mathbf{x}} \cdot [w(\mathbf{x})\, \mathbf{g}_\theta(\mathbf{x}; \mathbf{p})]$, which integrates to zero over $\mathbf{x}$ for every $\mathbf{p}$, and the integral field $I_\phi$ that carries the integral. Due to this zero-integral property, subtracting the first term from the integrand $f(\mathbf{x}; \mathbf{p})$ leaves the integral $I(\mathbf{p})$ unchanged but cancels its variation over $\mathbf{x}$, precisely the role of a control variate in Monte Carlo estimation. When the vector field $\mathbf{g}_\theta(\mathbf{x}; \mathbf{p})$ is well fit, the difference $f(\mathbf{x}; \mathbf{p}) - \nabla_{\mathbf{x}} \cdot [w(\mathbf{x})\, \mathbf{g}_\theta(\mathbf{x}; \mathbf{p})]$ is nearly constant in $\mathbf{x}$, so its mean, the integral, is recovered from few samples. Integral supervision instead averages the raw, strongly varying integrand and needs many more samples for the same accuracy. This variance reduction over the integration variable is the source of our sample efficiency. Beyond accuracy, our learned representation enables operations that repeated numerical integration cannot. The integral is differentiable in its parameters, so we can solve inverse problems by gradient descent through the learned model. Additionally, since the parameters are just an augmented field space, we can filter in a spatially varying manner. We provide detail for both applications in the appendix (Sec. A.2 and Sec. A.3), along with further analysis (Sec. A.4, Sec. A.5, Sec. A.6). Furthermore, the learned integral transforms can be evaluated *continuously* at arbitrary parameter values, without the binning required by discrete transforms. We see exciting future work in learned representations for differential and integral equations.

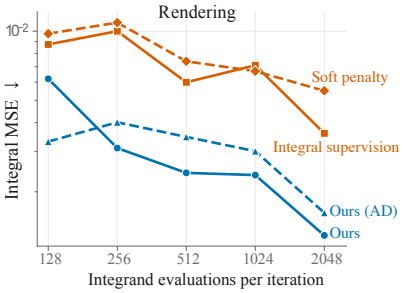



**Figure 5:** Ablation of our method against its auto-differentiation variant, soft zero-mean penalty, and integral supervision.

## 5 Conclusion

We introduced a neural representation for parametric integration that learns directly from pointwise integrand evaluations, without requiring numerically approximated integral targets. By decomposing the integrand into a zero-integral component and a component representing the integral, our approach amortizes integration across integrand and domain parameters while retaining general neural representations and requiring only first-order derivatives. Across filtering, integral transforms, and rendering, we demonstrated accurate parametric integration under limited sampling budgets, together with direct evaluation and differentiation of the learned integrals. We hope this provides a useful foundation for replacing repeated numerical integration with learned representations.

## 6 AI DISCLOSURE

We used generative AI tools to write and edit code that implements our methods and experiments, and to help polish the writing. We did not use them to generate datasets, develop the conceptual framing, formulate mathematical claims, propose or refine hypotheses, design our methodology or experiments, clean or reformat data, or interpret results. We have reviewed all AI-assisted work: all AI-generated code was verified and tested by the authors on the real experimental data. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

## REFERENCES

Eirikur Agustsson and Radu Timofte. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Álvaro Leitao and Jonatan Ráfales. Parametric numerical integration with (differential) machine learning. *arXiv preprint arXiv:2512.11530*, 2025.

Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8456–8465. IEEE, 2023.

Zilu Li, Guandao Yang, Qingqing Zhao, Xi Deng, Leonidas Guibas, Bharath Hariharan, and Gordon Wetzstein. Neural control variates with automatic integration. In *ACM SIGGRAPH 2024 Conference Papers*, pp. 1–9, 2024.

David B Lindell, Julien NP Martel, and Gordon Wetzstein. Autoint: Automatic integration for fast neural volume rendering. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14551–14560. IEEE, 2021.

Steffan Lloyd, Rishad A. Irani, and Mojtaba Ahmadi. Using neural networks for fast numerical integration and optimization. *IEEE Access*, 8:84519–84531, 2020.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021. doi: 10.1038/s42256-021-00302-5.

Daniel Maître and Roi Santos-Mateos. Multi-variable integration with a neural network. *Journal of High Energy Physics*, 2023(3):221, 2023.

