# Tractable Uncertainty-Aware Meta-Learning

**Young-Jin Park*** MIT
**Cesar Almecija*** MIT
**Apoorva Sharma†** NVIDIA
**Navid Azizan** MIT

## Abstract

Meta-learning is a popular approach for learning new tasks with limited data by leveraging the commonalities among different tasks. However, meta-learned models can perform poorly when context data is too limited, or when data is drawn from an out-of-distribution (OoD) task. Especially in safety-critical settings, this necessitates an uncertainty-aware approach to meta-learning. In addition, the often multimodal nature of task distributions can pose unique challenges to meta-learning methods. To this end, we present LUMA, a meta-learning method for *regression* that (1) makes probabilistic predictions on in-distribution tasks efficiently, (2) is capable of detecting OoD context data, and (3) handles heterogeneous, multimodal task distributions effectively. The strength of our framework lies in its solid theoretical basis, enabling analytically tractable Bayesian inference on a linearized model for principled uncertainty estimation and robust generalization. We achieve this by adopting a probabilistic perspective and learning a parametric, tunable task distribution via Bayesian inference on a linearized neural network, leveraging Gaussian process theory. Moreover, we make our approach computationally tractable by leveraging a low-rank prior covariance learning scheme based on the Fisher Information Matrix. Our numerical analysis demonstrates that LUMA quickly adapts to new tasks and remains accurate even in low-data regimes; it effectively detects OoD tasks; and that both of these properties continue to hold for multimodal task distributions.

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. 

## 1 INTRODUCTION

Learning to learn is arguably an essential part of natural intelligence but is still an active area of research in artificial intelligence. *Meta-learning* is a popular approach that aims to enable trained models to perform well on new tasks using limited data from the new task. It involves first a *meta-training* process, when the model learns useful features from a set of tasks. Then, at test time, using only a few datapoints (*context data*) from a new, unseen task, the model (1) *adapts* to the new task (i.e., performs *few-shot learning*) and then (2) *infers* by making predictions on new, unseen *query inputs* from the same task. A popular baseline for meta-learning, which has attracted considerable attention in the past few years, is model-agnostic meta-learning (MAML) (Finn et al., 2017), in which the adaptation process consists of fine-tuning the parameters of the model via gradient descent.

Despite their success, meta-learning methods can struggle in several ways when deployed in challenging real-world scenarios. First, when context data is too limited to fully identify the test-time task, accurate prediction can be challenging. As these predictions can be untrustworthy, this necessitates the development of meta-learning methods that can express uncertainty during adaptation (Yoon et al., 2018; Harrison et al., 2018). In addition, meta-learning models may not successfully adapt to "unusual" tasks, i.e., when test-time context data is drawn from an *out-of-distribution* (OoD) task not well represented in the training dataset (Jeong and Kim, 2020; Iwata and Kumagai, 2022). Finally, special care has to be taken when learning heterogeneous tasks. An important example is the case of tasks with a *multimodal* distribution, i.e., when there are no common features shared across all the tasks, but the tasks can be grouped into subsets (modes) in a way that the ones from the same subset share common features (Vuorio et al., 2019).

To address these challenges, we present LUMA (*Low-rank Uncertainty-aware Meta-learning Algorithm*), a

*Equal contribution. †Work completed prior to joining NVIDIA.

meta-learning method that leverages probabilistic tools to overcome the aforementioned issues for *regression* tasks. Specifically, LUMA models the true distribution of tasks with a learnable distribution constructed over a linearized neural network and uses analytic Bayesian inference to perform uncertainty-aware adaptation. Further, we show how LUMA effectively strikes a balance between learning a rich prior distribution over the weights and maintaining the expressivity of the network. Finally, through numerical analysis, we demonstrate that (1) our method allows for efficient probabilistic predictions on in-distribution tasks, (2) it is effective in detecting context data from OoD tasks at test time, and (3) both these findings continue to hold in the multimodal task-distribution setting.

In short, our key contributions are:

- We introduce a meta-learning framework for regression that models the task distribution via Bayesian inference on a linearized neural network (Section 5.1). This approach uniquely enables **analytically tractable posterior inference**, avoiding common (e.g., sample-based) approximations.
- To make our method scalable for deep networks, we introduce an efficient low-rank parameterization of the prior weight covariance based on the Fisher Information Matrix (FIM), making the approach **computationally tractable** (Section 5.2).
- The framework is extended to effectively handle **heterogeneous tasks** by modeling the task distribution as a mixture of Gaussian Processes, allowing it to adapt to different underlying task clusters. (Section 5.3)
- The analytically tractable posterior on the linearized model yields **principled uncertainty estimates** that provide superior OoD detection and robust few-shot learning performance, especially in low-data regimes.

