DIFFERENTIABLE KERNEL METHODS WITH ERROR CERTIFICATES FOR DEEP LEARNING PIPELINES

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern deep learning pipelines deliver strong accuracy but little per-prediction error control, whereas classical kernel methods come with a mature approximation theory that does not, by default, scale to or plug into such pipelines. We close that gap with two constructs. *Sparse kernels* make kernel ridge regression scalable: a localized lazy variant that defers computation to inference and solves only small local systems. *Differentiable kernel layers* make it modular: the same regressor exposed as an end-to-end trainable module, in a doubly-projected form that admits distinct embeddings for the query and the centers while keeping a symmetric positive-definite kernel, and whose three parameter sets—representations, targets, evaluation points—may each be fixed or learned. The two compose, and the design space that follows is broader than end-to-end training alone: training-free transfer, nonlinear probing, hybrid and purely kernel models, classification and policy heads.

Our thesis is that these constructs let a deep model carry a *computable, geometry-aware error indicator*, evaluated at each query and propagated through the pipeline—a kernel-specialized estimate that conformal prediction can build on, and a concrete step towards auditable predictions. Across vision, reinforcement learning, out-of-distribution detection and tabular data, such components integrate at competitive cost. And because the readout is a metric object, it suggests training objectives one would not otherwise consider: we report a Gromov–Monge criterion on the representation geometry that involves no likelihood and no logits.

1 INTRODUCTION

Deep neural networks dominate modern machine learning, yet they offer little per-prediction error control: a trained network returns an output, not a computable statement of how far that output may be from the truth. Kernel methods sit at the opposite end—a mature approximation theory equips them with pointwise error bounds—but classical kernels neither scale to modern data nor integrate as trainable components of deep pipelines. This paper sets out to bring the error theory of kernels *into* deep learning: to make kernel predictors scalable, differentiable, and modular enough to act as drop-in components, and to carry their error estimates over to the resulting deep models.

We organize the approach around two constructs. **Sparse kernels** make kernel ridge regression (KRR) scalable: a lazy variant of local KRR (Han et al., 2022; Schaback, 2025)[1] in which training is deferred to inference and reduces to small local systems, preserving consistency and continuity while lowering the cubic cost of dense KRR. **Differentiable kernel layers** make it modular: the same regressor exposed as an end-to-end-trainable module, building on deep kernel learning (Wilson et al., 2016; Wilson & Nickisch, 2015; Wenzel et al., 2026), in the doubly-projected form of Section 2.3—which admits distinct embeddings for the query and the centers while keeping a symmetric positive-definite kernel, and serves as a classification head, as a reinforcement-learning policy head, or alongside a transformer's position-wise feed-forward sub-block.

A central perspective is that KRR decomposes learning into three interacting components—feature representations, targets, and evaluation points—each of which may be fixed or learned. This decomposition broadens the design space beyond standard end-to-end training: it yields training-free inference

[1]We use "lazy" in the sense of deferred computation, not the lazy/NTK training regime of the infinite-width literature.

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.

for transfer learning, nonlinear probes of intermediate representations, and hybrid kernel–neural models. It also reframes the readout layer of deep models. Linear probes (Alain & Bengio, 2016) assess representations but are confined to linear function classes; kernel probes instead provide nonlinear, data-adaptive readouts that better capture the geometry of learned features.

What ties these constructs together is error theory. Each kernel component is an element of an RKHS, so classical pointwise estimates—the power-function (Schaback) bounds—apply. We adapt them to the sparse, localized setting, extend them to the deep readout through a certifiable surrogate, and *compose* them along a deep pipeline, so that the resulting error indicator survives intervening layers, including (kernel-smoother) attention. Being per-query, geometry-aware and computed directly from the kernel, it can serve as the score behind the marginal coverage of conformal prediction, as discussed below.

This matters beyond methodology: per-prediction error control is increasingly a condition of use in regulated or safety-critical domains. We claim no formal certification, only that endowing deep kernel components with computable estimates is a concrete step along that road—one that does not require abandoning deep architectures or their accuracy.

Related work

Deep kernel learning and the kernel view of deep models. Combining neural feature extractors with kernel predictors has a long history: deep kernel learning (Wilson et al., 2016; Wilson & Nickisch, 2015; Wenzel et al., 2026) learns data-dependent representations and applies a kernel method (often a Gaussian process) on top, while neural-tangent-kernel and infinite-width analyses (Ghorbani et al., 2020; Lee et al., 2020) characterize when finite networks behave like kernel machines. Attention itself admits a kernel reading (Tsai et al., 2019; Ramsauer et al., 2021), and end-to-end kernel learning has been explored in convolutional architectures (Mairal, 2016). Our doubly-projected readout (Section 2.3) keeps a *symmetric* positive-definite kernel—distinguishing it both from the asymmetric-kernel literature (Suykens, 2016), which forgoes a single RKHS, and from views that recast the feed-forward sub-block as attention over a learnable memory (Sukhbaatar et al., 2019; Lample et al., 2019).

Localized and structured kernel methods, and kernel libraries. Scalability has been pursued through inducing-point approximations such as KISS-GP (Wilson & Nickisch, 2015), structured deep kernels (Wenzel et al., 2026), and local kernel ridge regression (Han et al., 2022; Schaback, 2025). Our sparse kernels build on the latter, using M-NN to define a tessellation and deferring kernel solves to inference; compared with Han et al. (2022) we adopt a discontinuous extrapolation for performance. Among kernel libraries, GPyTorch (Gardner et al., 2018) offers scalable Gaussian-process inference, Falkon (Meanti et al., 2020) efficient Nyström KRR, and KeOps (Charlier et al., 2021) symbolic GPU kernel operations; none, however, exposes a kernel regressor's features, targets and evaluation points in a fully lazy, differentiable module usable as a layer.

Probing. Linear probes (Alain & Bengio, 2016) are a standard tool for analyzing learned representations; recent work extends them to more expressive predictors (Adam et al., 2025) or to transfer with ridge regression (Tang & de Sa, 2020). Our kernel probes provide a nonlinear, data-adaptive readout that, in the lazy regime, requires no additional training and exposes geometric notions of representation quality from the RKHS structure, such as the local error indicator ϵ_k^σ of Section 2.2.2.

Error estimates, uncertainty, and conformal prediction. Kernel methods carry a classical a posteriori error theory: the power function and Schaback bounds (Wendland, 2005; Schaback, 2025) certify RKHS interpolants pointwise, and computable a posteriori certificates are central to certified reduced-order modeling (Haasdonk, 2017). Model-based uncertainty is likewise available through the Gaussian-process posterior. In a complementary, model-agnostic direction, conformal prediction (Vovk et al., 2005; Angelopoulos & Bates, 2023) yields distribution-free *marginal coverage* around arbitrary predictors. Our indicator sits between these: model-based and per-query like the RKHS bounds, computed directly from the kernel, but—unlike posterior variance or coverage—transferred to the *sparse*, *deep*, and *composed* settings of a deep pipeline (Appendix A.11)—and able to shelter conformal machinery efficiently, a split-conformal wrapper turning it into a detector with a controlled false-positive rate at no additional cost (Appendix A.10.2).

Contributions We make four contributions.

- **Sparse kernels.** We give KRR a localized, lazy instantiation that defers computation to inference and scales to deep-learning batch sizes, its error estimate localizing cell by cell (Sections 2, 2.2).
- **Differentiable kernel layers.** We expose that regressor as a differentiable, modular layer with three controllable parameter sets (representations, targets, evaluation points), in a doubly-projected form that admits distinct embeddings while keeping a symmetric PSD kernel—unlike asymmetric-kernel methods or the kernel view of attention (Section 2.3). We deploy it as a classification head, as an RL policy head, and alongside the transformer's feed-forward path.
- **Error theory in deep pipelines.** We transfer classical RKHS estimates to the sparse and deep readouts (via a certifiable surrogate) and compose them through full pipelines, yielding a computable, per-query error indicator—an indicator that conformal schemes can build on (Appendix A.11).
- **Empirical study.** Across transfer, probing, a Gromov–Monge ViT, Double DQN and PPO, out-of-distribution detection and a tabular benchmark, kernel components integrate as readouts, as added modules and as full substitutes at competitive cost, each carrying the error indicator above.

2 Deep kernel methods

2.1 Dense kernel ridge regression

We follow the notational conventions of Appendix A.1, and recall the basics of kernel ridge regression (Wendland, 2005; Schölkopf & Smola, 2002) that we need to describe deep kernel approaches and sparse kernels. Given a finite set of N distinct features $x = (x^1, \dots, x^N) \in \mathbb{R}^{N \times D_x}$ with target values $y = (y^1, \dots, y^N) \in \mathbb{R}^{N \times D_y}$, we consider a strictly positive definite kernel, that is, a symmetric function $k : \mathbb{R}^{D_x} \times \mathbb{R}^{D_x} \to \mathbb{R}$ such that the Gram matrix $k(x, x) \in \mathbb{R}^{N \times N}$ is positive definite. This kernel defines a unique Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$, and also a finite-dimensional space of vector-valued functions,

$$\mathcal{H}_k^x = \mathrm{Span}\big\{k(\cdot, x^n),\ n = 1, \dots, N\big\}, \tag{1}$$

whose elements we denote by $y_{k,x,\theta}(\cdot)$, or $y_{k,\theta}(\cdot)$, which is equal to $k(\cdot, x)\,\theta$, with coefficients $\theta \in \mathbb{R}^{N \times D_y}$, and $\|y_{k,\theta}\|^2_{\mathcal{H}_k^x} = \theta^T k(x, x)\theta$. A word on naming: we follow the layer convention of deep learning (cf. (8)), in which functions are denoted by the outputs they produce—so $y_{k,\theta}$ is the kernel counterpart of the customary notation y_θ of neural networks, desambiguated with the kernel letter k.

We work throughout in the *interpolation regime*: the kernel being strictly positive definite, $\dim(\mathcal{H}_k^x) = N$ and $k(x, x)$ is invertible without any further regularization (e.g. Tikhonov). Strictly speaking, the resulting operator is thus *kernel interpolation*—the ridgeless limit of kernel ridge regression; we retain the name KRR for continuity with the literature, a small nugget being used in practice for numerical stability only.

The same space admits a second basis. The interpolation regime induces the N *cardinal* (or *Lagrange*) basis functions of $\mathcal{H}_k^x$,

$$\psi_k(\cdot, x) \equiv \psi_k(\cdot) = (\psi_k^1(\cdot), \dots, \psi_k^N(\cdot)) = k(\cdot, x)\, k(x, x)^{-1} \in \mathbb{R}^N, \tag{2}$$

which satisfy $\psi_k^n(x^m) = \delta_{n,m}$, so that

$$\mathcal{H}_k^x = \mathrm{Span}\big\{\psi_k^n(\cdot),\ n = 1, \dots, N\big\}. \tag{3}$$

Both bases describe the same parametrized space, before any loss is introduced: $y_{k,\theta_y}(\cdot) = \psi_k(\cdot)\,\theta_y = k(\cdot, x)\,\theta$ with $\theta_y = k(x, x)\,\theta$, and $\|y_{k,\theta_y}\|^2_{\mathcal{H}_k^x} = \theta_y^T k(x, x)^{-1}\theta_y$. Choosing between them is a modeling decision, and a problem-dependent one. In (3) the coefficients are, by construction, the function's *values at the centers*, $y_{k,\theta_y}(x) = \theta_y$—so the pair (x^n, θ_y^n) keeps its meaning under any restriction of the index set, which is what allows Section 2.2 to share values across neighborhoods, and $\theta_y = 0$ yields the zero function exactly, a property we exploit when initializing kernel modules inside larger architectures (Section 3.4). In (1) the coefficients are neighborhood-relative and cannot be shared.

A computable error indicator. Kernel methods come with local, global and confidence-interval-type error bounds (Singh & Vijaykumar, 2023). The one this paper carries through its pipelines is the pointwise estimate given by the *power function*. Let $f \in \mathcal{H}_k$ be continuous and let $y_{k,f(x)}(\cdot) = \psi_k(\cdot)\, f(x)$ be its interpolant on the centers x. For a normalized kernel, $k(z,z) = 1$, the Schaback bound reads

$$\big|y_{k,f(x)}(z) - f(z)\big| \le \epsilon_k\big(z,x\big)\, \|f\|_{\mathcal{H}_k}, \qquad \epsilon_k\big(z,x\big) = \sqrt{1 - \psi_k(z,x)\, k(x,z)}, \tag{4}$$

and follows from Caponnetto & De Vito (2007) together with the reproducing property (Schölkopf & Smola, 2002). Two features matter for what follows. The indicator $\epsilon_k(z,x)$ is *computable*: it is built from the same Lagrange functions (2) as the predictor, and depends on the query and the centers alone—neither on f nor on the targets. And it is *geometric*: it vanishes at the centers and grows as z leaves their neighborhood, so it reports the coverage of a query by the data rather than the difficulty of the target. The bound is a worst-case guarantee, and an attained one: $\epsilon_k(z,x)$ is the largest error over the unit ball of $\mathcal{H}_k$. What $\epsilon_k(z,x)$ is blind about is the scale $\|f\|_{\mathcal{H}_k}$, and whether the target f lies in the space $\mathcal{H}_k$. The representer theorem, which licenses searching minimizers in the form (1), and the closed form it yields for the squared loss, are recalled in Appendix A.3.

2.2 Sparse kernel methods

We now localize the construction of Section 2.1. Throughout we rely on an M-nearest-neighbor approach, since efficient GPU-distributed implementations are publicly available, and build the sparse kernel construction directly on top of this M-NN structure, without subsampling the dataset.

2.2.1 M-NN tessellation

Let $d(\cdot,\cdot)$ be a distance, and $x \in \mathbb{R}^{N,D_x}$ a set of points. Let $1 \le M \le N$ be a *bandwidth* parameter, and consider the local ordering $\sigma(\cdot) = (\sigma^1, \cdots, \sigma^M)(\cdot) \in \Sigma$ given by the combination satisfying $d(z, x^{\sigma^n(z)}) \le d(z, x^{\sigma^m(z)})$ for $n \le m \le M$ at any given point $z \in \mathbb{R}^{D_x}$. The map $z \mapsto \sigma(z) = (\sigma^1, \cdots, \sigma^M)(z)$, the first M-truncation of the nearest-neighbor ordering, is piecewise constant and tessellates $\mathbb{R}^{D_x}$, denoted $\mathbf{1}_{\Omega^\sigma}$, into regions sharing the same M-neighborhood:

$$1 = \sum_{\sigma \in \Sigma} \mathbf{1}_{\Omega^\sigma}(x), \quad \Omega^\sigma := \{x \in \mathbb{R}^{D_x} : \sigma(x) = \sigma\},$$

to which we associate $x^{\sigma(z)} = (x^{\sigma^1(z)}, \cdots, x^{\sigma^M(z)})$, the M-truncated vector of features; we write $\sigma = \sigma(\cdot) = (\sigma^n(\cdot))_{n=1,\cdots,M}$ for concision.

2.2.2 Sparse, Local, Reproducing Kernel Hilbert Spaces

With $x = (x^1, \ldots, x^N)$ as input features, we fix a positive definite kernel of the form $k(\cdot,\cdot) = \varphi(d(\cdot,\cdot))$ for some distance d and a positive definite function (activation) φ (Wendland, 2005), and let $\sigma(\cdot)$ denote the M-NN truncation introduced above. Localizing (3) is immediate: on each cell Ω^σ, the M centers x^σ induce their own Lagrange basis, and we define

$$\mathcal{H}_{k,\sigma} = \mathrm{Span}\big\{\psi_k^n(\cdot, x^\sigma),\ n = 1, \ldots, M\big\}, \qquad \psi_k(\cdot, x^\sigma) = k(\cdot, x^\sigma)\, k(x^\sigma, x^\sigma)^{-1}, \tag{5}$$

with $k(x^\sigma, x^\sigma)$ a **dense** matrix of size $M \times M$. The elements of the local space read, in the basis (5) of the cell, $y_{k,\theta_y^\sigma}(\cdot) = \psi_k(\cdot, x^\sigma)\, \theta_y^\sigma$, with $\theta_y^\sigma \in \mathbb{R}^{M \times D_y}$ free parameters per cell. A direct sum would involve $\#\Sigma \times M \times D_y$ coefficients, enormous and discontinuous. But the coefficients of (3) are function *values at centers*, so a value θ_y^n at x^n is meaningful in *every* cell whose neighborhood contains x^n: sharing values across cells glues the local spaces through a single global $\theta_y \in \mathbb{R}^{N \times D_y}$.

Sparse Lagrange functions are the functions $\Psi_k(\cdot, x^\sigma) = (\Psi_k^1, \ldots, \Psi_k^N)$, given in closed form by (11) of Appendix A.2, which also establishes their three properties: at most M of the N functions are nonzero at any z, whence the $\mathcal{O}(M^3)$ per-query cost; the Lagrange property survives localization, $\Psi_k^n(x^j, x^\sigma) = \delta_{n,j}$; and on each cell the global object *is* the local predictor, its M coefficients read off the single table θ_y. Hence the global sparse space, in exact parallel to (1)–(3):

$$\mathcal{H}_k^M = \mathrm{Span}\big\{\Psi_k^n(\cdot, x^\sigma),\ n = 1, \ldots, N\big\}, \qquad y_{k,\theta_y}(\cdot) = \Psi_k(\cdot, x^\sigma)\, \theta_y, \quad \theta_y \in \mathbb{R}^{N \times D_y}, \tag{6}$$

For positive-definite k this is a kernel ridge regressor carrying $N \times D_y$ coefficients and requiring no $N \times N$ factorization: it is piecewise continuous (globally discontinuous in our experiments; continuous variants in Appendix A.14, close in spirit to Han et al. (2022)), it interpolates the data in the lazy configuration $\theta_y = y$, and it is *lazy*—each prediction is independent and costs $\mathcal{O}(M^3)$.

The indicator localizes. The estimate (4) transfers to (6) cell by cell, and this is the mechanism by which RKHS error bounds carry over to the sparse setting in general. On the cell Ω^σ the predictor is, by property (iii) of (11), the interpolant on the M centers x^σ; the bound (4) therefore applies there with x^σ in place of x, giving for every query

$$\big|y_{k,\theta_y}(z) - f(z)\big| \le \epsilon_k^\sigma\big(z\big)\,\|f\|_{\mathcal{H}_k}, \qquad \epsilon_k^\sigma(z) = \epsilon_k\big(z, x^{\sigma(z)}\big) = \sqrt{1 - \psi_k(z, x^\sigma)\,k(x^\sigma, z)}, \quad (7)$$

in the lazy configuration $\theta_y = f(x)$. Computing it costs one more local system of the size already solved for the prediction, on the same M neighbours—which is what makes a per-query indicator affordable inside a pipeline. The construction (6) is the more general object of this paper, and the dense regressor of Section 2.1 its global member at $M = N$.

Remark 1 (Algorithmic complexity: a dial between regimes)**.** *A sparse prediction costs $\mathcal{O}(M^3) + \mathcal{O}(M^2 D_y)$ per query, independently of N, with no fit: large training sets favor small M (lazy, per-query local solves). The dense member costs one $\mathcal{O}(N^3)$ factorization amortized over all subsequent queries: moderate training sets facing large query streams favor $M = N$. Intermediate cases admit the hybrid constructions of Appendix A.14.*

2.3 Kernel layers for deep learning pipelines

With the dense regressor and its sparse family in hand, we turn to their integration as differentiable layers of standard pipelines.