D. Meister and T. Harada. Geometric integration for neural control variates. *Computer Graphics Forum*, 44(7):e70275, 2025.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Trans. Graph.*, 38(5):145:1–145:19, October 2019. ISSN 0730-0301. doi: 10.1145/3341156.

Thomas Müller, Fabrice Rousselle, Alexander Keller, and Jan Novák. Neural control variates. *ACM Transactions on Graphics (TOG)*, 39(6):1–19, 2020.

Ntumba Elie Nsampi, Adarsh Djeacoumar, Hans-Peter Seidel, Tobias Ritschel, and Thomas Leimkühler. Neural field convolutions by repeated differentiation. *ACM Transactions on Graphics (TOG)*, 42(6), Dec 2023.

Ntumba Elie Nsampi, Adarsh Djeacoumar, Hans-Peter Seidel, Tobias Ritschel, and Thomas Leimkühler. Volumetric inverse rendering via neural radiative transfer. *Computer Graphics Forum*, 45(4), 2026.

Katharina Ott, Michael Tiemann, Philipp Hennig, and François-Xavier Briol. Baysian numerical integration with neural networks. In *Uncertainty in Artificial Intelligence*, pp. 1606–1617. PMLR, 2023.

Poly Haven. Poly haven: The public 3D asset library. `https://polyhaven.com`.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1530–1538. PMLR, 2015.

Christian P Robert, George Casella, and George Casella. *Monte Carlo statistical methods*, volume 2. Springer, 1999.

Fizza Rubab, Ntumba Elie Nsampi, Martin Balint, Felix Mujkanovic, Hans-Peter Seidel, Tobias Ritschel, and Thomas Leimkühler. Learning neural antiderivatives. In *Vision, Modeling, and Visualization*, 2025.

Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pp. 1041–1044, 2014.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020.

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

Stanford Computer Graphics Laboratory. The stanford 3D scanning repository. `http://graphics.stanford.edu/data/3Dscanrep/`.

Charles Stein. A bound for the error in the normal approximation to the distribution of a sum of dependent random variables. In *Proceedings of the sixth Berkeley symposium on mathematical statistics and probability, volume 2: Probability theory*, volume 6, pp. 583–603. University of California Press, 1972.

Kartic Subr. Q-net: A network for low-dimensional integrals of neural proxies. In *Computer Graphics Forum*, volume 40, pp. 61–71. Wiley Online Library, 2021.

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *NeurIPS*, 2020.

Gregory H. Teichert, Anirudh R. Natarajan, Anton Van der Ven, and Krishna Garikipati. Machine learning materials physics: Integrable deep neural networks enable scale bridging by learning free energy functions. *Computer Methods in Applied Mechanics and Engineering*, 353:201–216, 2019.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Computer Graphics Forum*, 2022. ISSN 1467-8659. doi: 10.1111/cgf.14505.

Dejia Xu, Peihao Wang, Yifan Jiang, Zhiwen Fan, and Zhangyang Wang. Signal processing for implicit neural representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Mustafa B. Yaldiz, Ishit Mehta, Nithin Raghavan, Andreas Meuleman, Tzu-Mao Li, and Ravi Ramamoorthi. Spectral prefiltering of neural fields. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*, SA Conference Papers '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400721373.

Lina Yan, Jingjing Di, and Ke Wang. Spline basis neural network algorithm for numerical integration. *International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering*, 7(3):458–461, 2013.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 586–595, June 2018. doi: 10.1109/CVPR.2018.00068.

Zeng Zhe-Zhao, Wang Yao-Nan, and Wen Hui. Numerical integration based on a neural network algorithm. *Computing in Science & Engineering*, 8(4):42–48, 2006.

# A Additional Results and Details

## A.1 Per-application settings

**Datasets.** Our signals are obtained from standard public sources. Images for Gaussian filtering, bilateral filtering, and the Radon transform are natural images from DIV2K (Agustsson & Timofte, 2017). Audio for the Fourier transform are clips from UrbanSound8K (Salamon et al., 2014). Videos for motion blur are from UCF101 (Soomro et al., 2012). Meshes for rendering are the Stanford bunny and armadillo from the Stanford 3D Scanning Repository (Stanford Computer Graphics Laboratory). Environment maps for rendering are HDR panoramas from Poly Haven (Poly Haven). For each application we use 20 signals, held fixed across all methods and budgets.