## 2 RELATED WORK

**Bayesian inference with linearized DNNs.** Bayesian inference with neural networks is often intractable. Whereas LUMA linearizes the network to allow for practical Bayesian inference, existing work has used other approximations such as Laplace's method. Closely related to our work, Maddox et al. (2021) have linearized pre-trained networks and performed domain adaptation by conditioning the prior predictive with data from the new task. Our approach leverages a similar adaptation method and demonstrates how the prior distribution can be learned in a meta-learning setup.

**Meta-learning.** Probabilistic meta-learning models such as PLATIPUS or BaMAML (Yoon et al., 2018; Finn et al., 2018) augment MAML to perform approximate Bayesian inference. These approaches, like ours, learn (during meta-training) and make use of (at test-time) a prior distribution over the weights. In contrast, however, LUMA performs analytically tractable Bayesian inference on a linearized model at test-time. Therefore, unlike other probabilistic frameworks that estimate the posterior predictive distribution through sampling, our method yields an *analytically tractable* posterior distribution.

ALPaCA is a Bayesian meta-learning algorithm for neural networks, where only the last layer is Bayesian (Harrison et al., 2018). This framework yields an exact linear regression that uses as feature map the activations right before the last layer. Our work can be viewed as a generalization of ALPaCA, in the sense that LUMA restricted to the last layer matches ALPaCA's approach. The link between these methods is further discussed in Appendix C.

**Neural processes.** Neural Processes (NPs) (Garnelo et al., 2018b) are a family of meta-learning methods that parameterize stochastic processes via neural networks. Conditional Neural Processes (CNPs) (Garnelo et al., 2018a) use a permutation-invariant encoder to aggregate context data into a fixed-length latent representation for prediction. Transformer Neural Processes (TNP-D) (Nguyen and Grover, 2022) leverage attention mechanisms to capture richer context dependencies, achieving strong performance across a range of tasks. However, the encoder-based architecture of NPs maps context sets to a single latent representation, which can make it challenging to handle multimodal task distributions and to distinguish OoD tasks from in-distribution tasks. In contrast, our framework leverages a mixture-of-GPs formulation on linearized networks, providing analytically tractable per-component marginal likelihoods. This analytical inference allows for more principled and robust task-level OoD detection.

**Deep kernel learning.** Deep Kernel Transfer (DKT) combines deep feature extractors with GP inference by defining a kernel over learned feature outputs (Patacchiola et al., 2020). While DKT can leverage powerful backbones for strong regression performance, its uncertainty operates at the input level based on distance in feature space, rather than the task level. Our method operates in weight space via the Neural Tangent Kernel, enabling direct evaluation of the prior predictive for task-level uncertainty assessment.

**Meta-learning vs. fine-tuning.** A widely used approach for adapting foundation models is fine-tuning, but it can be computationally expensive and struggle when only a small number of labeled examples

are available. Meta-learning offers a more principled framework for adapting to families of related tasks, allowing for rapid generalization and greater robustness to domain shifts, particularly in low-data regimes.

A more comprehensive discussion of related work can be found in Appendix A.

# 3 PROBLEM STATEMENT

At test time, the prediction steps are broken down into (1) *adaptation*, that is identifying $f_i$ using $K$ context datapoints $(\boldsymbol{X}^i, \boldsymbol{Y}^i)$ from the task, and (2) *inference*, that is making predictions for $f_i$ on the *query inputs* $\boldsymbol{X}^i_*$. Later the predictions can be compared with the *query ground-truths* $\boldsymbol{Y}^i_*$ to estimate the quality of the prediction, for example in terms of mean squared error (MSE). The meta-training consists in learning valuable features from a *cluster of tasks*, which is a set of similar tasks (e.g., sines with different phases and amplitudes but same frequency), so that at test time the predictions can be accurate on tasks from the same cluster. We take a probabilistic, functional perspective and represent a cluster by $p(f)$, a theoretical distribution over the function space that describes the probability of a task belonging to the cluster. Learning $p(f)$ is appealing, as it allows for performing OoD detection in addition to making predictions. Adaptation amounts to computing the conditional distribution given test context data, and one can obtain an uncertainty metric by evaluating the negative log-likelihood (NLL) of the context data under $p(f)$.