Deep learning. To learn an association $x \mapsto y$, modern models are typically composed of L layers, starting from $y^0 = x \in \mathbb{R}^{N \times D_x}$ and producing the target $y^L = y \in \mathbb{R}^{N \times D_y}$. To each layer $l = 1, \dots, L$ we attach a parameter set θ^l and a function $y^l(\cdot, \theta^l) \in \mathbb{R}^{D^l}$. Letting $\theta = (\theta^1, \dots, \theta^L)$ denote the full parameter set, each layer is computed as

$$y_\theta^l(\cdot) = y^l(y_\theta^{l-1}(\cdot), \theta^l) \in \mathbb{R}^{D^l}, \qquad l = 1, \dots, L, \quad (8)$$

for some D^l. Let x, y be a training set. A loss functional $\mathcal{L}(y_\theta^L(x), y)$ is associated with this construction, and parameters are obtained by approximately solving $\inf_\theta \mathcal{L}(y_\theta^L(x), y)$, typically via stochastic gradient or quasi-Newton optimizers, whose derivatives $\nabla_\theta \mathcal{L}(y_\theta^L(x), y)$ are available through automatic differentiation. For example, in a convolutional network such as VGG (Simonyan & Zisserman, 2014), $\theta^1, \cdots, \theta^{L-1}$ describe the convolutional filter parameters, $y_\theta^{L-1}(\cdot)$ is the composition of all layers up to the classifier, that is, the convolutional trunk, and the classifier head is the last layer, $y_\theta^L(\cdot, \theta^L)$, having its own parameters.

The doubly-projected kernel layer. The *doubly-projected kernel layer* (DKL) adds to (8) a chosen SPD kernel k and two parameters θ_z, θ_x defining *adapter maps* into its argument space of dimension E. The two adapters do not have the same type, and that is what tells the two arguments apart: $P_{\theta_z} : \mathbb{R}^{D^{l-1}} \to \mathbb{R}^E$ maps a query into that space, while $P_{\theta_x} : \mathbb{R}^{D^{l-1}} \to \mathbb{R}^{C \times E}$ returns the C *centers* relevant to it—a constant map $P_{\theta_x}(\cdot) = x$ in the dense case, and the query-dependent selection $x^{\sigma(\cdot)}$ in the sparse one, so that center selection is part of the adapter. With $\theta_y \in \mathbb{R}^{C \times D_y}$ a set of free coefficients, the *dense layer* DKR reads

$$y^l_{k,\theta,\theta_z,\theta_x,\theta_y}(\cdot) \;=\; k\big(P_{\theta_z}(\cdot),\; P_{\theta_x}(\cdot)\big)\,\theta_y. \quad (9)$$

The subscript k indicates the kernel construction. Its Lagrange (sparse) version (5) is

$$y^l_{k,\theta,\theta_z,\theta_x,\theta_y}(\cdot) \;=\; \Psi_k(P^\sigma_{\theta_z}(\cdot), P^\sigma_{\theta_x}(\cdot))\,\theta^\sigma_y. \quad (10)$$

with $\sigma \equiv \sigma(P_{\theta_z}(\cdot))$ the M-NN truncation taken in the embedded space.

The form (9)–(10) is a general template, allowing for various combinations of parameters—each of which may be fixed or learned. Appendix A.4 lists the instances used in this paper, and states

them in full. To fix ideas, a kernel readout layer, adding no parameters, can be written as $y^l_{k,\theta}(\cdot) = \Psi_k(y^{l-1}_\theta(\cdot), y^{l-1}_\theta(x))\, y$, where $x \mapsto y$ is the training features and targets, see (12). This readout can be used either for transfer (no parameter learning), or with gradients flowing to retrieve θ in a learning phase.

The kernel remains symmetric and positive-definite. Taking $P_{\theta_z} = P_{\theta_x}$ recovers a standard symmetric construction, but we also accept $P_{\theta_z} \neq P_{\theta_x}$, and two kernels must then be distinguished. The function induced on the input space, $(z, x) \mapsto k(P_{\theta_z}(z), P_{\theta_x}(x))$, is not symmetric; the kernel k we use, on the embedding space $\mathbb{R}^E$, does not depend on the projections and stays symmetric PSD. Both layers are therefore linear combinations of sections of k and, as functions of the projected query, elements of $\mathcal{H}_k$—which grants them the error estimates of Appendix A.11, unlike asymmetric-kernel methods (Suykens, 2016) or the kernel view of attention (Tsai et al., 2019; Ramsauer et al., 2021), whose coefficients are values tied to x.

2.3.1 Losses and training

Let $\ell : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ be a per-sample loss (cross-entropy, mean squared error), aggregated as $\mathcal{L}(a, b) = \sum_i \ell(a^i, b^i)$. Given evaluation features $z \in \mathbb{R}^{N_z \times D_z}$ with target values $y_z \in \mathbb{R}^{N_z \times D_y}$, and writing $y_k = y_{k,\theta,\theta_z,\theta_x,\theta_y}$ for short, training the layer (9) reads $\inf_{\theta,\theta_z,\theta_x,\theta_y} \mathcal{L}\big(y_k(z), y_z\big)$. At frozen $\theta, \theta_z, \theta_x$ the remaining problem in θ_y is, by the representer theorem, the functional minimization $\inf_{y_k \in \mathcal{H}_k^{P_{\theta_x}}} \mathcal{L}(y_k(z), y_z)$ over the space (1) built on the projected centers. In the feature parameters it is non-convex, as for any model of the form (8), and we rely on the same stochastic first-order machinery as the rest of the pipeline. Learning with the Lagrange basis deserves separate mention, because Section 3.3 rests on it and it leads to a degenerate case; see Remark 3.

Implementation. Optimizers such as Adam, SGD or LBFGS require, in our setting, the three gradients $\nabla_{\theta_y} y_k(z)$, $\nabla_{\theta_x} y_k(z)$ and $\nabla_{\theta_z} y_k(z)$, with respect to the coefficients, the centers and the query adapter. These are computed by the kernel library of LeFloch et al. (2025) and exported to the deep learning backend.

3 Numerical experiments

Experimental details and complementary experiments are reported in the Appendix: the common setting in Appendix A.5, and one dedicated section per experiment, for instance Appendix A.6 for transfer learning.

3.1 Transfer learning

On a frozen ImageNet-pretrained ResNet-18 (He et al., 2016; Deng et al., 2009) we compare five readouts on CIFAR-10 over subsample sizes N_x from 500 to 50 000, all scored on the same $N_z = 10\,000$ test images: a linear head, a two-layer MLP, a k-NN vote, the sparse lazy readout (12) at bandwidth M, and Nyström KRR (Figure 1; Appendix A.6 for the full grid and the GPU pass).

The sparse readout reaches the accuracy of the trained heads without fitting a coefficient: 0.894 at $M = 500$ against 0.889 for the MLP and 0.870 for the linear probe. *At matched neighbourhood it improves on the k-NN vote everywhere*, by 1.7 to 15.3 points—both see the same neighbours and differ only in how they combine them, so the margin measures what solving the local system buys over averaging over it.

And accuracy and cost order the constructions differently, which is a division of labour rather than a tie. The dense member is the most accurate and the most expensive, its $\mathcal{O}(N_x^3)$ factorization amortized over the query stream; the sparse member costs $\mathcal{O}(M^3)$ per query irrespective of N_x and fits nothing. Nyström and the hierarchical construction of Appendix A.14 occupy the intermediate regimes. Remark 1 is the dial that moves along this family.

3.2 Probing

We probe (Alain & Bengio, 2016; Hewitt & Manning, 2019) a vision transformer (Dosovitskiy et al., 2020) and a VGG-style convolutional network, both trained from scratch on CIFAR-10 and read as

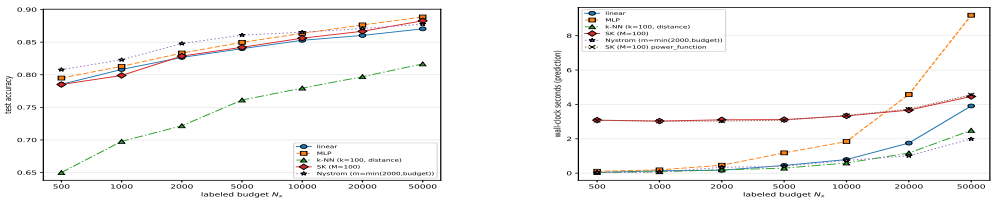


Figure 1: Accuracy and prediction wall-clock on CIFAR-10 over a frozen ImageNet ResNet-18, against the training subsample size N_x, for a linear head, a two-layer MLP, a k-NN vote, the sparse lazy readout at bandwidth M and Nyström KRR with m inducing points. All readouts are scored on the same $N_z = 10\,000$ test images. Appendix A.6 gives the full grid.

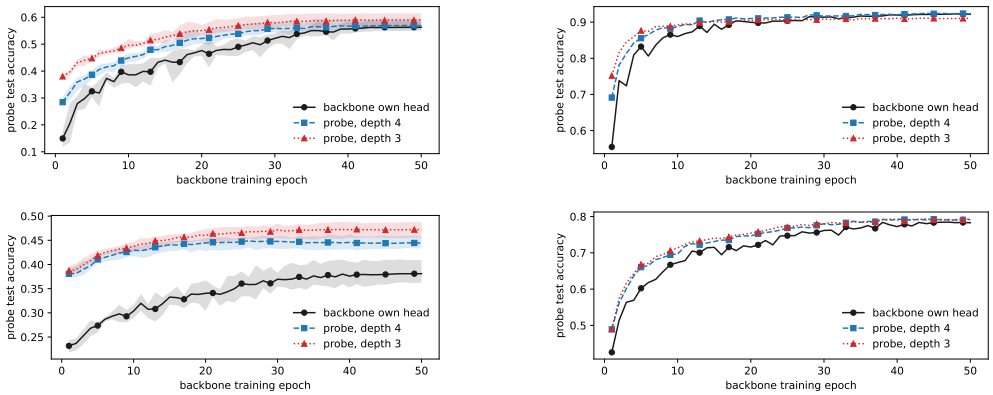


Figure 2: Test accuracy of the lazy readout (12) attached at the third and fourth of five blocks (blue, red), against the model's own trained head (black), at every epoch of backbone training. Top row: VGG-style convolutional network; bottom row: vision transformer. Left: $N_x = 1000$, four seeds (mean and min–max); right: $N_x = 50\,000$, one seed. Representations are flattened, and the readout fits nothing. Appendix A.7 gives the protocol and every measurement.

five sequential blocks: at every training epoch we truncate the feature extractor at a given block and attach the lazy readout (12), whose coefficients are the observed labels. Nothing is trained (Figure 2; Appendix A.7).

Reading a representation with a kernel readout rather than with the model's own head buys two things. *Accuracy*: at $N_x = 1000$ and convergence it reaches 0.472 on the transformer against 0.381 for that head, and 0.592 against 0.555 on the convolutional network. The gain is of the order of a point at $N_x = 50\,000$, and *how much of it the readout itself contributes depends on the architecture*: against a cross-validated ridge on the same features it is worth 1.6 points on the transformer and 0.5 on the convolutional network. The reason is geometric: cross-entropy collapses the convolutional features onto ten near-orthogonal class means, where any readout suffices (Appendix A.7).

And an error measure. Because the readout is an RKHS element, every prediction comes with the power function (4) evaluated at that query. It ranks errors better than chance at every depth, epoch and budget, its AUROC rising from 0.69 to 0.86 as training proceeds, so it is sharpest exactly where the readout would be deployed—a first, concrete step towards auditable predictions.

3.3 Learning with probes

The layer of Section 2.3 is differentiable, so it can be trained through rather than merely read. But on a training point a lazy readout reproduces that point's target whatever the representation, so its loss is constant in the backbone parameters and its gradient vanishes identically (Remark 3). The kernel layer thus *suggests* objectives one would have no reason to try otherwise, and we report one: matching distance matrices between consecutive representations, after the readout's own map, with the labels as the last term (Appendix A.8.1). It contains no likelihood and no logits.

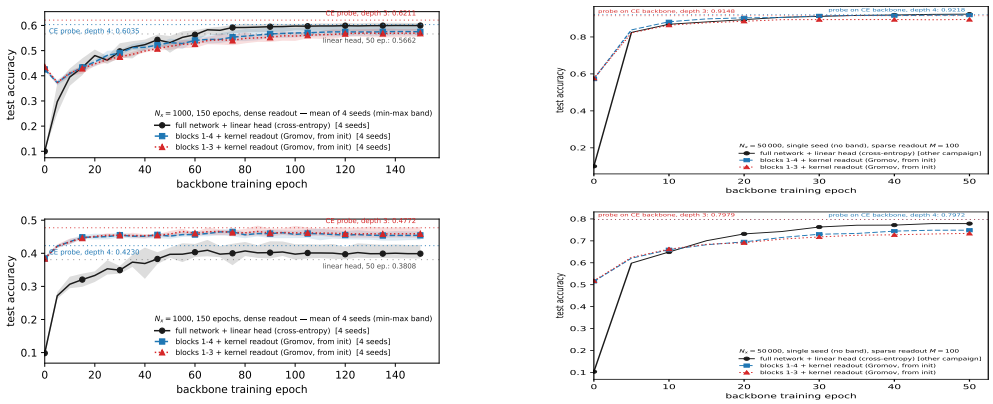


Figure 3: Test accuracy for backbones trained *from initialization* by the geometric objective (17) alone—no cross-entropy term—and read at the third and fourth of five blocks (blue, red); black is the same backbone trained by cross-entropy with its own head. Top: convolutional network; bottom: transformer. Left: $N_x = 1000$, 150 epochs, four seeds; right: $N_x = 50\,000$, 50 epochs, one seed. Dotted lines: probe on the cross-entropy backbone. Appendix A.8.1 gives the objective and every measurement.

Trained from initialization by that objective alone, and read at the fourth of five blocks, the transformer reaches 0.454 against 0.399 for its own linear head at the same protocol—5.5 points, with the same sign on all four seeds—while the convolutional network loses 2.5 points, also on all four seeds (Figure 3). A purely geometric criterion therefore matches or beats cross-entropy on one architecture and not on the other, at equal protocol and equal duration. The geometry it reaches is an attractor rather than an accident: the kernel contrast converges to 2.037 at $N_x = 1000$ and 2.002 at $N_x = 50\,000$, the same configuration with fifty times fewer images. But the contrast does not predict accuracy—the convolutional network attains the study's highest and loses, the transformer the lowest and wins—and at full budget both fall behind their heads. Whether a functional exists whose optimum is the representation a kernel readout wants remains open (Appendix A.8).

3.4 Reinforcement learning

On the LunarLander-v3 environment we place kernel readouts in two positions of an agent, and the two give opposite answers. *Added to the two internal paths of a Double DQN agent*, they raise the mean reward of the last fifty episodes from 174 to 246 over five seeds—and a parametric baseline matched to the same parameter count reaches 206, so the ordering $246 > 206 > 174$ separates what the construction contributes from what the added capacity does. Each module also helps on its own, and the across-seed range narrows from 27–244 to 218–257.

Substituted for both heads of a PPO agent, they reach the baseline's reward at $0.28\times$ its parameters—the parametric heads carry a hidden layer costing $\mathcal{O}(E^2)$ where the readout costs $\mathcal{O}(E)$—but they get there more slowly, trailing by 26 to 89 points at intermediate budgets before converging by 350k environment steps (Figure 4). They also halve the crash rate and hold a loss five times steadier than the parametric agent. That last figure needs the warning Section 3.3 already met: a lower PPO loss is not a better policy—it measures how little the policy changes between updates, and this agent spends a third of its episodes hovering. We therefore report the decomposition by outcome alongside the reward (Appendix A.9).

3.5 Out-of-distribution detection

The three preceding experiments use the error indicator as a diagnostic. Here it is the predictor. On the embeddings of a fine-tuned large language model, we place the power function (4) of Section 2.1 in the background-contrastive scheme of Ren et al. (2023)—the score of a query against in-distribution centers, minus its score against a background set—and use it directly to separate in-distribution from out-of-distribution prompts (Figure 5).

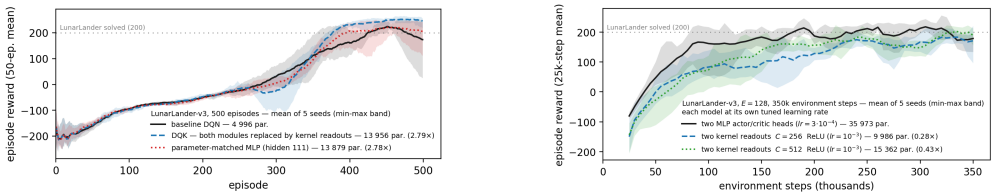


Figure 4: Reward on LunarLander-v3, five seeds, mean and min–max. Left, against episodes: a Double DQN agent with kernel modules added to two internal paths (DQK), against the original agent and against a parameter-matched parametric variant. Right, against environment steps so that both carry the same experience budget: a PPO agent whose actor and critic heads are replaced by the readout (9), each model at its own tuned learning rate, parameter counts in the legend. Losses in Figure 7—a lower loss is not a better policy (Appendix A.9).

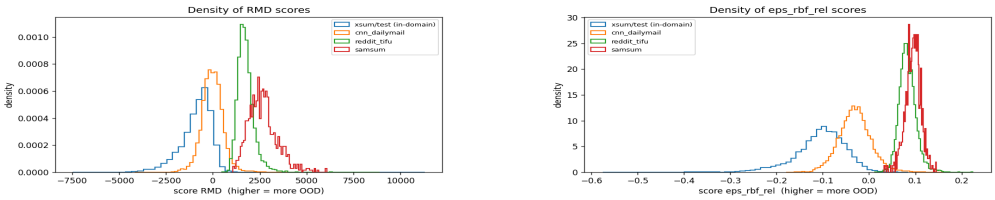


Figure 5: Out-of-distribution detection on large language model embeddings. The certificate of Section 2.1, used as a detector, separates in-distribution from out-of-distribution prompts without being fitted for that purpose. Appendix A.10 gives the benchmark, the replication of the reference method, and every score.

It exceeds the strongest input score of Ren et al. (2023) by a paired ΔAUROC of $+0.123$ on near-OOD, the regime the benchmark was built to be hard on, and the ordering is invariant across the numerical conventions we varied. The advantage survives a split-conformal scheme, where a detector is deployed rather than summarized (Appendix A.10). Nothing is fitted for the task: the quantity is the same one the readout already computes to bound its own error, evaluated at the query. This is the one experiment where the certificate is assessed as a detector rather than as an accessory ranking score, and the one at a scale other than CIFAR-10 and LunarLander. A companion study (Anonymous, 2026) takes that assessment further, over CIFAR-100 and ImageNet200 as well, and examines what the indicator measures when it is used as a selector rather than as a score.

4 Conclusion

Sparse kernels enter deep learning pipelines as modular components with a three-fold parameterization that may be fixed or learned, and across five experiments they integrate at competitive cost while carrying a computable, per-query error indicator.

A theme runs through those experiments and we state it plainly: *what is optimized is not what is measured*. An interpolating readout has a training loss that is constant by construction; a kernel-headed PPO agent holds a loss five times steadier for a policy that is not better; and the geometric contrast we introduced to explain the learning results does not predict accuracy across representations. Error-aware deep learning will need quantities that measure what one wants.

Limitations. **Tightness** of the certificate is an empirical property and must be reported on one task, one depth, one seed (Appendix A.11.1). **Scale**: we evaluate on CIFAR-10, LunarLander and one language-model benchmark; The per-query solve is cubic in M and flat in the training budget, on CPU and on a T4 alike, so what grows with the data is the M-NN lookup and not the solve—an approximate index is the natural replacement beyond the scales measured here (Appendix A.6).

Broader impact. Our contribution is methodological: sparse kernel readouts lower the cost of some transfer-learning workflows, and the per-query indicator is a step towards auditable predictions.

REPRODUCIBILITY STATEMENT

All experimental protocols are stated in full in the appendix, one section per experiment: Appendix A.5 gives the common setting (hardware, kernel, regularization, input maps), and Appendices A.6 to A.10 give, for each experiment, the architectures, optimizer settings, seed counts and every individual measurement behind the figures of Section 3. Hyperparameters are collected in Table 13. The error estimates of Section 2 and their composition along a pipeline are derived in Appendix A.11, with assumptions stated and a proof of Proposition 1; the empirical tightness study is Appendix A.11.1. All datasets used are public and referenced at first use, and the subsample keys are frozen across platforms, which is what makes the CPU and GPU passes of Appendix A.6 directly comparable.