**Hyperparameters and Hardware.** For each application, all methods were trained on the same GPU for an identical two-hour wall-clock budget, ensuring a fair comparison. Fourier transform, Gaussian filtering, motion blur, and rendering were trained using NVIDIA RTX 8000, while Radon transform and bilateral filtering were trained using NVIDIA A40. Hyperparameters were chosen on a small pilot set and are listed in Tab. 3.

**Parameter ranges and domains.** All ranges are given in the normalized coordinates our models use. Gaussian filtering integrates over the image plane $[-1, 1]^2$; the query is a kernel center in $[-0.75, 0.75]^2$ and an isotropic standard deviation $\sigma \in [0.01, 0.40]$. Motion blur integrates over the one-dimensional shutter interval of a space-time video volume; the query is a pixel in $[-1, 1]^2$, a shutter center $c \in [-0.35, 0.35]$ in normalized time $[-1, 1]$, and a half-exposure width $\Delta \in [0.03, 0.32]$. Bilateral filtering integrates over the image plane $[-1, 1]^2$; the query is a center in $[-0.75, 0.75]^2$, a spatial scale $\sigma_s \in [0.01, 0.40]$, and a range scale $\sigma_r \in [0.05, 0.50]$ over intensities in $[0, 1]$. Rendering integrates incident directions over the sphere; the query is a reflection direction on $\mathbb{S}^2$ and a Phong shininess $\alpha \in [1, 512]$, from near-matte to near-mirror, which we sample as $\alpha = 512^{\hat{\alpha}}$ with $\hat{\alpha} \in [0, 1]$ for a roughly log-uniform distribution. We parameterize the sphere by the unit square with an equal-area map ($\cos\theta = 1 - 2u_1$, $\phi = 2\pi u_2$), reducing rendering to the fixed-domain setting of Sec. 3.3. Fourier transform integrates the signal over $x \in [-1, 1]$ with frequency $\xi \in [0, 20]$, and is complex-valued (two output channels). Radon transform integrates along lines through the image, parameterized by orientation $\theta \in [0, \pi)$ and offset $\tau \in [-1, 1]$, over a line of half-length 1.5.

| Application | Ours $\epsilon$ | Integral supervision $b \times k$ | Soft penalty $\lambda$ |
|---|---|---|---|
| Gaussian filtering | $10^{-5}$ | $2 \times B/2$ | 3.0 |
| Motion blur | $10^{-4}$ | $B/4 \times 4$ | 5.0 |
| Bilateral filtering | $10^{-4}$ | $2 \times B/2$ | 0.1 |
| Rendering | $10^{-3}$ | $2 \times B/2$ | 3.0 |
| Fourier transform | $10^{-4}$ | $B/32 \times 32$ | 5.0 |
| Radon transform | $10^{-4}$ | $B/8 \times 8$ | 5.0 |

**Table 3:** Per-application hyperparameters. $B$ is the sample budget. $\epsilon$ is the finite-difference step for ours; $b \times k$ is the sample split of $b$ parameter values with $k$ Monte Carlo samples for integral supervision; $\lambda$ is the regularization weight for the soft penalty.

## A.2 Differentiating the learned integral

Our neural representation makes the learned integral differentiable in its parameters. We can therefore backpropagate through it and solve inverse problems by gradient descent, with no numerical integration in the loop. We demonstrate this on convolution as a target-matching task: we produce a target filtered output at a held-out kernel scale $\sigma^\star$ and fixed location, then recover the scale by minimizing the squared error $\|I_\phi(c_x, c_y, \sigma) - \text{target}\|^2$ over $\sigma$ with gradient descent through the model, initialized away from $\sigma^\star$. Each optimization step is a forward and backward pass through the learned integral, rather than a fresh numerical integration, so the recovery is cheap and fully differentiable. Fig. 6 shows the estimated scale converging to the target in 100 steps.