Thus, our goal is to construct a parametric, learnable functional distribution $\tilde{p}_\xi(f)$ that approaches the theoretical distribution $p(f)$, with a structure that allows tractable conditioning and likelihood computation, even in deep learning contexts. In practice, however, we are not given $p(f)$, but only a meta-training dataset $\mathcal{D}$ that we assume is sampled from $p(f)$: $\mathcal{D} = \{(\widetilde{\boldsymbol{X}}^i, \widetilde{\boldsymbol{Y}}^i)\}_{i=1}^N$, where $N$ is the number of tasks available during training, the superscript $i$ indexes each task, and $(\widetilde{\boldsymbol{X}}^i, \widetilde{\boldsymbol{Y}}^i) \sim \mathcal{T}^i$ is the entire pool of data from which we can draw subsets of context data $(\boldsymbol{X}^i, \boldsymbol{Y}^i)$. Consequently, in the meta-training phase, we aim to optimize $\tilde{p}_\xi(f)$ to capture properties of $p(f)$, using only the samples in $\mathcal{D}$, as illustrated in Figure 1.

Once we have $\tilde{p}_\xi(f)$, we can evaluate it both in terms of how it performs for few-shot learning (by comparing the predictions with the ground truths in terms of MSE), as well as for OoD detection (by measuring how well the NLL of context data serves to classify in-distribution tasks against OoD tasks, measured via the AUC-ROC score).

# 4 BACKGROUND

## 4.1 Bayesian linear regression and Gaussian Processes

Efficient Bayesian meta-learning requires a tractable inference process at test time. In general, this is only possible analytically in a few cases. One of them is the Bayesian linear regression with Gaussian noise and a Gaussian prior on the weights. Viewing it from a non-parametric, functional approach, this model is equivalent to a Gaussian process (GP) (Rasmussen and Williams, 2005).

Let $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K) \in \mathbb{R}^{D_x \times K}$ be a batch of $K$ $D_x$-dimensional inputs, and let $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K) \in \mathbb{R}^{D_y K}$ be a vectorized batch of $D_y$-dimensional outputs. In the Bayesian linear regression model, these quantities are related according to $\boldsymbol{y} = \phi(\boldsymbol{X})^\top \hat{\boldsymbol{\theta}} + \varepsilon \in \mathbb{R}^{D_y K}$ where $\hat{\boldsymbol{\theta}} \in \mathbb{R}^P$ are the weights of the model, and the inputs are mapped via $\phi : \mathbb{R}^{D_x \times K} \to \mathbb{R}^{P \times D_y K}$. Notice how this is a generalization of the usual one-dimensional linear regression ($D_y = 1$).

If we assume a Gaussian prior on the weights $\hat{\boldsymbol{\theta}} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and a Gaussian noise $\varepsilon \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_\varepsilon)$ with $\boldsymbol{\Sigma}_\varepsilon = \sigma_\varepsilon^2 \boldsymbol{I}$, then the model describes a multivariate Gaussian distribution on $\boldsymbol{y}$ for any $\boldsymbol{X}$. Equivalently, this means that this model describes a GP distribution over functions, with mean and covariance function (or kernel)

$$\begin{aligned} \boldsymbol{\mu}_{\text{prior}}(\boldsymbol{x}_t) &= \phi(\boldsymbol{x}_t)^\top \boldsymbol{\mu}, \\ \text{cov}_{\text{prior}}(\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2}) &= \phi(\boldsymbol{x}_{t_1})^\top \boldsymbol{\Sigma} \phi(\boldsymbol{x}_{t_2}) + \boldsymbol{\Sigma}_\varepsilon \\ &=: k_{\boldsymbol{\Sigma}}(\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2}) + \boldsymbol{\Sigma}_\varepsilon. \end{aligned} \tag{1}$$

This GP enables tractable computation of the likelihood of any batch of data $(\boldsymbol{X}, \boldsymbol{Y})$ given this distribution over functions. The structure of this distribution is governed by the feature map $\phi$ and the prior over the weights, specified by $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

This distribution can also easily be conditioned to perform inference. Given a batch of data $(\boldsymbol{X}, \boldsymbol{Y})$, the posterior predictive distribution is also a GP, with an updated mean and covariance function

$$\begin{aligned} &\boldsymbol{\mu}_{\text{post}}(\boldsymbol{x}_{t_*}) = \boldsymbol{\mu}_{\text{prior}}(\boldsymbol{x}_{t_*}) \\ &\quad + k_{\boldsymbol{\Sigma}}(\boldsymbol{x}_{t_*}, \boldsymbol{X}) \left(k_{\boldsymbol{\Sigma}}(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{\Sigma}_\varepsilon\right)^{-1} \left(\boldsymbol{Y} - \boldsymbol{\mu}_{\text{prior}}(\boldsymbol{X})\right), \\ &\text{cov}_{\text{post}}(\boldsymbol{x}_{t_{1*}}, \boldsymbol{x}_{t_{2*}}) = k_{\boldsymbol{\Sigma}}(\boldsymbol{x}_{t_{1*}}, \boldsymbol{x}_{t_{2*}}) \\ &\quad - k_{\boldsymbol{\Sigma}}(\boldsymbol{x}_{t_{1*}}, \boldsymbol{X}) \left(k_{\boldsymbol{\Sigma}}(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{\Sigma}_\varepsilon\right)^{-1} k_{\boldsymbol{\Sigma}}(\boldsymbol{X}, \boldsymbol{x}_{t_{2*}}). \end{aligned} \tag{2}$$