AI USE STATEMENT

In this work, we used generative AI tools to *implement methods*. The software producing the numerical results of this paper was written with substantial and sustained assistance from a large language model over a period of several months: the experimental harness, the training and evaluation scripts, the data-processing pipelines, the benchmark replications of Appendix A.10, and the plotting code. We additionally used generative AI tools, among the subtasks whose disclosure is recommended, to create and edit software code more generally, to create and modify the figures of this paper, to draft and edit parts of the manuscript for readability, and to format the references.

We also used generative AI tools to *provide feedback on experiments*. Which experiments to run, which quantities to report and which comparisons would settle a question were decided by the authors; within that programme, the analysis proceeded through extensive iteration with a large language model, in which candidate readings of intermediate results were proposed, criticised and discarded on both sides. Every interpretation retained here was adopted by the authors on the evidence, and the substantive ones were converted into a measurement before being stated: the bandwidth sweep of Table 8 was run to test a proposed explanation of the full-budget reversal, the conditioning break it revealed was traced to representation collisions that were then counted, and the mode-purity figures of Appendix A.7 were measured to check the geometric reading of the transformer/convolutional asymmetry. Readings that did not survive such a check were dropped rather than reported.

We have not used generative AI tools to generate synthetic data sets: every dataset used is public and cited at first use, and every number reported was obtained by executing the above code on those datasets and on the hardware stated in Appendix A.5. We have not used generative AI tools to propose or refine the hypotheses of this work, nor to develop its theoretical model: the RKHS construction of Section 2, the error estimates it carries, and the surrogate certificate and its composition along a pipeline (Proposition 1, Appendix A.11) were conceived and stated by the authors, and the argument they rest on, the block-by-block replacement of each deployed readout by its local surrogate, is the authors' own.

We did, however, use generative AI tools to *assist in the writing of proofs*. The written argument of Proposition 1 and of Appendix A.11 was produced by a large language model working from the authors' statement and proof strategy, under the authors' direction, and was then checked line by line by the authors, who are responsible for its correctness. The remaining subtasks with required disclosure, namely assistance with translation and the support of qualitative and thematic data analysis, are not applicable to this work.

We have reviewed all AI-assisted work. AI-written code was read, exercised and debugged by the authors; where a result admits an independent check we report it, as with the machine-precision agreement between the power function and the Gaussian-process posterior variance (Appendix A.10), the replication of the published baseline of Ren et al. (2023) to within the tolerance set in advance, and the bit-level reproducibility of the sparse readout across two hardware platforms and two software stacks (Appendix A.6). One diagnostic reported in that appendix, the normal-equations conditioning artefact, was found and corrected in exactly this way. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

REFERENCES

Maxwell Adam, Zach Furman, and Jesse Hoogland. The loss kernel: A geometric probe for deep learning interpretability. *arXiv preprint arXiv:2509.26537*, 2025.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

Anastasios N. Angelopoulos and Stephen Bates. Conformal prediction: A gentle introduction. *Foundations and Trends in Machine Learning*, 16(4):494–591, 2023.

Anonymous. Out-of-distribution detection with kernel power functions, 2026. Working paper.

Lior Bohbot, Cyril Letrouit, Gabriel Peyré, and François-Xavier Vialard. Token sample complexity of attention. *arXiv preprint arXiv:2512.10656*, 2026.

Andrea Caponnetto and Ernesto De Vito. Optimal rates for the regularized least-squares algorithm. *Foundations of Computational Mathematics*, 7(3):331–368, 2007.

Benjamin Charlier, Jean Feydy, Joan Alexis Glaunès, François-David Collin, and Ghislain Durif. Kernel operations on the GPU, with autodiff, without memory overflows. *Journal of Machine Learning Research*, 22(74):1–6, 2021.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. When do neural networks outperform kernel methods? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 70–79, 2019.

Bernard Haasdonk. Reduced basis methods for parametrized PDEs: A tutorial introduction for stationary and instationary problems. In *Model Reduction and Approximation: Theory and Algorithms*, pp. 65–136. SIAM, 2017.

Mingxuan Han, Chenglin Ye, and Jeff M. Phillips. Local kernel ridge regression for scalable, interpolating, continuous regression. *Transactions on Machine Learning Research*, 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1693–1701, 2015.

John Hewitt and Christopher D. Manning. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

Byeongchang Kim, Hyunwoo Kim, and Gunhee Kim. Abstractive summarization of Reddit posts with multi-level memory networks. *arXiv preprint arXiv:1811.00783*, 2018.

Hyunjik Kim, George Papamakarios, and Andriy Mnih. The Lipschitz constant of self-attention. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *PMLR*, pp. 5562–5571, 2021.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Jaehoon Lee, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: An empirical study. *arXiv preprint arXiv:2007.15801*, 2020.

Philippe G. LeFloch, Jean-Marc Mercier, and Shohruh Miryusupov. Reproducing kernel methods for machine learning, PDEs, and statistics. *arXiv preprint arXiv:2402.07084*, 2025.

Jeremiah Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax-Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. In *Advances in Neural Information Processing Systems*, volume 33, pp. 7498–7512, 2020.

Julien Mairal. End-to-end kernel learning with supervised convolutional kernel networks. *Advances in Neural Information Processing Systems*, 29, 2016.

Giacomo Meanti, Luigi Carratino, Lorenzo Rosasco, and Alessandro Rudi. Kernel methods through the roof: Handling billions of points efficiently. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1797–1807, 2018.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, volume 20, 2007.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, Victor Greiff, David Kreil, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need. In *International Conference on Learning Representations (ICLR)*, 2021.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Jie Ren, Stanislav Fort, Jeremiah Liu, Abhijit Guha Roy, Shreyas Padhy, and Balaji Lakshminarayanan. A simple fix to Mahalanobis distance for improving near-OOD detection. *arXiv preprint arXiv:2106.09022*, 2021.

Jie Ren, Jiaming Luo, Yao Zhao, Kundan Krishna, Mohammad Saleh, Balaji Lakshminarayanan, and Peter J. Liu. Out-of-distribution detection and selective generation for conditional language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Robert Schaback. Greedy adaptive local recovery of functions in Sobolev spaces. *Numerical Algorithms*, 2025.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA, 2002.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Rahul Singh and Suhas Vijaykumar. Kernel ridge regression inference. *arXiv preprint arXiv:2302.06578*, 2023.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Hervé Jégou, and Armand Joulin. Augmenting self-attention with persistent memory. *arXiv preprint arXiv:1907.01470*, 2019.

Yiyou Sun, Yifei Ming, Xiaojin Zhu, and Yixuan Li. Out-of-distribution detection with deep nearest neighbors. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2022.

Johan A. K. Suykens. SVD revisited: A new variational principle, compatible feature maps and nonlinear extensions. *Applied and Computational Harmonic Analysis*, 40(3):600–609, 2016.

Shuai Tang and Virginia R. de Sa. Deep transfer learning with ridge regression. *arXiv preprint arXiv:2006.06791*, 2020.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, pp. 2094–2100, 2016.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer, 2005.

Holger Wendland. *Scattered Data Approximation*, volume 17 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2005.

Tizian Wenzel, Gabriele Santin, and Bernard Haasdonk. Analysis of structured deep kernel networks. *Journal of Computational and Applied Mathematics*, 476:116975, 2026.

Andrew Gordon Wilson and Hannes Nickisch. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *PMLR*, pp. 1775–1784, 2015.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *PMLR*, pp. 370–378, 2016.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

A Appendix

This appendix supports Section 3. Appendices A.6 to A.10 give the setting, the protocol and the complete measurements of each experiment; Appendix A.11 develops the certifiable surrogate and its error estimates; Appendices A.12–A.14 collect additional methodological constructions, and Appendix A.15 a tabular benchmark. The section immediately below supports Section 2.3, stating which instances of (9) and (10) the experiments use.

A.1 Notation

We let $x = (x^1, \dots, x^N) \in \mathbb{R}^{N \times D_x}$ be a matrix, with rows $x^n = (x^n_1, \dots, x^n_{D_x}) \in \mathbb{R}^{D_x}$. The ℓ^p norm is $|x^n|_p = \left(\sum_{d=1}^{D_x} |x^n_d|^p\right)^{1/p}$ for vectors and $\|x\|_p = \left(\sum_{n=1}^{N} |x^n|_p^p\right)^{1/p}$ for matrices. For $\bar{x} \in \mathbb{R}^{N \times D_x}$, we write $\langle x, \bar{x} \rangle = \sum_{n,d} x^n_d \bar{x}^n_d$ for the standard ℓ_2 inner product. We use vectorized notation for function evaluations: if $y : \mathbb{R}^{D_x} \to \mathbb{R}^{D_y}$ then $y(x) = (y(x^1), \dots, y(x^N)) \in \mathbb{R}^{N \times D_y}$, while for $k : \mathbb{R}^{D_x} \times \mathbb{R}^{D_x} \to \mathbb{R}$ we have $k(\cdot, x) : \mathbb{R}^{D_x} \to \mathbb{R}^N$ and $k(x, x) = (k(x^i, x^j))_{i,j} \in \mathbb{R}^{N \times N}$. If $M \leq N$ and $\sigma = (\sigma^1, \dots, \sigma^M) \in \{1, \dots, N\}^M$ is a tuple of indices, then $x^\sigma = (x^{\sigma^1}, \dots, x^{\sigma^M})$

denotes the corresponding subset. We let $\Sigma_M^N = \{\sigma = (\sigma^1, \dots, \sigma^M) \in \{1, \dots, N\}^M : \sigma^n \neq \sigma^m \text{ if } n \neq m\}$ be the set of M-tuples of distinct indices, abbreviated as Σ when no ambiguity arises.

Finally, $I_N \in \mathbb{R}^{N \times N}$ is the identity matrix, $\mathbf{1}_\Omega(x)$ is the indicator function of a set Ω, and $\mathrm{ReLU}(a) = \max(a, 0)$ denotes the rectified linear activation.

A.2 Assembling the local spaces

This appendix gives the closed form of the sparse Lagrange functions announced in Section 2.2 and the three properties used there.

Assembling the local spaces with *independent* parameters—the localization of (1), whose coefficients are neighborhood-relative and cannot be shared across cells—yields the direct sum $\bigoplus_{\sigma \in \Sigma} \mathcal{H}_{k,\sigma}$, an object with $\#\Sigma \times M \times D_y$ parameters: possible, but enormous, and discontinuous with no structure linking adjacent cells. Sharing values instead, as (3) allows, the entries of $\sigma(z)$ being distinct so that each global index n occupies at most one slot of the local ordering, we may set

$$\Psi_k^n(z, x^\sigma) \;=\; \sum_{\sigma \in \Sigma} \mathbf{1}_{\Omega^\sigma}(z) \sum_{m=1}^{M} \delta_{n,\sigma^m}\, \psi_k^m(z, x^\sigma), \qquad n = 1, \dots, N, \tag{11}$$

each inner sum carrying at most one nonzero term. Three properties follow. *(i) Sparsity.* $\Psi_k^n(\cdot, x^\sigma)$ vanishes outside $\bigcup\{\Omega^\sigma : n \in \{\sigma^1, \dots, \sigma^M\}\}$, so at most M of the N functions are nonzero at any given z—whence the name, and the $\mathcal{O}(M^3)$ per-query cost. *(ii) The Lagrange property survives localization:* $\Psi_k^n(x^j, x^\sigma) = \delta_{n,j}$ for all $n, j \leq N$, since $\sigma^1(x^j) = j$ and $\psi_k^1(x^j, x^{\sigma(x^j)}) = 1$. *(iii) Cellwise reduction:* for $z \in \Omega^\sigma$, $\Psi_k(z)\,\theta_y = \psi_k(z, x^\sigma)\,\theta_y^\sigma$, so that on each cell the global object *is* the local predictor of $\mathcal{H}_{k,\sigma}$, its M coefficients being read off the single global table θ_y—the sharing announced above.

The norm of (6) follows cellwise, $\|y_{k,\theta_y}\|^2_{\mathcal{H}_k^M} = \sum_{\sigma \in \Sigma} \|y_{k,\theta_y^\sigma}\|^2_{\mathcal{H}_{k,\sigma}} = \sum_{\sigma \in \Sigma} (\theta_y^\sigma)^T k(x^\sigma, x^\sigma)^{-1}\, \theta_y^\sigma$.

A.3 Representer theorem and the squared loss

For any functional $\mathcal{L}$ of finitely many point evaluations, taken here at the centers x, the minimization problem $\inf_{f \in \mathcal{H}_k} \mathcal{L}(f(x))$ over the whole space $\mathcal{H}_k$ is attained on the finite-dimensional span of the kernel sections at those points, $\mathcal{H}_k^x$ (Wenzel et al., 2026; Schölkopf & Smola, 2002): minimizers can be searched in the form (1), equivalently (3). This holds for *any* such loss, convex or not; no closed form is implied in general.

Remark 2 (The squared-loss case)**.** *As a direct application, take the empirical squared loss on the observed data x, y, $\mathcal{L}(f(x)) = \|f(x) - y\|^2$: by* (3) *$y_{k,\theta_y}(x) = \theta_y$, and the minimizer over $\mathcal{H}_k$ is $y_{k,y} = \psi_k(\cdot)\, y$, the interpolant of the data—the readout used throughout Appendix A.6. We never rely on this closed form:* (2) *is a choice of basis made before any loss enters, and the parameters θ_y are trained below under arbitrary losses; the squared-loss case only explains the familiar look of $\psi_k(\cdot)\, y$.*

A.4 Instances of the kernel layer

This section states in full the instances summarized in Table 1.

(i) Lazy readout—nothing is learned. Both embeddings are the identity, the centers are the layer-$(l-1)$ features of the data x and the targets the *observed* labels y, a special case of (10):

$$y_{k,\theta}^l(\cdot) \;=\; \Psi_k(y_\theta^{l-1,\sigma}(\cdot), y_\theta^{l-1,\sigma}(x))\, y^\sigma. \tag{12}$$

This is available whenever y is observed, as is typical of the last layer of a classification problem, and on frozen θ it requires no parameter training at all. Sections 3.1 and 3.2 read frozen and training representations with it, and Section 3.3 trains a backbone *through* it, where the layer still has no parameter of its own.

Remark 3 (A degenerate objective)**.** *Take a loss of difference form, $\ell(a, b) = \varphi(a - b)$ with φ convex and minimal at the origin, and minimize it over the parameter-free layer* (12) *evaluated* at

its own centers, *that is* $\mathcal{L}\big(y^l_{k,\theta}(x),\, \theta_y\big)$ *with* $\theta_y = y^\sigma$ *the observed targets. The Lagrange property* $\psi^n_k(x^m) = \delta_{n,m}$ *of* (2) *gives* $y^l_{k,\theta}(x) = \theta_y$ *for* every θ*, so the objective is constant and*

$$\nabla_\theta\, \mathcal{L}\big(y^l_{k,\theta}(x),\, \theta_y\big) = 0. \tag{13}$$

The layer is nonetheless usable: what the gradient of $\mathcal{L}$ *cannot supply must come from elsewhere, and Appendix A.8 describes the schemes we tested and what they achieve.*

(ii) Shared embedding, symmetric. Both arguments share one embedding, here the identity, $P_{\theta_z} = P_{\theta_x} = I$, and the layer is written in the Lagrange formulation (3) as

$$y^l_{k,\theta,\theta_x,\theta_y}(\cdot) \;=\; \Psi_k(y^{l-1}_\theta(\cdot), \theta_x)\, \theta_y. \tag{14}$$

The θ_y are not observed labels but free parameters: the layer's values at the centers. The centers θ_x are a slot of their own, and are learned together with θ_y for the Double DQN agent of Section 3.4—which therefore no longer interpolates any observed sample.

(iii) Distinct embeddings, full model. All parameters of (9) are free, with

$$y^l_{k,\theta,\theta_z,\theta_x,\theta_y}(\cdot) \;=\; k\big(P_{\theta_z}(\cdot),\; P_{\theta_x}(\cdot)\big)\, \theta_y. \tag{15}$$

This is the actor and critic heads of the PPO agent (Section 3.4) and the tabular readout of Appendix A.15, where the *adapter maps* are linear:

$$P_{\theta_z}(\cdot) = y^{l-1}_\theta(\cdot)\, \theta_z, \quad \theta_z \in \mathbb{R}^{D^{l-1},E}. \tag{16}$$

Viewed as a function of the projected query, the layer remains an element of $\mathcal{H}_k$, which is what makes it amenable to the error certificate of Appendix A.11.

Table 1: Instances of (9) used in this paper. Instance (i) trains nothing: its centers are the data and its coefficients the observed labels, which is available whenever labels are observed. In the others θ_y holds free values at the centers—analogous to inducing-point values in sparse variational Gaussian processes, or to the weights of a final linear layer—so that $\theta_y = 0$ yields the zero function exactly, and a layer placed in parallel with an existing path starts from that path alone.

	learned	form	used in
(i) lazy	nothing	(12)	§3.1, §3.2, §3.3 (CNN)
(ii) symmetric	θ_x, θ_y	(14), $P_{\theta_z} = P_{\theta_x} = I$	§3.4 (DQN)
(iii) full layer	$P_{\theta_z}, P_{\theta_x}, \theta_x, \theta_y$	(9)	§3.4 (PPO), App. A.15

The instances at a glance. Two remarks carry across the instances. Minibatch optimizers make the relevant N of Remark 1 the batch size, so small batches favor the dense form (9) and larger ones the sparse form (10). *In the learning phase all our experiments use the dense form*, the batch being small enough for a single square solve; the sparse form appears only when reading at full budget, where a dense Gram over $50\,000$ centers would not fit in memory. Training times are roughly comparable to those of a neural layer of similar width—the PPO agent of Section 3.4 costs $2.15\times$ its parametric baseline at matched hyperparameters, so the same order of magnitude and not a change of scale. And laziness is irrelevant during training, the centers and embeddings moving at every step: what is retained is differentiability.

A.5 Experimental details

Unless stated otherwise, experiments at $N_x = 1000$ and the reinforcement learning campaigns are run on CPU, on a single desktop machine[2]. All experiments at the full $N_x = 50\,000$ budget are run on a GPU (a Colab T4), as is the extended probing campaign of Appendix A.8; the transfer benchmark is run on both, which is what makes the two platforms comparable (Appendix A.6).

[2]AMD Ryzen 9 7950X3D, 16 cores / 32 logical, 64 GB of RAM. No thread setting is imposed; the process occupies the 32 logical cores, measured by sampling. We do not report `torch.get_num_threads()`, which returns 16 and does not reflect actual use, the BLAS carrying its own thread pool.

Every experiment works in the interpolation regime, with the Tikhonov regularization parameter set to 10^{-9}. The M-NN lookup is an exact brute-force search—pairwise distances followed by a partial sort—on both platforms; approximate indices such as FAISS[3] are supported but not used in the runs reported here, so that no measurement depends on an approximation of the neighbour set. The kernel is the Matérn-$1/2$ kernel $k(x, y) = \exp(-|S(x) - S(y)|_2)$ throughout, a choice justified by the sweep of Appendix A.7: a Gaussian kernel on these representations produces Gram matrices conditioned around 10^9 where the Matérn ones sit at 10^4, and loses several accuracy points that no setting of the regularization recovers.

What does change across experiments is the input map S, which follows the geometry of the representation being read; each appendix states the map it uses. On the penultimate features of a frozen convolutional backbone, S standardizes each coordinate on the training subsample and then divides by the median pairwise distance of the centers. On representations read *inside* a network, S normalizes each row to unit length before the same median rescaling; for unit vectors $|a - b|^2 = 2 - 2\langle a, b\rangle$, so nearest neighbours in the Euclidean sense are nearest neighbours in the cosine sense and the lookup is unchanged.