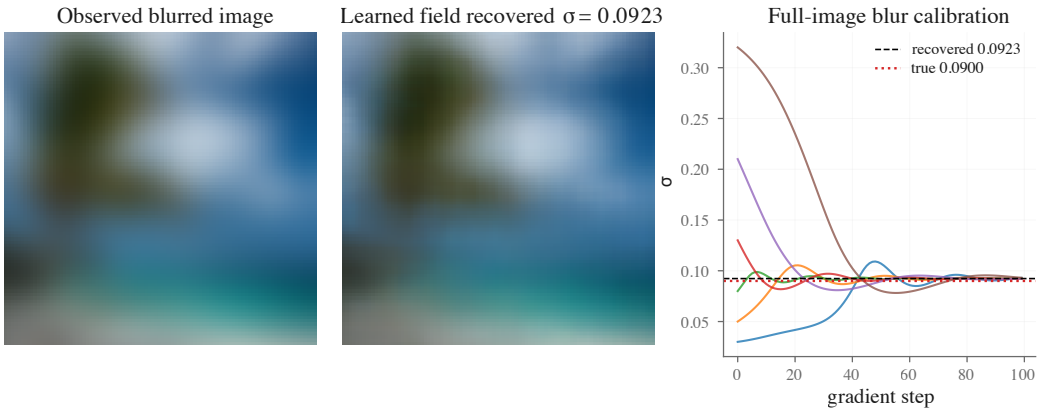



**Figure 6:** Blur recovery: the kernel scale that produces a target filtered output is recovered by gradient descent through the learned convolution integral.

### A.3 Spatially varying filtering

Our formulation supports spatially varying filtering without any additional training cost. Since both the kernel scale and its location are input parameters of the learned integral rather than fixed properties of the model, a single trained convolution model can apply a *different* filter at every output location without retraining. We define a spatial scale map $\sigma(c_x, c_y)$ and evaluate the model at $(c_x, c_y, \sigma(c_x, c_y))$ for each pixel in a single forward pass. This assigns a per-pixel filter directly from the trained representation, with no per-pixel optimization and no numerical integration. Fig. 7 shows two such maps computed from a single model: in one the kernel scale increases radially from the center outward, and in the other it increases from right to left. Both stay within the range of scales seen during training. The same mechanism applies to any parameter of a learned integral, so other filters can be made to vary spatially in the same way.

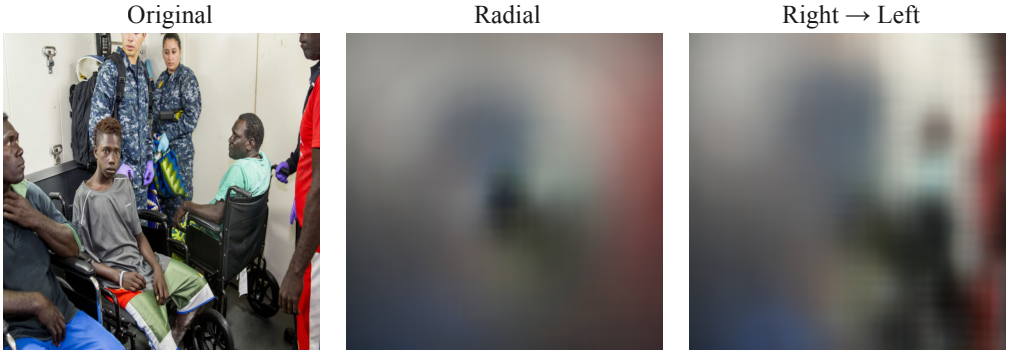



**Figure 7:** Spatially varying Gaussian filtering from a single learned model: the kernel scale increases radially from the center outward, and from right to left, respectively.

### A.4 Varying surface shininess

Shininess $\alpha$ is one of the parameters of the learned rendering integral, so a single trained model represents the pre-filtered environment across the full range of specular roughness. Fig. 8 shows two $\alpha$ settings, a broad and a sharp reflection, at a fixed viewpoint on two meshes. As $\alpha$ increases the specular lobe narrows and the reflected environment sharpens, and our reconstruction tracks the dense-sampling reference more closely than integral supervision, whose estimates remain noticeably noisier and show interference patterns at the sharper, higher-variance settings.