Here, $\boldsymbol{\mu}_{\text{post}}(\boldsymbol{X}_*)$ represents our model's adapted predictions for the test data, which we can compare to $\boldsymbol{Y}_*$ to evaluate the quality of our predictions, for example, via mean squared error (assuming that test data

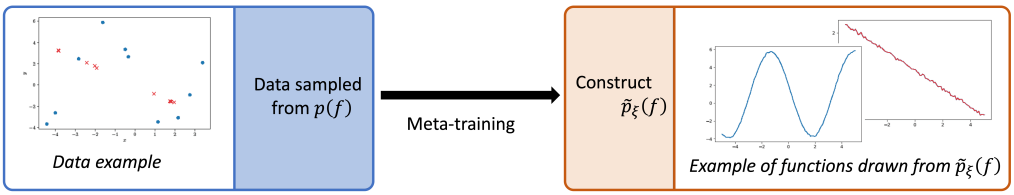



Figure 1: The true task distribution $p(f)$ can be multimodal, with multiple task clusters (e.g., lines and sines). LUMA models $p(f)$ with a tunable parametric distribution $\tilde{p}_\xi(f)$ via Bayesian linear regression on a linearized neural network.

is clean, following Rasmussen and Williams (2005)). The diagonal of $\text{cov}_{\text{post}}(\boldsymbol{X}_*, \boldsymbol{X}_*)$ can be interpreted as a per-input level of confidence that captures the ambiguity in making predictions with only a limited amount of context data.

### 4.2 The linearization of a neural network yields an expressive linear regression model

As discussed, the choice of feature map $\phi$ plays an important role in specifying a linear regression model. In the deep learning context, recent work has demonstrated that the linear model obtained when linearizing a deep neural network with respect to its weights at initialization, wherein the Jacobian of the network operates as the feature map, can well approximate the behavior of wide nonlinear deep neural networks, especially in the overparameterized regimes (Jacot et al., 2018; Azizan et al., 2021; Liu et al., 2020; Neal, 1996; Lee et al., 2018). Furthermore, Maddox et al. (2021) demonstrated that this linearized approximation effectively captures the local geometry of the loss landscape, making it well-suited for uncertainty-aware adaptation.

Let $f$ be a neural network $f : (\boldsymbol{\theta}, \boldsymbol{x}_t) \mapsto \boldsymbol{y}_t$, where $\boldsymbol{\theta} \in \mathbb{R}^P$ are the parameters of the model, $\boldsymbol{x} \in \mathbb{R}^{D_x}$ is an input and $\boldsymbol{y} \in \mathbb{R}^{D_y}$ an output. The linearized network (w.r.t. the parameters) around $\boldsymbol{\theta}_0$ is

$$f(\boldsymbol{\theta}, \boldsymbol{x}_t) - f(\boldsymbol{\theta}_0, \boldsymbol{x}_t) \approx \boldsymbol{J}_{\boldsymbol{\theta}}(f)(\boldsymbol{\theta}_0, \boldsymbol{x}_t)(\boldsymbol{\theta} - \boldsymbol{\theta}_0),$$

where $\boldsymbol{J}_{\boldsymbol{\theta}}(f)(\cdot, \cdot) : \mathbb{R}^P \times \mathbb{R}^{D_x} \to \mathbb{R}^{D_y \times P}$ is the Jacobian of the network (w.r.t. the parameters).

In the case where the model accepts a batch of $K$ inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K)$ and returns $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K)$, we generalize $f$ to $g : \mathbb{R}^P \times \mathbb{R}^{D_x \times K} \to \mathbb{R}^{D_y \times K}$, with $\boldsymbol{Y} = g(\boldsymbol{\theta}, \boldsymbol{X})$. Consequently, we generalize the linearization:

$$g(\boldsymbol{\theta}, \boldsymbol{X}) - g(\boldsymbol{\theta}_0, \boldsymbol{X}) \approx \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X})(\boldsymbol{\theta} - \boldsymbol{\theta}_0),$$

where $\boldsymbol{J}(\cdot, \cdot) : \mathbb{R}^P \times \mathbb{R}^{D_x \times K} \to \mathbb{R}^{D_y K \times P}$ is a shorthand for $\boldsymbol{J}_{\boldsymbol{\theta}}(g)(\cdot, \cdot)$. Note that we have implicitly vectorized the outputs, and throughout the work, we will interchange the matrices $\mathbb{R}^{D_y \times K}$ and the vectorized matrices $\mathbb{R}^{D_y K}$.