A.6 Transfer learning details

Setting. We use a ResNet-18 pretrained on ImageNet[4] as a frozen feature extractor, and transfer to the CIFAR-10 dataset[5]. The linear head is a single $512 \to 10$ linear layer; the MLP head is $512 \to 512 \to \text{ReLU} \to 10$. Both are trained with AdamW using a cross-entropy loss and a learning rate of 10^{-3}. The sparse kernel readout uses a bandwidth $M = 100$ unless stated otherwise.

We compare five readouts over subsample sizes N_x of the CIFAR-10 training set, from 500 to $50\,000$, all scored on the same $N_z = 10\,000$ test images: a linear head, a two-layer MLP, a k-NN vote, the sparse lazy readout (12) at bandwidth M, and Nyström KRR with m inducing points[6]—the global counterpart of the sparse construction, solving one $m \times m$ system instead of one $M \times M$ system per query. Figure 1 reports accuracy and prediction wall-clock; the paragraphs below give the full grid, the cost decomposition, the stability checks and the GPU pass.

Additional transfer comparisons. The tables below give the full sweep over the free parameter of each readout (Table 3), the operating points at full budget (Table 2), and the cost decomposition of the sparse readout. Protocol, features and splits are those stated above. All timings are single measurements, CPU wall-clock, on the machine of Appendix A.5, taken in one process over a single campaign; no thread setting is imposed and the process occupies the 32 logical cores. Re-measuring one point in isolation can differ by some tens of percent.

Parity with the trained heads. At full budget the sparse readout reaches 0.8827 at $M = 100$ and 0.8941 at $M = 500$—the widest bandwidth thus overtaking the trained head—against 0.8886 for the MLP and 0.8704 for the linear probe, with no fitting step: the coefficients are the observed labels, as in instance (i) of Section 2.3.

Regimes, in detail. At $M = 100$ the local solve costs 3.06 seconds at $N_x = 500$ and 3.54 at $N_x = 50\,000$—a sixth more for a hundredfold increase in data—while the M-NN lookup grows by a factor 34: the solve is $\mathcal{O}(M^3)$ per query irrespective of N_x. Prediction then costs 4.46 seconds at full budget against 9.19 for the MLP, though no longer less than the 3.91 of the linear probe. Against $M = 500$, Nyström leads at budgets up to $5\,000$, the ordering reversing from $10\,000$ on; against the $M = 100$ of Figure 1 it stays ahead until full budget. At $50\,000$ the sparse readout is ahead at both bandwidths, and beats $m = 10\,000$ on both axes (0.8941 in 39.6 s against 0.8913 in 42.9 s). Neither

[3] `https://ai.meta.com/tools/faiss/`

[4] Weights available at `https://download.pytorch.org/models/resnet18-f37072fd.pth`.

[5] Loaded via `torchvision.datasets.CIFAR10`: `https://docs.pytorch.org/vision/main/generated/torchvision.datasets.CIFAR10.html`.

[6] We benchmark against Nyström rather than against a hierarchical construction such as Han et al. (2022): it is the more widely used baseline, and it exhibits the same behaviour that matters here—a single global solve on a reduced set of representers, whose cost is independent of the query and whose accuracy saturates with the number of representers.

Table 2: Operating points at full budget ($N_x = 50\,000$, $N_z = 10\,000$). Time is prediction wall-clock, excluding the error indicator; the readouts marked † require no training. On the accuracy-cost frontier, Nyström at $m \leq 2\,000$, the sparse readout at $M \in \{100, 500\}$ and the MLP are non-dominated; the other rows are dominated by one of them.

Readout	Test accuracy	Time (s)
k-NN vote, $k = 10$ (distance)†	0.8443	2.29
k-NN vote, $k = 100$ (distance)†	0.8166	2.49
k-NN vote, $k = 500$ (distance)†	0.7842	2.64
Linear probe	0.8704	3.91
MLP $512 \to 512 \to 10$	0.8886	9.19
Nyström KRR, $m = 1\,000$	0.8680	0.55
Nyström KRR, $m = 2\,000$	0.8776	1.99
Nyström KRR, $m = 5\,000$	0.8852	11.26
Nyström KRR, $m = 10\,000$	0.8913	42.85
Sparse readout, $M = 10$†	0.8573	1.27
Sparse readout, $M = 100$†	0.8827	4.46
Sparse readout, $M = 500$†	**0.8941**	39.62

Table 3: Test accuracy for every readout and every labeled budget N_x. Nyström uses $m = \min(2\,000, N_x)$, so at $N_x \leq 2\,000$ it uses every available point and is exact dense KRR rather than an approximation; at $N_x = 500$ it then coincides with the sparse readout at $M = 500$, both reducing to the same closed-form regressor on the whole subsample. The single degenerate entry, k-NN at $k = 500$ with uniform weights and $N_x = 500$, is chance level by construction: every query then has the entire training set as its neighbourhood, the ten classes receive near-equal votes, and ties resolve to the first index; distance weighting breaks the symmetry. The MLP is the only entry that is not bit-reproducible on this machine: its 11 700 optimiser steps involve multithreaded reductions whose summation order is not fixed, and repeated runs settle on one of two trajectories differing by up to $7 \cdot 10^{-4}$ in accuracy. We report one run; every other readout reproduces exactly, across repetitions and across hardware.

N_x	500	1 000	2 000	5 000	10 000	20 000	50 000
Linear probe	0.7853	0.8081	0.8266	0.8400	0.8528	0.8603	0.8704
MLP $512 \to 512 \to 10$	0.7949	0.8129	0.8333	0.8497	0.8635	0.8764	0.8886
k-NN, $k = 10$, uniform	0.7025	0.7213	0.7569	0.7836	0.8011	0.8179	0.8421
k-NN, $k = 10$, distance	0.7030	0.7270	0.7596	0.7875	0.8046	0.8211	0.8443
k-NN, $k = 100$, uniform	0.6307	0.6918	0.7185	0.7597	0.7778	0.7963	0.8156
k-NN, $k = 100$, distance	0.6500	0.6978	0.7217	0.7612	0.7795	0.7970	0.8166
k-NN, $k = 500$, uniform	0.1000	0.5596	0.6176	0.7044	0.7352	0.7578	0.7836
k-NN, $k = 500$, distance	0.6606	0.6141	0.6317	0.7085	0.7379	0.7600	0.7842
Sparse readout, $M = 10$	0.7270	0.7517	0.7760	0.8014	0.8196	0.8358	0.8573
Sparse readout, $M = 100$	0.7850	0.7989	0.8286	0.8421	0.8561	0.8667	0.8827
Sparse readout, $M = 500$	0.8078	0.8203	0.8416	0.8586	0.8703	0.8778	**0.8941**
Nyström KRR	0.8078	0.8230	0.8478	0.8608	0.8652	0.8709	0.8776

construction dominates the other overall: on the accuracy-cost frontier at full budget Nyström holds the two cheapest useful points and the sparse readout the two most accurate, with the MLP between them, while $M = 10$, the linear probe and every k-NN setting are dominated (Table 2). All costs are measured once, on CPU, on a single machine; the two methods do not parallelize alike, so the ordering is hardware-dependent.

Matched neighbourhoods. At $k = M$ the two local readouts see exactly the same neighbours of each query. Table 4 isolates the difference that remains, which is only in how those neighbours are combined.

Table 4: Accuracy gap in percentage points, sparse readout minus k-NN with distance weighting, at matched neighbourhood $k = M$. The sparse readout is ahead at every budget and every neighbourhood size; the margin grows with the neighbourhood and narrows as the budget grows.

N_x	500	1 000	2 000	5 000	10 000	20 000	50 000
$k = M = 10$	$+2.40$	$+2.47$	$+1.64$	$+1.39$	$+1.50$	$+1.47$	$+1.30$
$k = M = 100$	$+13.50$	$+10.11$	$+10.69$	$+8.09$	$+7.66$	$+6.97$	$+6.61$
$k = M = 500$	$+14.72$	$+20.62$	$+20.99$	$+15.01$	$+13.24$	$+11.78$	$+10.99$

Stability. Two sources of variation were checked, both on the sparse readout at $M = 100$. Resampling the budget at $N_x = 10\,000$ over four draws gives a standard deviation of $0{,}0013$ in accuracy, against $0{,}0028$ for Nyström on the same draws—the local solves depend only on each query's neighbours, so a different subsample perturbs them less than it perturbs a global fit. And sweeping the lengthscale heuristic of Appendix A.5 over a fourfold range moves accuracy by at most $0{,}79$ points. This last figure is smaller than one might expect, and the reason is dimensional: after per-coordinate standardization in dimension 512, pairwise distances concentrate around $\sqrt{2d} \approx 32$ with a relative spread of order $1/\sqrt{d}$, so the median heuristic is doing little beyond a dimensional normalization. What matters at this dimension is the standardization, not the fine choice of bandwidth.

Bandwidth selection. Accuracy increases with M at every budget, and so does cost: at $N_x = 50\,000$ the three bandwidths give 0.8573, 0.8827 and 0.8941 for 1.27, 4.46 and 39.62 seconds. $M = 100$, used throughout the paper, is the knee of that curve—it recovers all but 1.1 points of the $M = 500$ accuracy at a ninth of the cost. This is guidance, not a selection rule: we did not study data-driven choices of M.

Nyström with more inducing points. Given a larger m the global solve keeps improving, and the two constructions cross rather than order each other: at $N_x = 20\,000$, $m = 10\,000$ regains the lead over $M = 500$, whereas at $N_x = 50\,000$ it does not, and costs more (Table 5).

Table 5: Nyström KRR with m inducing points, at the two largest budgets, against the sparse readout at $M = 500$.

	$m = 1\,000$	$m = 2\,000$	$m = 5\,000$	$m = 10\,000$
$N_x = 20\,000$, accuracy	—	0.8709	0.8783	**0.8835**
$N_x = 50\,000$, accuracy	0.8680	0.8776	0.8852	0.8913
$N_x = 50\,000$, time (s)	0.55	1.99	11.26	42.85
$M = 500$, accuracy	0.8778 at $N_x = 20\,000$		**0.8941** at $N_x = 50\,000$, 39.62 s	

Where the sparse cost goes. Table 6 separates the stages of a sparse prediction, on both devices. The M-NN index is built once per training set and cached; the lookup grows with N_x; the local solve does not. The last pair of columns is the cost of the error indicator (7), which our implementation obtains by rebuilding the local system rather than reusing the one already formed for the prediction, and which is therefore an upper bound—on either device it roughly doubles the total, and sharing the factorisation between the two paths is the largest single saving available to an implementation.

GPU. The same benchmark, on the same frozen subsets, was run on a Colab T4 (Table 7). Only the device and the M-NN backend change: the neighbour search is device-native (`cdist+topk`) rather than host-side, which returns the same neighbours since both are exact brute force. The grid was run three times in one session; we report the last pass, the first carrying one-off CUDA and BLAS initialisation that is visible only at the smallest budget (up to a factor 48 there, and within 10% from $N_x = 1\,000$ on). Timings are single measurements on a shared cloud runtime. The two passes run on different software stacks—torch 2.13 / NumPy 2.5 on CPU against torch 2.11 / NumPy 2.0 on the GPU—and on images fetched from different mirrors, so the accuracies below agree because the subsets are frozen, not because the environments match.

Table 6: Cost decomposition of the sparse readout, in seconds, CPU against GPU, query batch size 512 on CPU and 1 024 on the GPU (the batch affects memory, not the result). Solve time is prediction minus lookup. On the GPU the solve is constant in the budget to within 0.1% at $M = 500$ and 3% at $M = 100$, and all the growth is in the neighbour search; on CPU the same quantity is noisier—single measurements on an 11 GB working set at $M = 500$—and shows no trend either. Every timing is a single measurement.

		index		lookup		prediction		solve		indicator	
M	N_x	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
10	500	0.215	0.001	0.017	0.004	0.143	0.019	0.126	0.015	0.145	0.021
10	50 000	0.948	0.021	0.896	0.262	1.274	0.285	0.378	0.023	1.295	0.283
100	500	0.026	0.001	0.027	0.005	3.085	0.326	3.058	0.321	3.070	0.326
100	50 000	0.949	0.021	0.925	0.262	4.463	0.593	3.538	0.331	4.557	0.590
500	500	0.051	0.001	0.046	0.006	45.111	3.583	45.065	3.577	46.140	3.608
500	50 000	1.012	0.021	0.985	0.265	39.623	3.837	38.638	3.572	39.587	3.862

Table 7: CPU against GPU at full budget ($N_x = 50\,000$). Time is prediction wall-clock, excluding the error indicator. Nyström is reported at $m \leq 2\,000$ only: at $m \geq 5\,000$ our normal-equations implementation of the baseline is not usable in float32 (see below).

Readout	acc. CPU	acc. GPU	Δacc	CPU (s)	GPU (s)
Linear probe	0.8704	0.8705	$+0.0001$	3.91	11.55
MLP $512 \to 512 \to 10$	0.8886	0.8920	$+0.0034$	9.19	15.04
k-NN, $k = 10$ (distance)	0.8443	0.8443	0.0000	2.29	0.304
k-NN, $k = 100$ (distance)	0.8166	0.8166	0.0000	2.49	0.239
k-NN, $k = 500$ (distance)	0.7842	0.7842	0.0000	2.64	0.252
Nyström KRR, $m = 1\,000$	0.8680	0.8689	$+0.0009$	0.55	0.053
Nyström KRR, $m = 2\,000$	0.8776	0.8781	$+0.0005$	1.99	0.168
Sparse readout, $M = 10$	0.8573	0.8572	-0.0001	1.27	0.285
Sparse readout, $M = 100$	0.8827	0.8826	-0.0001	4.46	0.593
Sparse readout, $M = 500$	0.8941	0.8941	0.0000	39.62	3.837

The GPU slows down the two baselines that train. They are the two neural ones: at batch 128, thirty epochs over 50 000 points are 11 700 optimiser steps, each launching kernels too small to amortise their own submission, and the linear probe—the lightest arithmetic of all—suffers most. A larger batch would cut those launches, but at fixed epochs and learning rate it also cuts the number of gradient steps, so throughput there is traded against convergence: every batch size is a compromise, and we report a standard one.

The neighbour search must run on the device. Routing the same exact search through a host-side library, with the query chunks copied to the host and the indices back, costs a factor 11 on the total: 13.2 seconds against 1.2 at $M = 100$ and full budget, for identical predictions. The penalty exceeds the cost of the search itself, the transfers serialising with the device work. We report it because it is the first thing an implementation of this construction will meet, and because it inverts the ordering: measured through the host-side lookup, the sparse readout appears *slower* on the GPU than on the CPU.

Determinism across hardware, and its one exception. The sparse readout reproduces its CPU accuracy to 10^{-4} or better at full budget, and exactly at $M = 500$, as do all three k-NN settings—across different hardware, different software stacks and different image mirrors. The exception is instructive and is ours, not the baseline's: at $m \geq 5\,000$ the Nyström implementation solves the normal-equations form, which squares the condition number of the design. Measured at $m = 5\,000$, $N_x = 50\,000$: condition number $2.8 \cdot 10^9$, and an accuracy falling from 0.885 in float64 to 0.208 in float32. Solving the same problem through a Cholesky factorisation of K_{mm} followed by a QR on the augmented system never forms that product, and recovers the float64 result to four decimals in

float32, seven times faster than float64. This is a property of the route, not of the estimator, nor of single precision: the local solves of the sparse readout, whose conditioning is set by M and not by the training-set size, are unaffected.

A.7 PROBING DETAILS

Setting. The two backbones are a vision transformer (ViT) and a VGG-style convolutional network, both trained from scratch on CIFAR-10 at $32\times32\times3$ resolution and both read as five sequential blocks. The transformer follows the standard architecture[7] at patch size 4 (hence 64 tokens), embedding dimension 128, 4 heads, 4 transformer blocks and dropout 0.3, the fifth block being the classification head. The convolutional network follows VGG in four stages[8], the fifth block applying max-pooling, global average pooling and a $512 \to 512$ projection.

Training is identical for both: AdamW, learning rate 10^{-3}, weight decay $5 \cdot 10^{-4}$, label smoothing 0.05, batch size 100, cosine-annealed over 50 epochs, with random 32×32 crops from 4-pixel padding and random horizontal flips; the test set is not augmented. A checkpoint is kept at every epoch.

The probes use the Matérn-$1/2$ kernel with the cosine map of Appendix A.5, chosen on the evidence of the sweep reported at the end of this appendix, where a Gaussian kernel loses several accuracy points that no setting of the regularization recovers. Coefficients are the observed labels; nothing is trained. The dial of Remark 1 is exercised by the budget: at $N_x = 1000$ the readout is the *dense* member, every center entering one square solve, whereas at $N_x = 50\,000$ it is the sparse member at bandwidth $M = 100$. Representations are *flattened*[9] on both architectures: the transformer's 64 tokens of dimension 128 are concatenated, so the probed dimension is 8192 at both depths, and convolutional maps are flattened, so it varies with depth—4096, 8192 and 512 at depths 3, 4 and the head. Averaging the transformer's tokens instead, which collapses each image to the embedding dimension 128, is reported below as a comparison: it erases the depth effect. At $N_x = 1000$ we report four seeds; at $N_x = 50\,000$, one run per curve, for computational reasons.

Two features of the protocol matter. The probed task *is* the task the backbone is being trained on, so a probe that outperforms a deeper one is not reporting a transfer gap: it is reporting that a block of the model discards information relevant to the objective the model is optimizing. And the model's own classification head is trained end-to-end on that objective while the probes are not, so the head curve is a reference rather than a competitor.

At every one of the 50 training epochs we truncate the feature extractor at a given block, attach the readout, and measure test accuracy (Alain & Bengio, 2016; Hewitt & Manning, 2019), so that the experiment sweeps depth and training time at once. Because the probe fits nothing, it measures what a representation *contains* rather than what an optimizer extracts from it. And it settles what Appendix A.8 depends on: whether a kernel readout can serve as a head at all, and in which of the two regimes of Remark 1.

Reading earlier and wider beats the model's own head, and a kernel readout does so without fitting anything. At $N_x = 1000$ and convergence, read at the third of five blocks, the kernel readout reaches 0.472 on the transformer against 0.381 for its trained linear head—9.1 points—and 0.592 against 0.555 on the convolutional network, 3.7 points. To attribute that gain we add the canonical probing baseline, a cross-validated ridge regression on the *same* features (Table 9): most of it comes from the choice of representation and about a fifth from the readout itself.

The margin closes as the budget grows. At $N_x = 50\,000$ the same probe is ahead of the head by 1.9 points instead of 9.1, and the two aggregations, which differ by 17 points on an untrained transformer, differ by 0.6 at convergence—0.797 flattened against 0.791 averaged. The readout recovers what a head has not yet extracted, so it helps least where the head had the means to learn.

[7] Structurally that of Dosovitskiy et al. (2020); the configuration is sized for 32×32 inputs and is far smaller than ViT-B/16.

[8] $(3\to64, 64\to64)$, $(64\to128, 128\to128)$, $(128\to256, 256\to256)$, $(256\to512, 512\to512)$, each convolution followed by batch normalization and a ReLU, each of the first three stages closed by a 2×2 max-pooling.

[9] Keeping the token structure and reading each position separately is also possible, and we measured it: it is slightly inferior on these representations, and it changes no conclusion, so we report the flattened variant throughout.

On the attribution, the kernel readout is ahead of a cross-validated ridge on the same features by 1.6 points on the transformer and 0.5 on the convolutional network at $N_x = 1000$, and it is so with no fitted coefficient and no regularization to select—where the ridge sweeps 0.30 to 0.47 across its regularization path and its cross-validated choice costs up to 2.4 points against the best value in hindsight. The margin in accuracy is of the same order as that tuning uncertainty; what is not of that order is the advantage of the representation, and it is where most of the gain comes from.