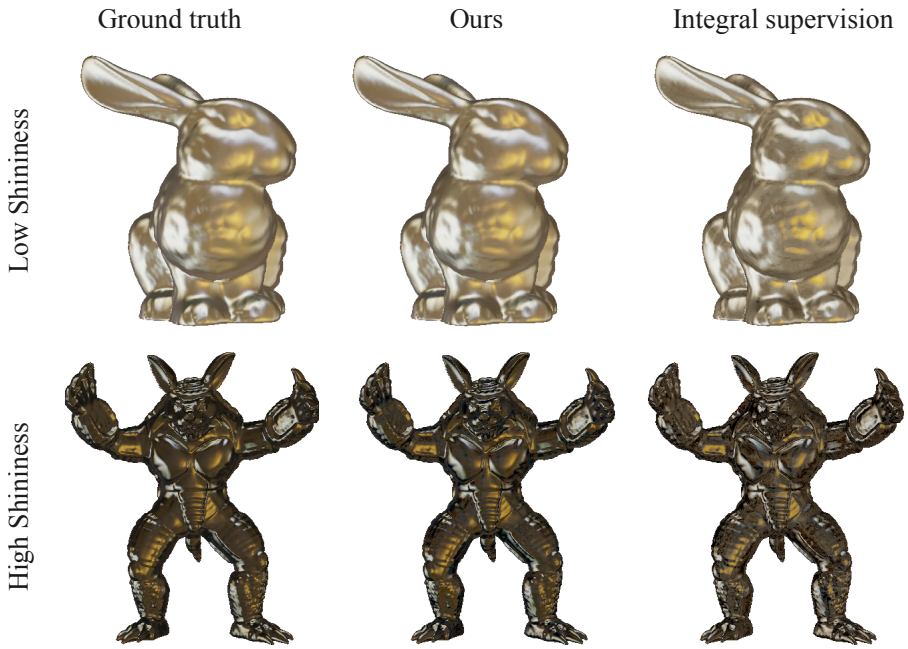



**Figure 8:** Rendering as the surface shininess $\alpha$ varies, from broad to sharp specular reflection.

### A.5 Quality across the parameter range

Different parameter settings pose different difficulty: some kernel scales or shininess values yield smoother, lower-variance integrands than others, so approximation error naturally varies across the parameter range. Fig. 9 breaks PSNR down by parameter for two applications, the kernel scale $\sigma$ for Gaussian filtering and the surface shininess $\alpha$ for rendering under a budget of 4096 samples.

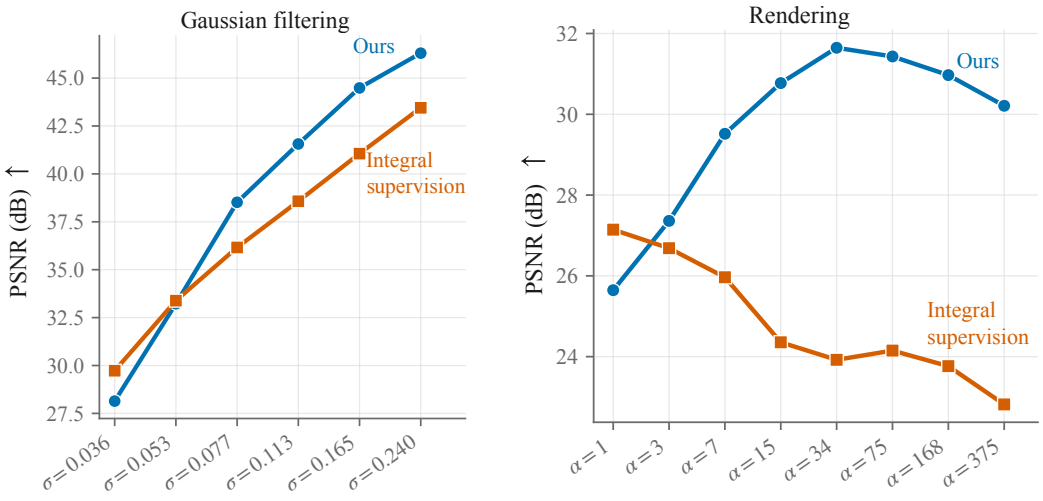



**Figure 9:** PSNR broken down by parameter setting: kernel scale $\sigma$ for Gaussian filtering (left) and surface shininess $\alpha$ for rendering (right).

### A.6 Training cost

Tab. 4 breaks down the training cost of each method. All methods share the same two-hour wall-clock budget, so a larger per-iteration budget yields proportionally fewer iterations. Our method performs the fewest iterations and, correspondingly, consumes the fewest integrand evaluations overall.