This linearization can be viewed as the $D_y K$-dimensional linear regression

$$\boldsymbol{z} = \phi_{\boldsymbol{\theta}_0}(\boldsymbol{X})^\top \hat{\boldsymbol{\theta}} \in \mathbb{R}^{D_y K}, \tag{3}$$

where the feature map $\phi_{\boldsymbol{\theta}_0}(\cdot) : \mathbb{R}^{D_x \times K} \to \mathbb{R}^{P \times D_y K}$ is the transposed Jacobian $\boldsymbol{J}(\boldsymbol{\theta}_0, \cdot)^\top$. The parameters of this linear regression $\hat{\boldsymbol{\theta}} = (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$ are the *correction* to the parameters chosen as the linearization point. Equivalently, this can be seen as a kernel regression with the kernel $k_{\boldsymbol{\theta}_0}(\boldsymbol{X}_1, \boldsymbol{X}_2) = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}_1)\boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}_2)^\top$, which is commonly referred to as the Neural Tangent Kernel (NTK) of the network. Note that the NTK depends on the linearization point $\boldsymbol{\theta}_0$. Building on these ideas, Maddox et al. (2021) show that the NTK obtained via linearizing a DNN *after* it has been trained on a task yields a GP that is well-suited for adaptation and fine-tuning to new, similar tasks. Furthermore, they show that networks trained on similar tasks tend to have similar Jacobians, suggesting that neural network linearization can yield an effective model for multi-task contexts such as meta-learning. In this work, we leverage these insights to construct our parametric functional distribution $\tilde{p}_\xi(f)$ via linearizing a neural network model.

## 5 METHODS

In this section, we describe our meta-learning regression algorithm LUMA and the construction of a parametric functional distribution $\tilde{p}_\xi(f)$ that can model the true underlying distribution over tasks $p(f)$. First, we focus on the single cluster case, where a Gaussian process structure on $\tilde{p}_\xi(f)$ can effectively model the true distribution of tasks, and detail how we can leverage meta-training data $\mathcal{D}$ from a single cluster of tasks to train the parameters $\xi$ of our model. Next, we will generalize our approach to the multimodal setting, with more than one cluster of tasks. Here, we

construct $\tilde{p}_\xi(f)$ as a mixture of GPs and develop a training approach that can automatically identify the clusters present in the training dataset without requiring the meta-training dataset to contain any additional structure such as cluster labels.

### 5.1 Tractable prior predictive distribution over functions

In our approach, we choose $\tilde{p}_\xi(f)$ to be the GP distribution over functions that arises from a Gaussian prior on the weights of the linearization of a neural network (equation 3). Consider a particular task $\mathcal{T}^i$ and a batch of $K$ context data $(\boldsymbol{X}^i, \boldsymbol{Y}^i)$. The resulting prior predictive distribution, derived from equation 1 after evaluating on the context inputs, is $\boldsymbol{Y}|\boldsymbol{X}^i \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{Y}|\boldsymbol{X}^i}, \boldsymbol{\Sigma}_{\boldsymbol{Y}|\boldsymbol{X}^i})$, where

$$\begin{aligned} \boldsymbol{\mu}_{\boldsymbol{Y}|\boldsymbol{X}^i} &= \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)\boldsymbol{\mu}, \\ \boldsymbol{\Sigma}_{\boldsymbol{Y}|\boldsymbol{X}^i} &= \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)\boldsymbol{\Sigma}\boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)^\top + \boldsymbol{\Sigma}_\varepsilon. \end{aligned} \tag{4}$$

In this setup, the parameters $\xi$ of $\tilde{p}_\xi(f)$ that we wish to optimize are the linearization point $\boldsymbol{\theta}_0$, and the parameters of the prior over the weights $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Given this Gaussian prior, it is straightforward to compute the joint NLL of the context labels $\boldsymbol{Y}^i$,

$$\begin{aligned} \mathrm{NLL}(\boldsymbol{X}^i, \boldsymbol{Y}^i) = \frac{1}{2}\Big( &\left\|\boldsymbol{Y}^i - \boldsymbol{\mu}_{\boldsymbol{Y}|\boldsymbol{X}^i}\right\|^2_{\boldsymbol{\Sigma}^{-1}_{\boldsymbol{Y}|\boldsymbol{X}^i}} \\ &+ \log\det \boldsymbol{\Sigma}_{\boldsymbol{Y}|\boldsymbol{X}^i} + D_y K \log 2\pi\Big). \end{aligned} \tag{5}$$

The NLL (a) serves as a loss function quantifying the quality of $\xi$ during training and (b) serves as an uncertainty signal at test time to evaluate whether context data $(\boldsymbol{X}^i, \boldsymbol{Y}^i)$ is OoD. Given this model, *adaptation* is tractable as we can condition this GP on the context data analytically. In addition, we can efficiently make probabilistic predictions by evaluating the mean and covariance of the resulting posterior predictive distribution on the query inputs, using equation 2.