Two readouts widen the picture, and the second changes the reading at full budget. Nyström KRR, the global counterpart of the sparse construction, trails the kernel readout by 1 to 3 points at $N_x = 50\,000$ and coincides with it at $N_x = 1000$, where $m = N_x$ leaves nothing to approximate. More interesting is the composite "trend + residual", a cross-validated ridge used as a global trend with a kernel readout on its residual. At $N_x = 1000$ it is indistinguishable from the kernel readout alone in three cases of four—the kernel *subsumes* the linear part, which is a stronger statement than the paper's own attribution—but at $N_x = 50\,000$ it is ahead in all four, by 1.1 and 1.2 points at depth three on the transformer and the convolutional network respectively, where it also overtakes the model's own head (0.923 against 0.922).

That reversal is a truncation effect rather than a property of linearity, and a bandwidth sweep establishes it. The readout is dense at $N_x = 1000$ and sparse at full budget, so at $M = 100$ it sees two centers in a thousand per query while the trend is fitted on all $50\,000$: the composite recovers non-local information that the truncation discards, and the gap should close as M grows. Table 8 shows that it does, monotonically on both architectures between $M = 100$ and $M = 500$, where the readout alone catches up. Nothing linear was missing: what was missing was reach.

Pushed the other way, the sweep says something stronger than the diagnosis it was run for. Over the whole range the readout alone moves by 0.039 on the transformer and 0.026 on the convolutional network, while the composite moves by 0.011 and 0.002—and the per-query solve is cubic in M, so that range spans a factor $125\,000$ in cost. A global trend, fitted once and amortized over the query stream, therefore makes the bandwidth nearly *immaterial*: on the convolutional network the composite holds within two thousandths from $M = 10$ to $M = 500$. This is another member of the family of Remark 1, trading a global object against a local one, and here the global one is by far the cheaper.

A last reading of Table 9, and it bounds what the construction buys. On the convolutional network at full budget and depth four, *every* readout lands within three thousandths of the others—head 0.922, ridge 0.922, kernel 0.923, Nyström 0.924, trend + residual 0.925—and a linear readout is among them. That is a ceiling, and the geometry says why: cross-entropy training has driven the representation into ten near-orthogonal clusters, at 99.8% mode purity and within-class distances at 0.03 of between-class ones, so the task is linearly separable there and no readout can do better than the representation allows. On that architecture, at that budget and that depth, the kernel readout buys *nothing* in accuracy; what it buys is the per-query error estimate, which none of the others carries. The transformer at the same budget is not at its ceiling—2.8 points separate the ridge from the composite at depth three—and that is where the readout still pays in accuracy as well.

Two observations on the sweep itself. The interpolation residual breaks by two orders of magnitude between $M = 25$ and $M = 50$, abruptly and on both architectures, from the scale of the nugget to some two hundred times it. Since that residual is of order nugget$/\lambda_{\min}$ of the local Gram, the smallest eigenvalue drops by a factor 100 when the neighbourhood grows past roughly thirty points—consistent with the representation collisions this budget produces, 36 strictly identical feature pairs on the convolutional network, which begin entering neighbourhoods at about that size. Below $M \approx 30$ the local solve is therefore the better conditioned, and the composite loses almost nothing there. And the composite's own curve saturates rather than keeps improving as M falls: on the convolutional network it is flat within two thousandths from $M = 10$ on, which one seed cannot resolve further.

Depth matters at small budget, and how the representation is read decides whether one sees it. At $N_x = 1000$ the shallower probe leads the deeper one by 2.3 points on the transformer and 3.0 on the convolutional network, while reading half as many features on the latter—so not on capacity. On the transformer the effect is visible only when the token sequence is *flattened*: averaging the tokens, which collapses each image to the embedding dimension, erases it. And it vanishes at full budget, where the two depths agree to 0.07 points. A probe reports on a representation together with the way one chooses to read it, and on how much data produced it.

Table 8: Bandwidth sweep at $N_x = 50\,000$, depth three, one seed. The trend is fitted once per architecture and reused unchanged; only the readout's bandwidth varies, on both sides of the comparison. Last column: interpolation residual at the centers, whose two-order break between $M = 25$ and $M = 50$ is discussed in the text.

	M	readout alone	trend + residual	interp. residual
transformer	10	0.776	0.803	$3 \cdot 10^{-9}$
	25	0.787	0.806	$4 \cdot 10^{-9}$
	50	0.794	0.810	$2 \cdot 10^{-7}$
	100	0.802	**0.813**	$2 \cdot 10^{-7}$
	200	0.807	0.812	$2 \cdot 10^{-7}$
	500	0.814	0.814	$2 \cdot 10^{-7}$
conv. net	10	0.897	0.922	$2 \cdot 10^{-9}$
	25	0.899	**0.923**	$2 \cdot 10^{-9}$
	50	0.901	0.923	$2 \cdot 10^{-7}$
	100	0.911	0.923	$2 \cdot 10^{-7}$
	200	0.917	0.923	$2 \cdot 10^{-7}$
	500	0.923	0.923	$2 \cdot 10^{-7}$

The indicator sharpens as the backbone trains, its AUROC rising from 0.69 to 0.86 on the convolutional network, so it is most informative exactly where the readout would be used.

One boundary, because Appendix A.8 crosses it: a probe characterizes a representation *as produced*, not the potential of an architecture—the blocks it reads were trained to feed the model's own head, with its own loss. Training them for the kernel readout instead does not reproduce these numbers, and on the transformer it degrades them.

Table 9: Attributing the probing gain, at convergence and depth three (resp. four), against the model's own head. Four seeds at $N_x = 1000$; *one seed* at $N_x = 50\,000$, no other checkpoint existing at that budget. The ridge fits $d \times 10 + 10$ coefficients and a regularization chosen by cross-validation; the kernel readout and Nyström fit *none*. "Trend + residual" is the cross-validated ridge as a global trend plus a kernel readout on its residual—polynomial reproduction in the RBF literature, universal kriging in geostatistics. The readout is dense at $N_x = 1000$ and sparse at $M = 100$ at full budget, a $50\,000$-center dense solve being infeasible; Nyström uses $m = \min(2000, N_x)$, so at $N_x = 1000$ it coincides with the dense readout by construction.

		head	ridge	kernel	Nyström	trend + residual
$N_x = 1000$	transformer, d3	0.381	0.456	**0.472**	0.472	0.472
	transformer, d4	0.381	0.433	0.445	0.445	**0.448**
	conv. net, d3	0.563	0.585	**0.590**	0.590	0.589
	conv. net, d4	0.563	0.570	0.569	0.569	0.569
$N_x = 50\,000$	transformer, d3	0.784	0.785	0.802	0.789	**0.813**
	transformer, d4	0.784	0.795	0.800	0.799	**0.801**
	conv. net, d3	0.922	0.905	0.911	0.907	**0.923**
	conv. net, d4	0.922	0.922	0.923	0.924	**0.925**

Probing: kernel, regularization and map. The readout above is Matérn-$1/2$ at nugget 10^{-9} with the cosine map, dense at $N_x = 1000$ and sparse at bandwidth $M = 100$ at $N_x = 50\,000$. This section reports the measurements behind that choice, on the transformer's representation at depth four, one seed, at the first and last training epoch.

Probe against head, in detail. At $N_x = 1000$ and convergence the deeper probe is ahead of the head by 1.4 points on the convolutional network and 1.5 on the transformer; at $N_x = 50\,000$, by 0.2 and 1.0. The min–max bands overlap—on the transformer the head's band encloses both probe curves—so we read the comparison *paired by seed*: the probe is ahead in 82 of the 84 paired comparisons for the transformer and in 84 of 84 for the convolutional network, over the last twenty

epochs. Early in training the margin is far larger—13.7 points at epoch 1 at full budget, where the head scores 0.555 against 0.691 for the readout—and the mechanism bounds the claim: after a single epoch the head has extracted little of what the convolutional stack already carries, and a kernel readout reads that geometry without fitting anything. What is left at convergence is the part the head never extracts.

That the three readings converge to nearly the same value at full budget is the expected outcome, and a reassuring one: a readout durably beating a head trained end-to-end for the same objective would indicate an undertrained head rather than a better readout. What the probe contributes is a reading available *throughout*, at no fitting cost, and read off a curve markedly steadier than the head's: over the fifty epochs the head's worst single-epoch regression is 2.6 points against 0.55 for the probes, and its epoch-to-epoch variation is twice as dispersed.

Depth, in detail. At $N_x = 1000$ the shallower probe is ahead of the deeper one on the convolutional network at every seed and every epoch—2.2 points on average over the last twenty, ahead in 84 of 84 paired comparisons—and the confound runs the wrong way to explain it, the winning probe reading 4096 features against 8192. At full budget that ordering *inverts*, and at a datable moment: the shallower probe leads until epoch 13, crosses, and ends 1.3 points below the deeper one and 1.1 below the head. On the transformer, whose probed dimension is constant at 128 under token averaging, the comparison is *undecided* at both budgets: the shallower probe leads by 0.3 points in 49 of 84 paired comparisons at $N_x = 1000$, and by 0.02 points in 6 of the last 10 epochs at full budget. We therefore report the depth effect as a property of the convolutional architecture that the transformer does not reproduce.

The indicator, in detail. Its AUROC improves monotonically with training: on the convolutional network at depth four it rises from 0.69 at epoch 1 to 0.86 at epoch 50, and on the transformer from 0.63 to 0.77. The deeper probe carries the better indicator throughout, by three to nine points of AUROC over the shallower one—the reverse of the accuracy ordering on the convolutional network at small budget, a dissociation we record without explaining it.

Why the readout buys more on the transformer. The gain over a linear readout on the same features is three times larger on the transformer than on the convolutional network—1.6 points against 0.5 in Table 9—and the mode structure of the two representations accounts for it. At full budget the convolutional features form ten near-orthogonal clusters: within-class distances at 0.03 of between-class ones, 99.8% mode purity, 91.8% recall. The task is then close to linearly separable, and there is little left for a local solve to exploit—widening the neighbourhood from $M = 100$ to $M = 500$ buys 0.04 points. The transformer keeps a continuum, 83% purity and 37% recall, and the same widening buys 0.55 points. The same asymmetry appears in Appendix A.8.1, where a geometric objective works on the transformer and not on the convolutional network, and it has one cause: cross-entropy training saturates the convolutional geometry and does not saturate the transformer's.

The kernel, not the regularization, is what matters here. The Gaussian kernel on these representations produces a Gram matrix conditioned at $1.4 \cdot 10^9$ with coefficients of norm $3.4 \cdot 10^6$—for targets that are one-hot—against $2.2 \cdot 10^4$ and $3.1 \cdot 10^2$ for Matérn on the same points. The solve is exact in both cases, with relative residual below 10^{-8} and interpolation at the centers at 1.0000: the Gaussian predictor passes through every training point and oscillates between them, a Runge phenomenon rather than a failure of the linear algebra. The consequence is a gap of 5.7 points at epoch 1 and 7.4 at epoch 50 in favour of Matérn.

Raising the nugget does not close it. Across 10^{-9}, 10^{-5} and 10^{-3} the Matérn accuracies agree to 0.001 and the interpolation stays at 1.0000, so the nugget is doing what Section 2.1 says it does—numerical hygiene. For the Gaussian it becomes a genuine hyperparameter, worth up to 9 points, and it buys those points by abandoning interpolation: at 10^{-3} the accuracy at the centers falls to 0.902. Its best configuration still does not reach Matérn at 10^{-9}. We therefore read the Gaussian out not on its accuracy—that would be tuning on the evaluation set—but on the observation that no setting of its regularization keeps the interpolation regime this paper works in.

Locality regularizes. At $N_x = 1000$ the two regimes can be compared directly, and the comparison separates the effect of the kernel from the effect of the construction (Table 10):

Table 10: Test accuracy at $N_x = 1000$, transformer depth four, nugget 10^{-9}, cosine map. The sparse readout uses $M = 100$, so each query sees a tenth of the available centers; the dense one solves on all of them. These figures predate the protocol used above—tokens averaged rather than flattened, embedding dimension 64—so their absolute level is not comparable to the numbers reported there. They are used here to compare *kernels and regimes* on a fixed representation, which is what the surrounding paragraphs establish, and that comparison does not depend on the aggregation.

		Gaussian	Matérn
epoch 1	dense	0.2043	0.2612
	sparse	0.2268	0.2515
epoch 50	dense	0.2932	0.3676
	sparse	0.3138	0.3607

With Matérn the ordering is the expected one: the dense solve, which sees ten times more centers, is ahead by 0.97 then 0.69 points. With the Gaussian it *inverts* —the sparse readout wins by 2.25 then 2.06 points—because a local system over 100 neighbours stays well conditioned where the global one over 1000 does not. The locality of the sparse construction thus acts as an implicit regularizer, and it is exactly when the global solve degenerates that this matters. The margin in favour of the dense solve staying under one point is the other half of the observation: at this budget the sparse readout recovers nearly all of what the global one achieves, at $\mathcal{O}(M^3)$ per query instead of $\mathcal{O}(N_x^3)$ once.

The input map is immaterial. Three maps were compared, a plain cosine normalization, the same followed by the median rescaling, and a per-sample standardization followed by that rescaling. On the transformer the last two are *bit-identical*: its encoder ends in a per-token layer normalization, token averaging preserves the resulting zero channel mean, and a zero-mean vector satisfies $\mathrm{LayerNorm}(x) = \sqrt{d}\, x/\|x\|$, the constant $\sqrt{d}$ then cancelling in the median rescaling. On the convolutional network, whose blocks end in a rectifier and whose activations are therefore not centered, the two maps are genuinely different—the resolved lengthscale differs by a factor 112—and yet the accuracies differ by 0.0002. A hundredfold change in lengthscale is absorbed because the kernel divides by it and the shape of the distance distribution is unchanged. We use the cosine normalization followed by the median rescaling at both budgets and for both architectures.

Conditioning improves as the backbone trains. One might expect the opposite: training separates classes, and a point cloud collapsing onto a few directions degenerates its Gram matrix—a synthetic model with ten well-separated clusters reproduces exactly that, the condition number climbing from $4.6 \cdot 10^3$ to $6.2 \cdot 10^8$ as separation grows. The measurement on real representations goes the other way: between epoch 1 and epoch 50 the condition number *falls*, by a factor 1.6 for Matérn and 33 for the Gaussian. Training evidently enriches the within-class diversity of the representation faster than it collapses the between-class structure. We report this because it matters for reading Figure 2: a numerical artefact cannot produce the apparent decline of a probe curve over training, since the systems being solved get better conditioned, not worse.

A.8 Learning details

Appendix A.7 read representations from outside the network. Here the kernel layer enters the training loop, replacing the head. Two objectives are reported: a purely geometric one, which is the experiment cited in Section 3.3, and the lazy readout of instance (i) trained through, which is reported here for completeness.

A.8.1 A geometric objective

Setting. Architectures, data and optimizer follow Appendix A.7: AdamW, learning rate 10^{-3}, weight decay $5 \cdot 10^{-4}$, batch size 100, random crops and horizontal flips, representations flattened, and the same frozen subsample keys. Two differences. Training runs for 150 epochs with

`CosineAnnealingLR`($T_{\max} = 150$) at $N_x = 1000$, because the objective converges more slowly than cross-entropy—at 50 epochs the convolutional network is still improving—and for 50 epochs at full budget, where an epoch carries fifty times more gradient steps. And no cross-entropy term appears at any point: the readout is attached at depth 3 or 4 and the backbone is trained from initialization by

$$\mathcal{L}(\theta) \;=\; \sum_{l=1}^{L-1} \mathrm{GM}\big(\overline{y_\theta^{l}},\, \overline{y_\theta^{l+1}}\big) + \mathrm{GM}\big(\overline{y_\theta^{L-1}},\, f(x)\big), \qquad \mathrm{GM}(Y, Y') = \frac{1}{b^2}\sum_{i,j}\big(D(Y)_{ij} - D(Y')_{ij}\big)^2, \tag{17}$$

where $D(Y)_{ij} = \|Y_i - Y_j\|_2$ and $\overline{\cdot}$ is the cosine map of Appendix A.5. The sum starts at $l = 1$, the patch embedding carrying no weights.

Why the readout's own map, applied first. Without it the objective is dominated by a scale mismatch and not by class structure. Flattened transformer features have norm $\sqrt{64 \times 128} \approx 90$ after layer normalization, so their pairwise distances are of order 128, whereas one-hot targets are $\sqrt{2}$ apart—a ratio of 91. Minimizing the last term of a raw (17) therefore asks first for a contraction by that factor, and the network complies: the lengthscale falls from 1.382 to 0.029 within one epoch and the representation is destroyed. After the cosine map all distances lie in $[0, 2]$, the terms are commensurable by construction, and contraction is no longer available as an answer: the lengthscale then *rises*, from 0.965 to 1.25 on the convolutional network.

Results. Four seeds, mean and range over the last five measurements, against each backbone's own linear head trained at the same protocol.

Table 11: Geometric objective at $N_x = 1000$, 150 epochs, four seeds. The paired column gives the number of seeds on which the objective beats the backbone's own head; the inter-seed range of the values themselves is 3.2 to 4.1 points, so the paired sign is what decides.

	own head	depth 4	depth 3	paired (depth 4)
transformer	0.3991	**0.4541** (+5.50)	**0.4588** (+5.97)	**4/4**
convolutional net	0.5994	0.5749 (−2.45)	0.5690 (−3.04)	0/4

On the transformer the objective beats cross-entropy by 5.5 to 6.0 points with the same sign on all four seeds, and it also exceeds the probe on a cross-entropy backbone read at depth 4 (0.4230 at this protocol). On the convolutional network it loses 2.5 to 3.0 points, again with the same sign on all four seeds. The split is architectural, not an artefact of duration: both were trained for 150 epochs, and the same measurement at 50 epochs would have flattered the transformer, whose reference *drops* 2.6 points between the two durations while the convolutional one *gains* 4.2.

At full budget, one seed at depth 4, the objective reaches 0.7486 on the transformer against 0.7787 for its head, and 0.9180 against 0.9234 on the convolutional network—within half a point of parity there, and behind on the transformer. The advantage at $N_x = 1000$ is therefore a small-budget advantage.

The geometry reached is an attractor. The kernel contrast—the ratio of the mean kernel value on same-class pairs to that on different-class pairs—is the quantity the solve exploits, and it converges to the same value at both budgets: 2.037 at $N_x = 1000$ after 150 epochs, range 0.035 over four seeds, against 2.002 at $N_x = 50\,000$ after 50. The same geometry with fifty times fewer images, the difference being the number of gradient steps needed to reach it. What the contrast does *not* do is predict accuracy: the convolutional network attains the highest contrast of the study and loses to its head, the transformer the lowest and wins.

An ablation at one seed did not separate the label term of (17) from the full objective, while the objective deprived of that term degrades below random initialization; we keep the form above.

A.8.2 Training through the lazy readout

Setting. Identical to Appendix A.8.1 except for the objective and the duration: cross-entropy on the readout, 50 epochs with $T_{\max} = 50$. Instance (i) is attached at depth 3 or 4—centers the current features of the minibatch, detached; coefficients the observed labels—so the head carries *no*

parameter of its own, and whatever gain follows is attributable to the geometry of the readout rather than to added capacity.

The objective is degenerate, and the scheme is an alternation. By Remark 3 the loss is constant in θ; we measure it at 1.50615, which is the cross-entropy of one-hot logits under our label smoothing, and its total derivative vanishes to 10^{-9}, the branch through the queries and the branch through the centers cancelling. Detaching the centers keeps the first branch only, and that is what *creates* a descent direction. The procedure is therefore an alternation—descend at frozen centers, refresh the centers, repeat—closer to a target network than to a minimization of $\mathcal{L}$, and the refresh rate is its step.