**Table 4:** Training cost per application and per-iteration sample budget, averaged over signals. Iters is the number of optimization steps completed, ms/it the mean time per step, and Evals the total integrand evaluations consumed (budget $\times$ iterations, in millions).

| | | Ours | | | Integral supervision | | |
|---|---|---|---|---|---|---|---|
| Application | Budget | Iters | ms/it | Evals | Iters | ms/it | Evals |
| Gaussian filtering | 128 | 6k | 1206.2 | 0.8 | 14k | 521.8 | 1.8 |
| | 256 | 5k | 1480.2 | 1.2 | 14k | 521.4 | 3.5 |
| | 512 | 3k | 2237.0 | 1.7 | 14k | 533.9 | 6.9 |
| | 1024 | 2k | 3494.5 | 2.1 | 13k | 562.9 | 13.1 |
| | 2048 | 1k | 6560.8 | 2.3 | 12k | 601.2 | 24.6 |
| | 4096 | 529 | 13770.5 | 2.2 | 11k | 679.5 | 43.5 |
| Motion blur | 128 | 7k | 1071.4 | 0.9 | 13k | 550.9 | 1.7 |
| | 256 | 6k | 1267.7 | 1.5 | 13k | 561.8 | 3.3 |
| | 512 | 4k | 1712.4 | 2.2 | 12k | 578.5 | 6.4 |
| | 1024 | 3k | 2561.6 | 2.9 | 11k | 648.8 | 11.4 |
| | 2048 | 2k | 4235.7 | 3.5 | 10k | 748.9 | 19.7 |
| | 4096 | 868 | 8331.4 | 3.6 | 8k | 932.0 | 31.7 |
| Bilateral filtering | 128 | 8k | 953.1 | 1.0 | 15k | 544.6 | 1.9 |
| | 256 | 6k | 1218.1 | 1.5 | 15k | 490.4 | 3.8 |
| | 512 | 4k | 1642.8 | 2.2 | 13k | 643.5 | 6.5 |
| | 1024 | 3k | 2471.0 | 3.0 | 13k | 594.9 | 12.9 |
| | 2048 | 2k | 4121.5 | 3.6 | 10k | 812.5 | 20.0 |
| | 4096 | 894 | 9113.7 | 3.7 | 8k | 993.6 | 31.5 |
| Rendering | 128 | 4k | 2462.4 | 0.5 | 8k | 1052.8 | 1.0 |
| | 256 | 3k | 2771.5 | 0.9 | 11k | 690.3 | 2.8 |
| | 512 | 2k | 4265.0 | 1.1 | 12k | 591.9 | 6.2 |
| | 1024 | 2k | 3269.8 | 2.6 | 11k | 660.7 | 11.2 |
| | 2048 | 2k | 4416.9 | 3.4 | 8k | 942.2 | 15.7 |
| | 4096 | 524 | 13868.1 | 2.1 | 7k | 1035.1 | 28.6 |
| Fourier transform | 128 | 12k | 596.0 | 1.6 | 69k | 105.1 | 8.8 |
| | 256 | 9k | 790.7 | 2.3 | 68k | 105.8 | 17.4 |
| | 512 | 6k | 1233.1 | 3.0 | 64k | 112.9 | 32.7 |
| | 1024 | 4k | 1995.3 | 3.7 | 60k | 120.1 | 61.5 |
| | 2048 | 2k | 3599.2 | 4.1 | 56k | 129.8 | 113.9 |
| | 4096 | 984 | 7342.6 | 4.0 | 47k | 154.2 | 192.6 |
| Radon transform | 128 | 10k | 892.4 | 1.3 | 25k | 337.5 | 3.2 |
| | 256 | 9k | 1061.0 | 2.3 | 44k | 193.9 | 11.3 |
| | 512 | 6k | 1609.8 | 2.9 | 32k | 295.1 | 16.4 |
| | 1024 | 3k | 2714.8 | 3.5 | 27k | 349.9 | 27.7 |
| | 2048 | 3k | 3268.1 | 5.7 | 33k | 250.3 | 67.4 |
| | 4096 | 2k | 3967.4 | 7.5 | 23k | 377.4 | 93.3 |