### 5.2 Efficient parameterization of the prior covariance

When working with deep neural networks, the number of weights $P$ can easily surpass a million. While it remains tractable to deal with $\boldsymbol{\theta}_0$ and $\boldsymbol{\mu}$, whose memory footprint grows linearly with $P$, it can quickly become intractable to make computations with (let alone store) a dense prior covariance matrix over the weights $\boldsymbol{\Sigma} \in \mathbb{R}^{P\times P}$. Thus, we must impose some structural assumptions on the prior covariance to scale to deep neural network models.

**Imposing a unit covariance.** One simple way to tackle this issue would be to remove $\boldsymbol{\Sigma}$ from the learnable parameters $\xi$, i.e., fixing it to be the identity $\boldsymbol{\Sigma} = \boldsymbol{I}_P$. In this case, $\xi = (\boldsymbol{\theta}_0, \boldsymbol{\mu})$. This computational benefit comes at the cost of model expressivity, as we lose a degree of freedom in how we can optimize our learned prior distribution $\tilde{p}_\xi(f)$. In particular, we are unable to choose a prior over the weights of our model that captures correlations between elements of the feature map.

**Learning a low-dimensional representation of the covariance.** An alternative is to learn a low-rank representation of $\boldsymbol{\Sigma}$, allowing for a learnable weight-space prior covariance that can encode correlations. Specifically, we consider a covariance of the form $\boldsymbol{\Sigma} = \boldsymbol{Q}^\top \mathrm{diag}(\boldsymbol{s}^2)\boldsymbol{Q}$, where $\boldsymbol{Q}$ is a fixed projection matrix on an $r$-dimensional subspace of $\mathbb{R}^P$, while $\boldsymbol{s}^2$ is learnable. In this case, the parameters that are learned are $\xi = (\boldsymbol{\theta}_0, \boldsymbol{\mu}, \boldsymbol{s})$. We define $\boldsymbol{S} := \mathrm{diag}(\boldsymbol{s}^2)$. The computation of the covariance of the prior predictive (equation 4) could then be broken down into two steps:

$$\begin{cases} A := \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)\boldsymbol{Q}^\top \\ \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)\boldsymbol{\Sigma}\boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)^\top = A\boldsymbol{S}A^\top \end{cases}$$

which requires a memory footprint of $O(P(r + D_y K))$, if we include the storage of the Jacobian. Because $D_y K \ll P$ in typical deep learning contexts, it suffices that $r \ll P$ so that it becomes tractable to deal with this new representation of the covariance.

**A trade-off between feature-map expressiveness and learning a rich prior over the weights.** Note that even if a low-dimensional representation of $\boldsymbol{\Sigma}$ enriches the prior distribution over the weights, it also restrains the expressiveness of the feature map in the kernel by projecting the $P$-dimensional features $\boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X})$ on a subspace of size $r \ll P$ via $\boldsymbol{Q}$. This presents a trade-off: we can use the full feature map, but limit the weight-space prior covariance to be the identity matrix by keeping $\boldsymbol{\Sigma} = \boldsymbol{I}$: LUMA-I. Alternatively, we could learn a low-rank representation of $\boldsymbol{\Sigma}$ by randomly choosing $r$ orthogonal directions in $\mathbb{R}^P$, with the risk that they could limit the expressiveness of the feature map if the directions are not relevant to the problem that is considered: LUMA-R. To mitigate the issue of selecting random directions, we can choose the projection matrix more intelligently and project to the most impactful subspace of the full feature map—in this way, we can reap the benefits of a tunable prior covariance while minimizing the useful features that the projection drops. To select this subspace, we construct this projection map by choosing the top $r$ eigenvectors of the Fisher information matrix (FIM) evaluated on the training dataset $\mathcal{D}$: LUMA-F. The proposed FIM approach is inspired by (Sharma et al., 2021), which demonstrates that the