Results. Four seeds at $N_x = 1000$, one at full budget, against the same heads.

Table 12: Lazy readout trained through, cross-entropy, 50 epochs. Paired columns count the seeds on which the readout beats the head.

	own head	depth 4	depth 3	paired (d4)	paired (d3)
transformer	0.3811	**0.4278** (+4.67)	**0.4228** (+4.17)	**4/4**	**4/4**
convolutional net	0.5659	0.5752 (+0.93)	0.5690 (+0.31)	3/4	2/4

The pattern is the same as for the geometric objective and weaker: clear on the transformer, within seed noise on the convolutional network, where a range of 2.0 to 3.2 points dwarfs a 0.31-point effect. At full budget both lose—the transformer by 5.3 points at depth 4, the convolutional network by 0.65—so the gain is again a small-budget gain.

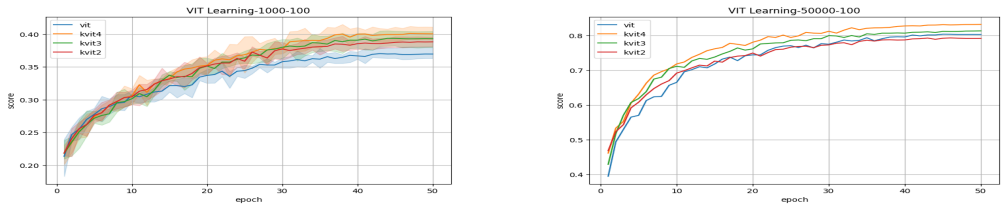
Reading at depth 3 leaves the fourth block and the head unused, so the trained model is three blocks and a parameter-free readout: $1{,}147{,}200$ parameters against $4{,}957{,}002$ for the full convolutional network, or 23%. That buys accuracy *comparable* to the full network rather than better—$+0.31$ point on two seeds of four—which is a compression statement and not an accuracy one. On the transformer, whose blocks are of equal size, the same reading keeps 75% of the parameters.

What we tried and did not get to work. Starting instead from a converged cross-entropy checkpoint, whose representation a probe reads at 0.477 on the transformer, training through the readout *degrades* it to 0.436. We traced this across three objectives reaching the same attractor: the constant loss above; an asymmetric layer (9) with distinct projections, whose loss is genuinely non-constant but whose optimizer walks back to the symmetric point where interpolation is exact and there is again nothing to minimize; and a leave-batch-out objective, predicting each minibatch from the remaining centers, which is sound, demonstrably optimized—its held-out interpolation accuracy improves by 14 points—and yet transfers nothing, losing 4.2 points on test. Replacing cross-entropy by a squared loss, better matched to a readout whose output lies in the simplex rather than in logit space, recovers 1.3 of those points but does not change the sign. Appendix A.16 records these diagnostics.

A.8.3 A KERNEL MODULE IN PARALLEL WITH THE FEED-FORWARD PATH

The experiments above place the kernel readout *at* a chosen depth. A complementary construction places one *inside* every encoder block of the Vision Transformer: each block keeps its parametric position-wise feed-forward sub-block and a kernel module is added to that path, the classification head being a kernel readout as well—the integration regime of the Double DQN agent of Section 3.4, related to persistent-memory augmentation (Sukhbaatar et al., 2019; Lample et al., 2019). Figure 6 reports test accuracy: at full depth the improvement over the canonical baseline reaches $+3$ points at $N_x = 1000$ and $+2.1$ points at $N_x = 50\,000$.

Three remarks delimit the claim, and the third is a scope restriction on our own framework. The comparison is *not* at parity—the model carries strictly more capacity than the baseline—so this measures what the added kernel capacity contributes alongside the parametric sub-block, not that the parametric components are suboptimal. Outright *substitution* of the feed-forward sub-block is not supported: replacing it with either readout degrades accuracy (Appendix A.16), and substituting the attention sub-blocks as well yields no further gain, consistent with their kernel structure (Tsai et al., 2019; Ramsauer et al., 2021). The model thus remains a hybrid.



(a) $N_x = 1000$, five runs (average and min–max) (b) $N_x = 50\,000$, one run

Figure 6: Test accuracy on CIFAR-10 for the Vision Transformer carrying a kernel module in parallel with the feed-forward path of each encoder block, at decreasing depth (topmost blocks removed). The improvement over the canonical baseline reaches $+3$ points at $N_x = 1000$ and $+2.1$ points at full budget. The comparison is not at parity of capacity, and this module normalizes each kernel row by a softmax, so it is a normalized smoother rather than an element of $\mathcal{H}_k$; see the discussion below. These runs use the C++ implementation of the kernel library.

Finally, this particular module normalizes each kernel row before combining the coefficients. Its kernel $k(a, b) = \big(2 - \langle \bar{a}, \bar{b} \rangle\big)^{-1}$ on normalized arguments is strictly positive definite—it is the series $\frac{1}{2} \sum_{n \geq 0} (\langle \bar{a}, \bar{b} \rangle / 2)^n$ and satisfies $k \in [\frac{1}{3}, 1]$—and so is $\exp(k/\tau)$. What leaves $\mathcal{H}_k$ is not the exponential, which composes with a readout harmlessly, but the *row-wise division* that follows it: it breaks the symmetry of the pairing, so the module is a normalized smoother and the certificate of Appendix A.11 does not extend to it as stated. Its output is the ratio of two elements of $\mathcal{H}_{\exp(k/\tau)}$ whose denominator is bounded below by $\frac{1}{3}$, so a certificate is derivable in principle; we have not derived it. These runs also use the C++ implementation of the kernel library rather than the PyTorch port used elsewhere in this paper.

A.9 Reinforcement learning details

Setting. We use the LunarLander-v3 environment from the Gymnasium library[10] for both agents, with state dimension $S = 8$ and action space size $A = 4$. The two agents have their own settings. *Double DQN*: discount factor $\gamma = 0.99$, smooth L^1 loss, hidden dimension $L = 64$, $B = 64$ kernel centers per module, AdamW at batch size 64 and learning rate 10^{-3}, no scheduler, 500 episodes. *PPO*: $\gamma = 0.999$, GAE at $\lambda = 0.95$, clipping 0.2, eight epochs per update, update interval 1024 steps aligned on episode ends, minibatch 32, entropy coefficient $1.21 \cdot 10^{-3}$, value coefficient 0.333, gradient clipping 0.5; the learning rate is tuned per model and reported with each figure. Results are averaged over five seeds. Figure 7 reports the training losses of both agents.

This experiment illustrates how the kernel-differentiable machinery can augment an existing reinforcement learning algorithm—namely, Double DQN (van Hasselt et al., 2016)—on the LunarLander environment.

Figure 4 reports the reward and loss curves, averaged over 50 games and across five independent runs of 500 episodes, with shaded min–max ranges. Over the last fifty episodes the kernel-augmented agent reaches a mean reward of 246 against 174 for the baseline, and it does so more reliably: its across-seed range is 218–257 where the baseline's is 27–244.

Two ablations locate the gain. A parameter-matched MLP baseline—the same architecture with the kernel modules replaced by feed-forward blocks of equal parameter count—reaches 206, so the ordering is $246 > 206 > 174$: added capacity accounts for part of the gain and the kernel construction for the rest. And each module helps on its own, 225 for the first layer and 217 for the last, the two together being better than either. Freezing the centers at their initialization still reaches 231, so their adaptive placement contributes but is not critical—the lazy-versus-learnable dial of Section 2.3.

Freezing the interpolation *targets* instead is not a well-posed ablation, and the reason is worth stating: unlike the supervised settings of Sections 3.1 to 3.3, where θ_y is given by the observed labels, here it holds the value-function coefficients under optimization. Freezing them at zero makes the module

[10] Maintained by the Farama Foundation: `https://gymnasium.farama.org/`.



(a) Training losses (DQN). (b) Training losses (PPO).

Figure 7: Training losses for the two agents of Section 3.4. The kernel variants show markedly flatter loss curves; note that a lower PPO loss does not imply a better policy—it measures how little the policy changes between updates, and a hovering policy yields predictable returns.

the zero map—byte-identical to the baseline—and freezing them elsewhere imposes an arbitrary constraint. Instance (i) is therefore unavailable in this setting: laziness requires observed targets.

The previous experiment *augments* a neural agent with a kernel module; we now consider the complementary regime, in which the kernel module instead *replaces* the parametric head. We use Proximal Policy Optimization (PPO) (Schulman et al., 2017), a deep reinforcement learning algorithm built on an actor–critic model whose two heads are usually parameterized by MLPs. We compare the original PPO to a variant in which each head is the deep kernel readout (9). Three measurements delimit what the substitution achieves. At embedding width $E = 128$ the readout reaches the parametric agent's reward at $0.28\times$ its parameters—the parametric heads carry a hidden layer costing $\mathcal{O}(E^2)$ where the readout costs $\mathcal{O}(E)$, so the ratio improves with width and is structural rather than incidental. It gets there more slowly, trailing by 26 to 89 points between 100k and 300k environment steps before the curves meet by 350k: the difference is a rate of learning, not a ceiling. And its training cost is of the same order but not equal, $2.15\times$ the parametric baseline at matched hyperparameters.

Two further observations, both of which the reward alone hides. The kernel-headed agent halves the crash rate—21.7% against 42.6% at $E = 128$—and it holds a loss four to five times steadier across widths. That second figure must not be read as a better policy: the PPO objective measures how little the policy changes between updates, and this agent spends a third of its episodes hovering, which yields predictable returns for the critic and no landing. We therefore report the decomposition by outcome—landing rate, crash rate, mean cost of a crash, hovering rate—alongside the reward, and count episodes in *environment steps* rather than in episodes, an episode costing 120 steps when the policy crashes and 1000 when it hovers. Being a deep kernel model, this variant inherits the error estimates of Appendix A.11.

Double DQN (Section 3.4). Double DQN is an online, off-policy, temporal-difference algorithm that estimates the action-value function. Let $(s_t, a_t, r_t, s_{t+1}) \in \mathcal{D}$ be transitions sampled from a replay buffer $\mathcal{D}$, where $s_t \in \mathbb{R}^S$ is the state, $a_t \in \mathcal{A}$ the discrete action, with $|\mathcal{A}| = A$ the number of possible choices, and $r_t \in \mathbb{R}$ the reward. The training objective is

$$\inf_{\theta} \; \mathbb{E}_{\mathcal{D}}\Big[\mathcal{L}\big(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta)\big)\Big], \tag{18}$$

where γ is the discount factor, θ the parameters of the online network, θ^- those of the target network, and $\mathcal{L}$ a loss function. The original DQN model uses

$$Q(s, a; \theta) = \mathrm{ReLU}\big(\mathrm{ReLU}(s\,\theta_1)\,\theta_2\big)\,\theta_3, \quad \theta_1 \in \mathbb{R}^{S\times L},\ \ \theta_2 \in \mathbb{R}^{L\times L},\ \ \theta_3 \in \mathbb{R}^{L\times A}, \tag{19}$$

where L is the latent dimension. We refine this model by adding a kernel perturbation to the first and final layer:

$$s_1 = \mathrm{ReLU}\big(\big[\mathrm{ReLU}(s\,\theta_1) + y_{k_1,\theta_{x,1},\theta_{y,1}}(s)\big]\,\theta_2\big), \qquad Q_k(s, a; \theta) = s_1\,\theta_3 + y_{k_3,\theta_{x,3},\theta_{y,3}}(s_1), \tag{20}$$

where y_{k,θ_x,θ_y} is the readout (14) of instance (ii), adding the parameter sets $\theta_{x,1} \in \mathbb{R}^{B\times S}$, $\theta_{y,1} \in \mathbb{R}^{B\times L}$, $\theta_{x,3} \in \mathbb{R}^{B\times L}$, $\theta_{y,3} \in \mathbb{R}^{B\times A}$, where B is the number of learnable kernel centers per module—i.e., the size of the support set $(\theta_{x,i}, \theta_{y,i})$ on which the readout $y_{k_i,\theta_{x,i},\theta_{y,i}}$ is built. The model is

trained with the same algorithm and the same loss (18), treating the additional kernel parameters $\theta_{x,i}, \theta_{y,i}$, $i = 1, 3$, as learnable parameters.

Consistently with (14), these are *free* readout coefficients, not observed targets: the $\theta_{y,i}$ are initialized to zero, so that the kernel modules are switched off exactly at initialization and the augmented agent starts from the baseline.

Table 13: Hyperparameters across experiments. The learning column covers both objectives of Appendix A.8; its bandwidth is *dense on the minibatch*, the centers being the batch's own features, so M equals the batch size by construction. The two agents differ enough that the RL column is only indicative; Appendix A.9 states each in full.

	Transfer	Probing	Learning	RL
Optimizer	AdamW	AdamW	AdamW	AdamW
Learning rate	10^{-3}	10^{-3}	10^{-3}	10^{-3}
Weight decay	—	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	—
Batch size	128	100	100	64 (DQN), 32 (PPO)
Bandwidth M	100	dense at N_x=1000, else 100	dense (batch)	—
Scheduler	—	cosine	cosine	—
Epochs	—	50	150 at N_x=1000, 50 at full budget	—

A.10 Out-of-distribution detection details

A.10.1 Out-of-distribution detection on a large language model.

This section addresses the evaluation-scale limitation stated in Section 4: it measures the power function on the embeddings of a real large language model and compares it, on identical data, against the published out-of-distribution (OOD) detection benchmark of Ren et al. (2023) for conditional language models. Two scoping remarks are in order before any number is reported. First, *what is measured here is the dense power function* (4), computed over 10^4 centers in the interpolation regime of Section 2.1—not the sparse M-NN construction of Section 2.2. The sparse variant is the scaling vehicle that inherits these estimates cell-wise through (7); validating it against the dense score, used as an oracle, is deferred to future work. Second, the background-contrastive score ϵ_{rel} introduced below is *not* part of the main text: contrasting an in-domain score against a generic background distribution is a protocol adaptation borrowed from the Relative Mahalanobis Distance (RMD) of Ren et al. (2021), extended to conditional language models in Ren et al. (2023). Our contribution is to show that the power function, transferred unchanged to LLM embeddings and placed in the same contrastive scheme, exceeds the strongest comparable score of that benchmark. The machine-precision identity with the Gaussian-process posterior variance underlying SNGP-style estimation (Liu et al., 2020) is what makes the comparison a controlled one: placed in a single family, the differences we measure separate into the kernel, the random-feature approximation, and the input map—the last being the largest of the three.

Protocol. We replicate exactly the *input-score* protocol of Ren et al. (2023). The model is $\mathrm{PEGASUS}_{\mathrm{LARGE}}$ (Zhang et al., 2020) fine-tuned on the xsum dataset (Narayan et al., 2018) (`google/pegasus-xsum`), the same model as their Section 3.1. The embedding $z \in \mathbb{R}^{1024}$ of a document is the masked average of the encoder's final-layer hidden states, their exact recipe (Section 2.2 of Ren et al. (2023)). In-domain centers are 10^4 documents sampled (fixed seed) from the xsum training split; the background set is 10^4 documents from C4 (Raffel et al., 2020). Neither set is ever scored. Negatives are the full xsum test split (11,333 documents); positives are the test splits of cnn_dailymail (Hermann et al., 2015) (11,490, near-OOD), reddit_tifu (Kim et al., 2018) (5,000, far-OOD), and samsum (Gliwa et al., 2019) (819, far-OOD). We re-implement three baselines on these embeddings: the Mahalanobis distance (MD) to the in-domain Gaussian, $\mathrm{RMD} = \mathrm{MD}_{\mathrm{in}} - \mathrm{MD}_{\mathrm{bg}}$ (the strongest input score of Ren et al. (2023)), and the KNN score of Sun et al. (2022) ($k = 1000$, ℓ^2-normalized embeddings, same 10^4 centers). All non-baseline scores follow one contrastive template: for a nonnegative novelty score s built once on the in-domain centers and once, independently, on the C4 background (each with its own standardization map),

$$s_{\mathrm{rel}}(z) \;=\; s_{\mathrm{in}}(z) \;-\; s_{\mathrm{bg}}(z). \tag{21}$$

The template is instantiated with: (i) the dense power function (4) over the 10^4 centers with a Gaussian kernel $k(x, y) = \exp(-|S(x) - S(y)|_2^2/(2\ell^2))$, under two input maps S: a *diagonal* affine (per-dimension) standardization, written ϵ_{rel}^d, and the *whitening* $S(x) = \Sigma_{\text{in}}^{-1/2}(x - \mu_{\text{in}})$ by the in-domain Gaussian itself, written ϵ_{rel}^w—the same $(\mu_{\text{in}}, \Sigma_{\text{in}})$ that parameterizes RMD, so that the comparison is matched on geometry and not only on data;[11] (ii) the same power function with the Matérn $\nu{=}1/2$ (exponential) kernel family of Section A.5, $k(x, y) = \exp(-|S(x) - S(y)|_2/\ell)$, used as a robustness comparator with its lengthscale swept and reported at its best (see the kernel-family paragraph below); (iii) the *exact* dense posterior variance of a Gaussian-process regressor with the same Gaussian kernel (Rasmussen & Williams, 2006), which coincides with (i) at machine precision (see below) and is reported as a single table row; and (iv) $\hat{\sigma}_{\text{rel}}^2$, its SNGP-style approximation (Liu et al., 2020), computed post hoc on the frozen embeddings with $D = 1024$ random Fourier features (Rahimi & Recht, 2007) and SNGP default regularization. All kernel systems are solved through the *direct* regularization $(K + tI)^{-1}$ with reference nugget $t = 10^{-10}$, the interpolation regime of the paper; the results below are invariant over $t \in [10^{-10}, 10^{-3}]$. The metric is the AUROC (in-domain negatives against each OOD positive set), with 95% bootstrap confidence intervals (1,000 resamples). Comparisons with Ren et al. (2023) are made on input scores and at the AUROC level only; their output scores are computed on decoded summaries and are not comparable to input-side quantities.

Table 14: AUROCs for input-side OOD detection with $\text{PEGASUS}_{\text{LARGE}}$–xsum embeddings (negatives: xsum test). "Target" denotes the published value of Ren et al. (2023) (Tables 1 and A.2). Brackets: 95% bootstrap confidence intervals. All kernel scores use the direct regularization $(K + tI)^{-1}$; the Gaussian ϵ_{rel} row coincides with the exact GP posterior variance at machine precision, and its value is reported as the (unchanged) range over $t \in [10^{-10}, 10^{-3}]$. (†): the KNN gap is a documented protocol divergence, not a defect of the score; see text.

	Near-OOD	Far-OOD	
Score (ours / target)	cnn_dailymail	reddit_tifu	samsum
MD	0.687 / 0.651	0.986 / 0.974	0.997 / 0.995
RMD	0.818 / 0.828	0.999 / 0.998	1.000 / 0.999
KNN (k=1000)	0.460 / 0.887 (†)	0.901 / 0.944	0.977 / 0.955
$\hat{\sigma}_{\text{rel}}^2$ (SNGP–RFF, D=1024)	0.849 [0.844, 0.853]	0.999	1.000
ϵ_{rel}^d (Gaussian, diagonal) = exact GP variance	0.892 [0.888, 0.896]	1.000	1.000
ϵ_{rel}^w (Gaussian, whitened)	**0.941** [0.938, 0.945]	1.000	1.000
ϵ_{rel} (Matérn $\nu{=}1/2$, steel-manned: best ℓ)	0.887 [0.883, 0.891]	1.000	1.000
ϵ_{in} alone (Gaussian)	0.647	0.965	0.996
ϵ_{bg} alone (Gaussian)	0.220	0.143	0.730

The evaluation pipeline replicates the published benchmark. The authority of Table 14 rests on replication: our RMD re-implementation reproduces the published near-OOD figure to within -0.010 (0.818 vs. 0.828), inside the ± 0.01 band we set in advance, and matches the far-OOD figures to the third decimal. The residual MD gaps (at most 0.036) are attributable to the sampling of the 10^4 centers and to C4 preprocessing details, not to the model—the RMD replication, which uses the same Gaussians, rules the model out. The comparisons below are therefore made *within* a validated pipeline, on strictly identical embeddings, centers, and test sets.