**Algorithm 1** LUMA-F

Find intermediate $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$ with LUMA-I ▷ see Alg. 2
Find $\boldsymbol{Q}$ via FIMProj(r); initialize $\boldsymbol{s}$. ▷ see Alg. 4
**for all** epoch **do**
  Sample $n$ tasks $\{\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)\}_{i=1}^{i=n}$
  **for all** $\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)$ **do**
    $\Sigma_i \leftarrow \boldsymbol{J}\boldsymbol{Q}^\top \mathrm{diag}(\boldsymbol{s}^2)\boldsymbol{Q}\boldsymbol{J}^\top + \boldsymbol{\Sigma}_\varepsilon$ ▷ $\boldsymbol{J} = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)$
    $NLL_i \leftarrow \textsc{GaussNLL}(\boldsymbol{Y}^i; \boldsymbol{J}\boldsymbol{\mu}, \Sigma_i)$
  **end for**
  Update $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$, $\boldsymbol{s}$ with $\nabla_{\boldsymbol{\theta}_0 \cup \boldsymbol{\mu} \cup \boldsymbol{s}} \sum_i NLL_i$
**end for**

FIM for deep neural networks exhibits rapid spectral decay, suggesting that retaining only a few top eigenvectors suffices to encode an expressive task-tailored prior. In the following sections, we use LUMA to refer to LUMA-F. The pseudo-code for LUMA-F is described in Algorithm 1. See Appendix B for more details, including the pseudo-codes for LUMA-I and LUMA-R. We also provide a detailed computational complexity analysis in the Appendix B.3, which shows that our method's **cost scales linearly** with the number of model parameters ($P$), comparable to MAML, ensuring its practicality for common meta-learning applications.

### 5.3 Generalization to a mixture of Gaussians

When learning on multiple clusters of tasks, $p(f)$ can become non-unimodal, and thus cannot be accurately described by a single GP. Instead, we can capture this multimodality by structuring $\tilde{p}_\xi(f)$ as a *mixture* of Gaussian processes.

**Building a more general structure.** We assume that at train time, a task $\mathcal{T}^i$ comes from any cluster $\{\mathcal{C}_j\}_{j=1}^{j=\alpha}$ with equal probability. Thus, we choose to construct $\tilde{p}_\xi(f)$ as an equal-weighted mixture of $\alpha$ Gaussian processes.

For each element of the mixture, the structure is similar to the single cluster case, where the parameters of the cluster's weight-space prior are given by $(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$. We choose to have both the projection matrix $\boldsymbol{Q}$ and the linearization point $\boldsymbol{\theta}_0$ (and hence, the feature map $\phi(\cdot) = \boldsymbol{J}(\boldsymbol{\theta}_0, \cdot)$) shared across the clusters. This yields improved computational efficiency, as we can compute the projected features once, simultaneously, for all clusters. This yields the parameters $\xi_\alpha = (\boldsymbol{\theta}_0, \boldsymbol{Q}, (\boldsymbol{\mu}_1, \boldsymbol{s}_1), \ldots, (\boldsymbol{\mu}_\alpha, \boldsymbol{s}_\alpha))$.

This can be viewed as a mixture of linear regression models, with a common feature map but separate, independent prior distributions over the weights for each cluster. These separate distributions are encoded using the low-dimensional representations $\boldsymbol{S}_j$ for each $\boldsymbol{\Sigma}_j$. Notice how this is a generalization of the single cluster case, for when $\alpha = 1$, $\tilde{p}_\xi(f)$ becomes a Gaussian and $\xi_\alpha = \xi$[1].

**Prediction and likelihood computation.** The NLL of a batch of inputs under this mixture model can be computed as

$$\mathrm{NLL}_{\mathrm{mixt}}(\boldsymbol{X}^i, \boldsymbol{Y}^i) = \log \alpha - \mathrm{LSE}(-\mathrm{NLL}_1(\boldsymbol{X}^i, \boldsymbol{Y}^i), \ldots, -\mathrm{NLL}_\alpha(\boldsymbol{X}^i, \boldsymbol{Y}^i)), \quad (6)$$

where $\mathrm{NLL}_j(\boldsymbol{X}^i, \boldsymbol{Y}^i)$ is the NLL with respect to each individual Gaussian, as computed in equation 5, and $\mathrm{LSE}(\cdot) := \log\sum\exp(\cdot)$ computes the logarithm of the sum of the exponential of these arguments, avoiding underflow issues.

To make exact predictions, we would require conditioning this mixture model. To simplify this, we propose to first *infer the cluster* from which a task comes from, by identifying the Gaussian $\mathcal{G}_{j_0}$ that yields the highest likelihood for the context data $\left(\boldsymbol{X}^i, \boldsymbol{Y}^i\right)$. Then, we can *adapt* by conditioning $\mathcal{G}_{j_0}$ with the context data and finally *infer* by evaluating the resulting posterior distribution on the queried inputs $\boldsymbol{X}_*^i$.