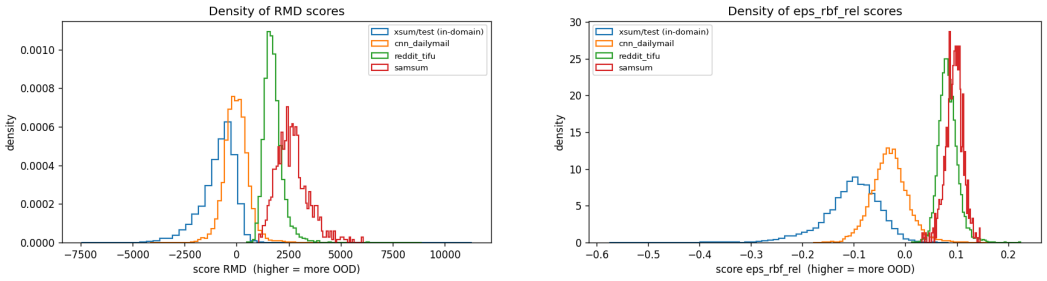
The power function *is* the GP posterior variance. This identity is what licenses the ingredient-by-ingredient comparison below: it places our score and SNGP-style ones in a single family, so that measured differences are attributable to the kernel, the approximation and the input map, and not to comparing unlike objects. For a Gaussian-process prior of precision τ, the exact posterior predictive variance satisfies $\text{var}(z) = \tau^{-1}\epsilon^2(z)$; with both quantities computed through the direct regularization $(K + tI)^{-1}$, we verify it at machine precision on this benchmark: $\max_z |\tau \,\text{var}(z) - \epsilon^2(z)| = 5.1 \cdot 10^{-15}$, Spearman rank correlation 1.0. The Gaussian rows of Table 14 are therefore one object,

[11] Lengthscales are $\ell = 45.0$ (resp. 44.2) for the diagonal (resp. whitened) map, in each case the median pairwise distance between mapped in-domain centers—an in-domain-only heuristic: no OOD data enters either choice. The whitening is ridge-regularized at 10^{-6}; $\text{cond}(\Sigma_{\text{in}}) = 1.7 \cdot 10^8$.

reported once. We note in passing that this *dense, post-hoc* variance was never evaluated on the benchmark, only its random-feature approximation.

A note of numerical rigor is in order. An earlier version of this study solved the same systems through the normal-equations form $(K^\top K + tI)^{-1}K^\top$, which squares the conditioning of K: at small t it produces deviations of order 10^{-2} and run-to-run non-reproducibility, and the nugget sensitivity we then observed was an artefact of that solver, not a property of the power function. The least-squares form remains the appropriate default for *rectangular* systems, such as the sparse constructions of Section 2.2; the verdict here is "direct by default" for the square SPD case, not a retirement of the least-squares form.

ϵ_{rel} outperforms RMD on near-OOD—invariantly. On cnn_dailymail, ϵ^d_{rel} (resp. ϵ^w_{rel}) reaches an AUROC of 0.892 (resp. 0.941), a *paired* improvement of $+0.073$ (resp. $+0.123$) over RMD computed in the same pipeline (95% paired bootstrap CI $[+0.070, +0.076]$, resp. $[+0.119, +0.128]$; $1{,}000$ resamples, identical folds), with far-OOD at 1.000 under both. Matching RMD's own geometry thus costs nothing and gains $+0.050$ over the diagonal map, so the input map is a lever of the same order as the score itself. The figure is invariant to every choice we could vary: the nugget across seven orders of magnitude ($t \in [10^{-10}, 10^{-3}]$: 0.891–0.892 throughout), the scale on which the contrast (21) is taken (ϵ vs. ϵ^2: shifts ≤ 0.001), and parallelism (five thread configurations: bit-for-bit identical scores). No tuned choice remains: the lengthscale is fixed by an in-domain heuristic (footnote above), and every other convention leaves the number unchanged. Figure 8 shows the effect at a glance: under RMD, the near-OOD mass sits largely inside the in-domain distribution; under ϵ_{rel} it visibly shifts out ($0.818 \to 0.892$), while the near/far gradation is preserved and far-OOD separates fully under both.



(a) RMD

(b) ϵ_{rel} (Gaussian)

Figure 8: Input-score densities on the summarization datasets, RMD (left) vs. Gaussian ϵ_{rel} (right). The near-OOD cnn_dailymail mass shifts visibly out of the in-domain xsum distribution under ϵ_{rel} (AUROC $0.818 \to 0.892$), while both scores preserve the near/far gradation and separate far-OOD fully. N.B.: these are *input* scores and are not comparable to the *output*-score densities of Figure 3 in Ren et al. (2023).

What separates our score from an SNGP-style one, ingredient by ingredient. Read through the identity above, an SNGP-style uncertainty score is a *contrastive score built on the Schaback error of a Gaussian kernel*, with two further choices attached: a random-feature approximation of that error, and an input map. Naming the three ingredients separates what is inherited from what is contributed, and each can be priced on this benchmark.

The *contrastive scheme* we inherit, from Ren et al. (2023): in-domain score minus background score. The *kernel* we do not fix—the Schaback error is defined for any SPD kernel, so the score generalizes beyond the Gaussian, and the Matérn row of Table 14 is the same construction under a different one. The *approximation* we remove and then price: with $D = 1024$ random Fourier features (Rahimi & Recht, 2007) it costs 0.042 of AUROC on near-OOD ($\hat{\sigma}^2_{\text{rel}}$ at 0.849 against 0.891 exact), while far-OOD is insensitive, both above 0.999—benign exactly where every method succeeds, and costly exactly on the hard regime. And the *input map*, which turns out to be the largest of the three levers: whitening by the in-domain Gaussian rather than standardizing per dimension gains 0.049, more than the approximation costs. The map is not incidental to such a score; it is the choice one should report first.

These are three separable contributions on one object—kernel-agnostic, exact, and map-explicit—and they are why we do not present this as an SNGP variant. What we compute is the Schaback error of a stated kernel under a stated map, evaluated per query; the Gaussian, random-feature, diagonally-standardized instance is one point in that space, and not the best one measured here. At scale it also argues for approximations of the dense score whose error is controlled against it, as intended for the sparse construction of Section 2.2.

Robustness to the kernel family. The headline number is not an artifact of the Gaussian choice. To give the alternative its best shot, we *steel-manned* the Matérn $\nu{=}1/2$ kernel—the kernel family of Section A.5—sweeping its lengthscale over $\{0.5, 1, 2, 4\}\times$ the median heuristic and citing it at its best, an explicit, disclosed exception to the no-tuning rule, reserved for the comparator. It plateaus at 0.887 $[0.883, 0.891]$, and an extended sweep confirms the plateau ($\ell = 178$ and $\ell = 356$ both give 0.887), against 0.892 $[0.888, 0.896]$ for the untuned Gaussian: even steel-manned, Matérn $\nu{=}1/2$ does not exceed the untuned Gaussian—one more invariance of the result.[12] The comparison surfaced one methodological caution: the *diagonal affine* standardization map is not incidental. Substituting a non-affine map calibrated on the centers' range degrades the same Matérn kernel, under the same solver, from 0.887 to 0.800: OOD points, by definition, leave the calibration range, and calibrated non-affine maps extrapolate them poorly. Affine maps are the safe default for OOD scoring.

The decomposition is interpretable, per query. Unlike RMD, whose contrastive value is a single opaque number, the decomposition (21) is informative term by term. On the Gaussian kernel, ϵ_{in} alone struggles on cnn_dailymail (0.647; a documented length confound, cf. the KNN discussion below), while ϵ_{bg} alone reveals the reason: its AUROC of 0.220—far *below* chance—means that cnn_dailymail documents are markedly *closer* to the C4 background than the in-domain xsum documents themselves. Near-OOD data thus carries a two-dimensional signature: moderately far from the in-domain set *and* anomalously close to the generic background. The score ϵ_{rel} exploits this signature, and, because both coordinates are available for every query, makes it visible per-query—where RMD delivers the contrast as a black box. Figure 9 renders the four populations directly separable by eye, and its caption records a finding that this per-query view surfaced in passing: a verbatim train/test duplicate in the benchmark itself.

The KNN gap is a protocol divergence, not a bug. Our KNN figure on cnn_dailymail (0.460) departs from the published 0.887 because of a documented difference in index size, aggravated by a length confound; we report our number as measured rather than tuning k to recover theirs.[13]

Scope of "in-domain", and the choice of background. Throughout, *in-domain* is defined relative to the *fine-tuning* distribution: the xsum test split is in-domain in distribution while held out in instance (never seen during fine-tuning). Contamination of the *pre-training* corpora (HugeNews, C4) by test documents cannot be excluded, but it is symmetric across the compared sets and therefore does not favor any score. Using C4 as the background is an assumed modeling choice, identical to that of Ren et al. (2023), who report their method to be robust to it.

Cost. Fitting amounts to two dense $10^4 \times 10^4$ factorizations and runs in 24.5 s on the CPU machine of Section A.5; scoring takes about 107 s per test dataset. Per kernel, the contrastive score and its two components (ϵ_{rel}, ϵ_{in}, ϵ_{bg}) are byproducts of a single computation.

A.10.2 Validity and detection at a controlled false-positive rate.

The AUROCs above summarize a whole ROC curve; a deployed detector runs at a fixed alarm budget, and the two need not order scores alike. To measure the latter we wrap each score in a split-conformal

[12] Kernel parameterizations, for reproducibility: Gaussian $\exp(-h^2|\cdot|_2^2)$ with $h = 1/(\ell\sqrt{2})$; Matérn $\nu{=}1/2$ (exponential) $\exp(-h|\cdot|_2)$ with $h = 1/\ell$.

[13] Ren et al. (2023) index, in all likelihood, the full xsum training split ($\approx$ 204k documents, so $k/N \approx 0.5\%$ at $k = 1000$), whereas our index holds the same 10^4 centers as all other scores ($k/N = 10\%$): at that ratio the k-th-neighbor distance measures centrality within the index rather than local novelty. The effect is compounded by a length confound: 75% of cnn_dailymail documents exceed the 512-token truncation limit against 32% for xsum, and the masked average contracts long documents toward the embedding centroid. A sweep over k is monotone, from 0.630 at $k = 10$ down to 0.460 at $k = 1000$.

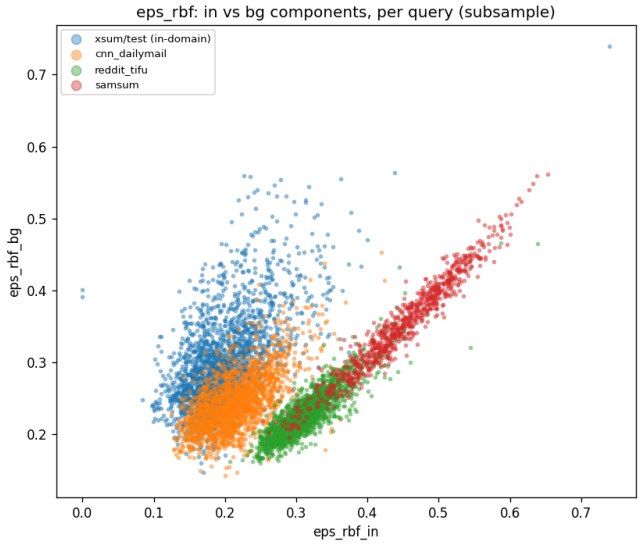


Figure 9: Per-query decomposition of the Gaussian contrastive score: ϵ_{in} vs. ϵ_{bg}, colored by dataset (subsample). Two OOD regimes, two geometries: near-OOD (cnn_dailymail) separates from in-domain *chiefly* through its anomalously low ϵ_{bg}—closer to the C4 background than xsum itself—while far-OOD (reddit_tifu, samsum) moves away from both references along the diagonal. This is the decomposition that RMD computes but does not expose. The isolated in-domain point at $\epsilon_{\text{in}} \approx 0$ (left edge) illustrates per-query interpretability: it is an xsum *test* document duplicated verbatim in the training split (a BBC photo-gallery caption, "A selection of photos from across the African continent this week:"; test stream index 2755, center index 1810)—a train/test leak of the benchmark, detected in passing because the power function vanishes exactly on the centers. Parametric scores such as MD or RMD have no notion of coincidence with a training point and are structurally unable to flag such cases.

scheme: the xsum test split is halved at random into calibration and test (20 disjoint draws, fixed seed), and the conformal p-value of a query is $p(z) = (1 + \#\{i : s_i \geq s(z)\})/(n_{\text{cal}} + 1)$ over the calibration scores s_i. No OOD data enters the calibration. Note that this wrapper cannot change an AUROC—the p-value is a monotone transform of the score—so what it adds is validity and an operating point, not discrimination.

Validity holds, and it holds for every score: measured false-positive rates on held-out in-domain data are 0.0092, 0.0493 and 0.0982 at nominal 0.01, 0.05 and 0.10, a maximal deviation of 0.0018 across draws. The Matérn variant of the score, whose detection is markedly weaker, validates to the same tolerance—validity is a property of the wrapper, not of the score it wraps.

The operating points, however, separate the scores, and more sharply than the AUROC does (Table 15). On near-OOD, $\epsilon^{d}_{\text{rel}}$ (resp. $\epsilon^{w}_{\text{rel}}$) is ahead of RMD by 2.5, 9.7 and 16.3 (resp. 10.7, 31.0 and 34.8) points of detection at the three levels; far-OOD saturates for all three scores. The margin therefore *grows* with the alarm budget, which is the opposite of what a single summary figure would suggest.

Two shape effects close the exercise, and neither is a defect of the map. First, the detection rates sit below what a Gaussian-separated score of the same AUROC would give: an AUROC of 0.941 predicts 0.456 detection at 1% and $\epsilon^{w}_{\text{rel}}$ delivers 0.253, where RMD sits at 0.146 against 0.149 predicted—exactly where the Gaussian equivalence puts it. The reason is structural: ϵ is bounded in $[0, 1]$ by construction, so ϵ_{rel} lies in $[-1, 1]$ and cannot have a Gaussian tail, whereas a difference of Mahalanobis distances is unbounded. Second, whitening lifts the whole curve without changing its shape—the ratio to the Gaussian equivalent is 0.55 against 0.61 for the diagonal map—so the gain is in separation, not in tail behavior.

The last column is the limit of the exercise, and it is a limit of conformal calibration rather than of the score. The guarantee is *marginal*: correct on average over the stream, and not within subpopulations. Partitioning the in-domain set into five unsupervised clusters, the measured false-positive rate spreads

Table 15: Fraction of OOD prompts flagged at nominal false-positive rate α, under the split-conformal wrapper calibrated on in-domain data alone. Mean over 20 disjoint calibration/test draws; inter-draw standard deviation at most 0.010. The rightmost column reports the spread of the *measured* false-positive rate across five unsupervised clusters of the in-domain set.

	cnn_dailymail (near)			far-OOD	conditional
α	$\epsilon^{w}_{\mathrm{rel}}$	$\epsilon^{d}_{\mathrm{rel}}$	RMD	far-OOD, all	spread of FPR
0.01	**0.253**	0.171	0.146	≥ 0.9997	0.008
0.05	**0.674**	0.461	0.364	1.000	0.060
0.10	**0.829**	0.644	0.481	1.000	0.118

by 0.118 at nominal 0.10—some clusters are flagged several times more often than others while the pooled rate is exact. A per-query, geometry-aware quantity is what one would want instead, and that is the argument this paper makes for ϵ; measuring how well it serves that purpose, rather than as a detector, is left to future work.

Positioning. This experiment extends the stance of the main text—the bound holds by construction, its scale $\|f\|_{\mathcal{H}_k}$ does not, and what can be measured is that scale—to a real large language model, where the measurement exceeds the state-of-the-art model-based score of a published benchmark, with a margin invariant to every numerical and convention choice we varied. As in the main text, ϵ is a score a conformal scheme can wrap rather than a rival to one; within OOD detection, ϵ_{rel} reads at once as the RKHS counterpart of RMD and, through the identity above, as a kernel- and map-explicit generalization of SNGP-style Gaussian-process scores: per-query, geometry-aware, and interpretable through its decomposition (21).

A.11 A CERTIFIABLE SURROGATE FOR THE DEEP KERNEL READOUT

Setup. Fix a symmetric positive-definite kernel k on $\mathbb{R}^E$ with RKHS $\mathcal{H}_k$, and write pointwisely the trained readout block as

$$M(z) \;=\; k(zB,\, xA)\, c, \tag{22}$$

with projections $A, B \in \mathbb{R}^{D_x \times E}$, centers $x \in \mathbb{R}^{N \times D_x}$, and coefficients $c \in \mathbb{R}^{N \times D_y}$ (the doubly-projected readout (9) of Section 2.3, with $c \equiv \theta_y$, $B \equiv P_z$, $A \equiv P_x$). For a query z, the row $k(zB, xA) \in \mathbb{R}^{1 \times N}$ pairs the projected query zB against the projected centers xA. Because (22) carries no Gram inverse, $M(x) \neq c$ and M is *not* the interpolant of (x, c); the Schaback power function, which certifies an interpolant, therefore does not bound M directly.

Surrogate. Let (x_0, y_0) be the training set, $x_0 \in \mathbb{R}^{N_0 \times D_x}$, $y_0 \in \mathbb{R}^{N_0 \times D_y}$, and define the full-data interpolant [14]

$$R(z) \;=\; k(zB,\, x_0A)\, k(x_0A, x_0A)^{-1}\, y_0. \tag{23}$$

Both the deployed readout M and the surrogate R are elements of $\mathcal{H}_k$ as functions of the projected argument, and R interpolates the data in that space.

Schaback bound on the surrogate. Since R is the $\mathcal{H}_k$-interpolant of any target $f \in \mathcal{H}_k$ on the centers x_0A, the standard power-function estimate applies: for every z,

$$\big|f(z) - R(z)\big| \;\leq\; \epsilon_k(zB, x_0A)\, \|f\|_{\mathcal{H}_k}, \tag{24}$$

where $\epsilon_k(z, x)$ denotes the Schaback power function of the centers x defined in (4).

Distortion. As $M, R \in \mathcal{H}_k$, the distortion $M - R \in \mathcal{H}_k$. By the reproducing property, for every z,

$$\big|M(z) - R(z)\big| \;\leq\; \|M - R\|_{\mathcal{H}_k}\, \sqrt{k(zB, zB)}. \tag{25}$$

For scalar output ($D_y = 1$), writing $c_0 = k(x_0A, x_0A)^{-1} y_0$,

$$\|M - R\|^2_{\mathcal{H}_k} \;=\; c^\top k(xA, xA)\, c \;-\; 2\, c^\top k(xA, x_0A)\, c_0 \;+\; y_0^\top k(x_0A, x_0A)^{-1} y_0. \tag{26}$$

(For vector output the three scalars are replaced by their traces.)

[14] Or the sparse version of section 2.2.2, using the power-function (7).

Certificate. Combining (24) and (25) by the triangle inequality yields, for any target $f \in \mathcal{H}_k$ and any z,

$$\big|f(z) - M(z)\big| \;\le\; \underbrace{\big|M(z) - R(z)\big|}_{\text{distortion (carries } c)} \;+\; \underbrace{\epsilon_k(zB, x_0A)\,\|f\|_{\mathcal{H}_k}}_{\text{Schaback on } x_0 \;(c\text{-free})}. \tag{27}$$

The first term is the deep-versus-dense discrepancy between the deployed readout M and the dense full-data interpolant R; it is computable and carries all the dependence on the trained coefficients c. The second is the Schaback term on the c-free reference R, small where the data x_0 covers the query.

Reading. Both terms are computable: the distortion pointwise from (22) and (23), or uniformly via (25)–(26) together with a bound on $\|f\|_{\mathcal{H}_k}$. Neither term is small by construction. The certificate is tight exactly when the deployed readout stays close to the dense interpolant and the query lies within the data's coverage; its tightness is thus an empirical condition that is empirical, the unknown scale $\|f\|_{\mathcal{H}_k}$ being measured rather than assumed. The surrogate R is an offline certification object, computed once, and is not itself deployed.

From a single block to a composition. The certificate (27) bounds one readout block evaluated on a given representation. We now extend it to a full model, so as to delimit the class of architectures to which the estimate applies. We work on a bounded *coverage* set $\Omega \subset \mathbb{R}^{D_0}$ (the region populated by the data) and write the model as a composition

$$\widehat{\Phi} = \Phi_L \circ \Phi_{L-1} \circ \cdots \circ \Phi_1, \qquad z_0 = u \in \Omega, \quad z_\ell = \Phi_\ell(z_{\ell-1}), \tag{28}$$

with intermediate representations $z_\ell \in \Omega_\ell := \Phi_\ell(\Omega_{\ell-1})$.

Local admissibility. We say that $\widehat{\Phi}$ is *locally kernel-admissible on* Ω if every block is of one of two types:

(K) a *kernel readout* of the form (22), i.e. $\Phi_\ell = M_\ell$ with $M_\ell(z) = k_\ell(zB_\ell, x_\ell A_\ell)\, c_\ell$ for a symmetric positive-definite kernel k_ℓ, projections A_ℓ, B_ℓ, learnable centers x_ℓ and coefficients c_ℓ (we write $\ell \in \mathcal{K}$);

(L) a map $\Phi_\ell = g_\ell$ that is *locally Lipschitz* on the coverage, with constant $L_\ell := \mathrm{Lip}_{\Omega_{\ell-1}}(g_\ell) < \infty$.

For each $\ell \in \mathcal{K}$ we attach, as in the single-block construction, the dense *local* surrogate R_ℓ built from the reference data on the cell containing $z_{\ell-1}B_\ell$ (the sparse, local version of (7)), and we let $\widehat{R} = \Phi_L^R \circ \cdots \circ \Phi_1^R$ be the *surrogate model*, in which each M_ℓ is replaced by R_ℓ and each type-(L) block is left unchanged. As above, $\widehat{R}$ is an offline certification object and is not deployed. For $\ell < L$ we set $\Lambda_{>\ell}(u) = \prod_{j=\ell+1}^{L} \mathrm{Lip}_{\Omega_{j-1}}(\Phi_j)$ (with $\Lambda_{>L} \equiv 1$); type-(K) blocks, being finite sums of smooth kernel sections, are themselves locally Lipschitz on the bounded coverage and so enter this product.

Definition (model of kernel type). A model is *of kernel type* (on Ω) if it is locally kernel-admissible on Ω.

Proposition 1 (Local composition certificate)**.** *Let $\widehat{\Phi}$ be of kernel type on Ω. Then for every $u \in \Omega$,*

$$\big\|\widehat{\Phi}(u) - \widehat{R}(u)\big\| \le \sum_{\ell \in \mathcal{K}} \Lambda_{>\ell}(u)\, \|M_\ell - R_\ell\|_{\mathcal{H}_{k_\ell}} \sqrt{k_\ell\big(z_{\ell-1}B_\ell,\, z_{\ell-1}B_\ell\big)}, \tag{29}$$

where each $\|M_\ell - R_\ell\|_{\mathcal{H}_{k_\ell}}$ is given in closed form by (26)*. If moreover, at each $\ell \in \mathcal{K}$, the local reference interpolates a target $f_\ell \in \mathcal{H}_{k_\ell}$ and the type-(L) blocks are common to $\widehat{\Phi}$ and the target map $F^\star$, then for every $u \in \Omega$,*

$$\begin{aligned}\big\|F^\star(u) - \widehat{\Phi}(u)\big\| \;\le\; & \underbrace{\sum_{\ell \in \mathcal{K}} \Lambda_{>\ell}(u)\, \|M_\ell - R_\ell\|_{\mathcal{H}_{k_\ell}} \sqrt{k_\ell(z_{\ell-1}B_\ell, z_{\ell-1}B_\ell)}}_{\textit{distortion (computable)}} \\ & + \underbrace{\sum_{\ell \in \mathcal{K}} \Lambda_{>\ell}(u)\, \epsilon^{\sigma}_{k_\ell}\big(z_{\ell-1}B_\ell,\, x_{0,\ell}A_\ell\big)\|f_\ell\|_{\mathcal{H}_{k_\ell}}}_{\textit{Schaback (c-free)}},\end{aligned} \tag{30}$$

where $\epsilon^{\sigma}_{k_\ell}$ is the local power function (7)*.*

Proof sketch. Replace M_ℓ by R_ℓ one block at a time, from input to output, forming hybrids $H_0 = \widehat{\Phi}, \ldots, H_L = \widehat{R}$. Two consecutive hybrids differ only at a single kernel block ℓ and share all downstream blocks; the single-block distortion bound (25) controls the discrepancy at the output of block ℓ by $\|M_\ell - R_\ell\|_{\mathcal{H}_{k_\ell}} \sqrt{k_\ell(\cdot,\cdot)}$, which the downstream maps amplify by at most $\Lambda_{>\ell}(u)$, each being Lipschitz on the relevant coverage. Summation over $\mathcal{K}$ and the triangle inequality give (29). For (30), split $F^\star - \widehat{\Phi} = (F^\star - \widehat{R}) + (\widehat{R} - \widehat{\Phi})$: the second term is (29), and the first follows from the same telescoping with the per-block Schaback estimate (24) in place of the distortion, the type-(L) blocks contributing nothing as they are common to $\widehat{R}$ and $F^\star$. □

Consequence. The augmented model of Appendix A.8 is of kernel type: the kernel modules added to its feed-forward paths are readouts of the form (22); its attention sub-blocks are locally Lipschitz on bounded domains (Kim et al., 2021; Bohbot et al., 2026); and its retained parametric FFN sub-blocks, normalization, and patch-embedding maps are (locally) Lipschitz on the bounded domain Ω (normalization away from degenerate, zero-variance inputs), hence type-(L) blocks. Its classification head is itself of the form (22), hence a type-(K) block: the direct Schaback bound (4) requires *observed* targets, $\theta_y = y$ as in instance (i) of Section 2.3—it is available for the lazy readouts of Sections 3.1 and 3.2—whereas a head with free coefficients is not the interpolant of any observed data and carries a distortion term like every other kernel block. For the augmented ViT no block therefore escapes the surrogate, and (30) applies with the head included in $\mathcal{K}$. The full transformer nonetheless admits an end-to-end, computable a posteriori error analysis, obtained by propagating the readout certificates through the intervening Lipschitz maps; the resulting constants grow with depth and with the domain radius (cf. Bohbot et al., 2026, Cor. D.7). In the PPO and tabular experiments the feature map is the identity (or linear), so $\mathcal{K} = \{L\}$ and (30) reduces to the single-block certificate (27); these are the *pure* case, with no Lipschitz factors from non-kernel blocks.

Remark 4 (attention and locality)**.** *Standard dot-product self-attention is* not *globally Lipschitz (Kim et al., 2021), so the global surrogate* (23) *cannot be propagated through an attention sub-block. On the bounded coverage Ω, each attention sub-block nonetheless has a finite local Lipschitz constant, which is precisely the quantity entering $\Lambda_{>\ell}$. Locality is therefore not a refinement but a prerequisite for the composition to traverse non-kernel blocks. As in the single-block case, both terms of* (30) *are computable and neither is small by construction; how tight the composition is remains a property to measure on Ω.*

A.11.1 Tightness is an empirical property, measured here on CIFAR-10.

The bound (24) cannot be checked directly: $\|f\|_{\mathcal{H}_k}$ is the norm of the *unknown* target, and it is not $\theta^T k(x,x)\theta$, which is the norm of a constructed function—the interpolant being of minimal norm among interpolants, substituting one for the other would *under*-state the bound. What can be measured is the smallest constant that makes the bound true at each query,

$$C(z) \;=\; \frac{|f(z) - R(z)|}{\epsilon_k(z)}, \tag{31}$$

on a set held out from the centers. Probing at $N_x = 1000$ and depth three, one seed, on the $10\,000$ CIFAR-10 test images (verified disjoint from the centers), C has median 1.19 (resp. 1.08) on the transformer (resp. the convolutional network), tenth and ninetieth percentiles 0.72 and 1.57 (resp. 0.50 and 1.57), and maximum 2.66 (resp. 2.51). No query exceeds five times the median.

Two things follow, and they are of unequal weight. That C is *concentrated* is largely structural: writing g_z for the Riesz representer of $f \mapsto f(z) - s_f(z)$, one has $\epsilon_k(z) = \|g_z\|$ and $C(z) = |\langle f, \hat{g}_z\rangle|$, a projection of a fixed f onto a varying unit direction, and such projections concentrate in high dimension. That C is *bounded by a small number* is not structural: it says f behaves as an element of $\mathcal{H}_k$ of norm about 2.5, which the one-hot target read through a Matérn-$1/2$ had no obligation to do. And no query has a small ϵ_k with a large error—the dangerous case for a certificate—which the absence of a right tail rules out here.

We therefore report $\hat{C} \approx 2.5$ as an empirical stand-in for $\|f\|_{\mathcal{H}_k}$ on this task, conditional on $f \in \mathcal{H}_k$ and never derived from a constructed norm. It is one task, one depth, one seed.

A.12 GREEDY SELECTION

In some settings one wishes to reduce the computational cost of dense or sparse KRR. A simple approach is to select a representative subsequence of $N' < N$ points from the dataset $(x^1, \dots, x^N)$ by means of an incremental nested sequence $X^n \subset X^{n+1} \subset \cdots \subset X^N$, and the following greedy rule[15]:

$$X^{n+1} = X^n \cup \arg \sup_{x' \in \{x^{n+1}, \dots, x^N\}} d(x', X^n), \tag{32}$$

where $d(x', X^n) = \inf_{x \in X^n} d(x', x)$ and d is a user-defined distance.

A.13 OPTIMAL TRANSPORT ERROR LOSS FUNCTION

Optimal-transport losses produce continuous *bijective maps* that can match discrete features $x \mapsto y$ up to a permutation $\sigma \in \Sigma$ with kernel machines. Suppose first that (x, y) live in the same space $\mathbb{R}^D$. For a local cost $c(\cdot, \cdot)$, consider the transport functional

$$M(x, y) = \sum_{n=1}^{N} c\big(x^n, y^n\big). \tag{33}$$

Minimizing it over permutations, $\overline{\sigma} = \arg\inf_{\sigma \in \Sigma} \sum_n c(x^n, y^{\sigma^n})$, recovers the c-convex *Monge* problem, which can be solved by linear-sum-assignment algorithms.

The *Gromov–Monge* functional generalizes this to unrelated spaces $D_x \neq D_y$ via two cost functions c_x, c_y (e.g., Euclidean):

$$GM(x, y) = \sum_{i,j=1}^{N} \big| c_x(x^i, x^j) - c_y(y^i, y^j) \big|^2. \tag{34}$$

This functional is a natural candidate to add to a base loss, $\mathcal{L}(\theta) + GM(\theta_x P_x, \theta_y)$, in order to encourage a balanced spatial distribution of points whenever the embedded centers $\theta_x P_x$, the targets θ_y, or both, are learnable parameters of a kernel readout as in (14).

A.14 HIERARCHICAL, CONTINUOUS SPARSE KERNEL REGRESSORS

As noted in the main text, the space $\mathcal{H}_k^M$ contains functions that are continuous on each cell Ω^σ but possibly discontinuous globally, because of the discontinuity of $\mathbf{1}_{\Omega^\sigma}(z)$. When global continuity is required, we describe two alternatives; the first follows Han et al. (2022).

To obtain a *linear* globally continuous model class, one introduces a continuous partition of unity, namely a family of continuous functions $(w^\sigma)_{\sigma \in \Sigma}$ subordinate to the tessellation (locally finite overlap, $\sum_\sigma w^\sigma(z) = 1$), and considers

$$\mathcal{H}_k^{M,\mathrm{c}} = \Big\{ y_k(\cdot) = \sum_\sigma w^\sigma(\cdot)\, y_{k,\theta_y^\sigma}(\cdot), \quad y_{k,\theta_y^\sigma} \in \mathcal{H}_{k,\sigma} \Big\}, \tag{35}$$

equipped with the induced (image/quotient) norm

$$\|y_k\|^2_{\mathcal{H}_k^{M,\mathrm{c}}} \coloneqq \inf \Big\{ \sum_\sigma \|y_{k,\theta_y^\sigma}\|^2_{\mathcal{H}_{k,\sigma}} : y_k = \sum_\sigma w^\sigma y_{k,\theta_y^\sigma} \Big\}.$$

Under mild boundedness and local finiteness conditions, $\mathcal{H}_k^{M,\mathrm{c}}$ is a Hilbert space. The resulting blended predictor $y_k(z) = \sum_\sigma w^\sigma(z)\, y_{k,\theta_y^\sigma}(z)$ is globally defined and avoids discontinuities at cell boundaries. A typical choice for the weights is the sparse Nadaraya–Watson form

$$w^\sigma(\cdot) = \frac{\varphi(d(\cdot, \Omega^\sigma))}{\sum_{\tau \in \Sigma} \varphi(d(\cdot, \Omega^\tau))},$$

for some positive function φ, where $d(\cdot, \Omega^\sigma)$ is a distance-type function, e.g., $d(z, \Omega^\sigma) = \inf_n |z - x^{\sigma^n}|$ or simply $|z - x^{\sigma^1}|$ as in Han et al. (2022).

[15] Often referred to as the Gonzalez algorithm; see, e.g., `https://www.cs.umd.edu/class/spring2025/cmsc451-0101/Lects/lect06-greedy-k-center.pdf`.

A second way to obtain a globally continuous regressor is to use a positive definite kernel k that vanishes on cell boundaries, i.e., $k^\sigma(x, y) = 0$ whenever x or y lies on the boundary Γ^σ of Ω^σ. Denoting by $\mathcal{H}^0_{k,\sigma}$ the resulting RKHS on Ω^σ, we introduce the following hierarchical sparse construction:

$$\mathcal{H}_k^{M,\mathrm{h}} = \Big\{ y_k(\cdot) = y_{k,\theta_y^0}(\cdot) + \sum_{\sigma\in\Sigma} \mathbf{1}_{\Omega^\sigma}(\cdot)\, y_{k,\theta_y^\sigma}(\cdot), \quad y_{k,\theta_y^\sigma} \in \mathcal{H}^0_{k,\sigma},\ y_{k,\theta_y^0} \in \mathcal{H}_{k,\sigma^0} \Big\}, \tag{36}$$

where $\mathcal{H}_{k,\sigma^0}$ is a global, dense, coarse RKHS as in Section 2.1, built from a small subset x^{σ^0} of selected points—obtained, for instance, via the greedy procedure of (32).

A.15 Tabular benchmark: credit risk

To complement the vision and reinforcement-learning experiments with tabular data, we evaluate the deep kernel construction on *Give Me Some Credit*, a large class-imbalanced credit-scoring benchmark (≈150k clients, ≈7% default rate, with median replacement of missing values). We report metrics suited to imbalance—AUC, KS, the Brier score, and PR-AUC—averaged over ten seeds, comparing Deep KRR (a learned feature map with a kernel readout, Eq. (9)) to a plain KRR readout on raw features, a two-layer MLP, logistic regression, and XGBoost. Three observations follow (Table 16): (i) Deep KRR is competitive with the MLP across metrics, a third instance of kernel–neural parity; (ii) gradient boosting leads, as expected on tabular data, and we make no claim of improving on it; (iii) a naive plain KRR collapses while Deep KRR recovers, indicating that the learned feature map—not the kernel alone—drives competitiveness, in line with the paper's thesis. The setting is a predictive benchmark, not a deployed scoring system.

Table 16: Credit-risk benchmark on *Give Me Some Credit* (mean ± std over ten random initializations on a fixed train/test split; deterministic baselines report a single value). Higher is better for AUC, KS, PR-AUC; lower for the Brier score.

Model	AUC	KS	Brier	PR-AUC
Logistic regression	0.810	0.483	0.0541	0.360
MLP (two-layer)	0.822 ± 0.002	0.513 ± 0.003	0.0524 ± 0.0003	0.369 ± 0.003
XGBoost	0.856	0.557	0.0510	0.389
KRR (raw features)	0.754 ± 0.004	0.404 ± 0.010	0.2000 ± 0.0017	0.213 ± 0.010
Deep KRR (ours)	0.819 ± 0.000	0.510 ± 0.001	0.0533 ± 0.0009	0.361 ± 0.003

A.16 Substituting sub-blocks: what a readout can and cannot replace

Appendix A.8 reports gains obtained by *augmenting* the ViT feed-forward path with the doubly-projected readout (9). A natural, stronger claim would be that a kernel readout can *replace* the FFN sub-block outright; our experiments do not support it for either readout, and we record the negative result here for completeness.

Replacing each FFN sub-block with the symmetric readout (14) degrades test accuracy to 73.1, against 80.2 for the canonical ViT baseline at $N{=}50000$, only modestly above an ablated, purely *linear* FFN with the ReLU removed (69.8); substituting the doubly-projected readout (9) instead—free coefficients θ_y and distinct embeddings $P_z \neq P_x$—degrades accuracy comparably (73.2). Conversely, the augmentation regime is robust to the choice of readout: *composing* the retained FFN with a parallel module of the form (12) yields gains similar to the (9)-augmentation of Figures 3(c-d).

The operative distinction is therefore not the choice of kernel readout but the *regime*. With the parametric FFN removed, both readouts land close to the linear-FFN ablation: on these representations, neither reproduces the function the perceptron implements, whose removal costs about 7 points regardless of the substitute. With the FFN retained, both readouts add complementary capacity, and the memory-augmentation reading (Sukhbaatar et al., 2019; Lample et al., 2019) applies to this composition—not to a replacement.

Whole blocks, read by a kernel head, are a different matter. The negative result above concerns the *feed-forward sub-block* of an encoder layer, whose output is then consumed by the *parametric*

attention of the following layers. Replacing whole encoder blocks, and reading the result with a kernel head, does not degrade. At $N_x = 1000$ on the transformer we chained transition readouts $y^k \mapsto y^{k+1}$—centers the activations at level k, coefficients those at level $k+1$—and a network in which *no* encoder block survives, only the patch embedding followed by four such readouts, reaches 0.382 against 0.376 for the same model with its trained head. Each link interpolates exactly on its centers, so the chain introduces no error of its own; what is lost in test accuracy is generalization.

Two caveats. The full chains remain 8 to 9.5 points below the best direct probe, which traverses no predicted transition; and the head they beat is a weak reference, the transformer overfitting heavily at this budget (0.489 on its own training set against 0.376 in test).

And the two results are consistent once one notices that a reconstruction is only as good as what reads it next: per-position transitions reconstruct better and compose well with a kernel head, while flattened transitions reconstruct worse but produce an error coherent across tokens, which parametric attention tolerates better. Substitutability is a property of the pair, not of the readout alone.

One transition is hard, and measurably so. The first link, from the patch embedding to the first attentional representation, is the most expensive of the chain—5 points against 0 to 2 for the interior links, in two independent aggregations. Per position it is not merely expensive but *singular*: the patch embedding contains 63 strictly identical tokens out of $64\,000$—saturated uniform patches at the same position in different images—so two rows of the Gram matrix coincide and the Cholesky factorization fails at nugget 10^{-9}. The four other levels contain none.

These observations delimit the thesis of the paper: in our experiments, kernel readouts substitute for parametric components at the *interfaces* of a model (classification and policy heads) and *augment* its interior blocks; substitution of interior perceptron sub-blocks fails for both readouts on these representations.