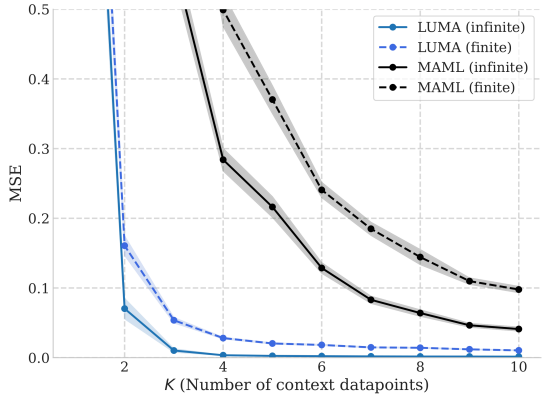
### 5.4 Meta-training the Parametric Task Distribution

The key to our meta-learning approach is to estimate the quality of $\tilde{p}_\xi(f)$ via the NLL of context data from training tasks, and use its gradients to update the parameters of the distribution $\xi$. Optimizing this loss over tasks in the dataset draws $\tilde{p}_\xi(f)$ closer to the empirical distribution present in the dataset, and hence towards the true distribution $p(f)$.

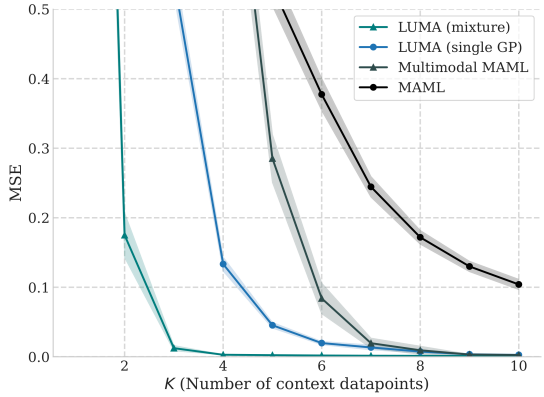
**Computing the likelihood.** In the algorithms, the function $\textsc{GaussNLL}(\boldsymbol{Y}^i;\ m,\ K)$ stands for NLL of $\boldsymbol{Y}^i$ under the Gaussian $\mathcal{N}(m, K)$ (see equation 5). In the mixture case, we instead use MixtNLL, which wraps equation 6 and calls GaussNLL for the individual NLL computations. In this case, $\boldsymbol{\mu}$ becomes $\{\boldsymbol{\mu}_j\}_{j=1}^{j=\alpha}$ and $\boldsymbol{s}$ becomes $\{\boldsymbol{s}_j\}_{j=1}^{j=\alpha}$ when applicable.

**Finding the FIM-based projections.** The FIM-based projection matrix aims to identify the elements of $\phi = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X})$ that are most relevant for the problem. However, this feature map evolves during training, because it is $\boldsymbol{\theta}_0$-dependent. How do we ensure that the directions we choose for $\boldsymbol{Q}$ remain relevant during training? We leverage results from Fort et al. (2020), stating that the NTK (the kernel associated with the Jacobian feature map) changes significantly at the beginning of training and that its evolution slows down

[1] In theory, it is possible to drop $\boldsymbol{Q}$ and extend the identity covariance case to the multi-cluster setting; however, this leads to each cluster having an identical covariance function, and thus is not effective at modeling heterogeneous behaviors among clusters.




(a) Unimodal




(b) Multimodal

Figure 2: MSE on predictions: We evaluate the performance of LUMA with varying numbers of context datapoints, $K$. In the unimodal setting, LUMA trained on a finite task dataset outperforms the baseline, even when the baseline is trained on an infinite task dataset. In the multimodal case, not only does our multimodal LUMA (mixture) outperform the baselines, but even the single-GP LUMA still surpasses them as well.

as training goes on. This suggests that as a heuristic, we can compute the FIM-based directions after partial training, as they are unlikely to deviate much after the initial training. For this reason, LUMA-F (Algorithm 1) first calls LUMA-I (Algorithm 2) before computing the FIM-based $\boldsymbol{Q}$ that yields intermediate parameters $\boldsymbol{\theta}_0$ and $\boldsymbol{\mu}$. Then the usual training takes place with the learning of $\boldsymbol{s}$ in addition to $\boldsymbol{\theta}_0$ and $\boldsymbol{\mu}$.

## 6 NUMERICAL ANALYSIS

In this section, we evaluate the empirical efficacy of the proposed framework by examining the following key aspects of LUMA: (1) Comparative accuracy of the proposed probabilistic framework against baselines in both unimodal and multimodal settings, (2) OoD detection performance of the proposed method, and (3) ablation study regarding the trade-off between learning a rich prior distribution and maintaining a complete feature map.

**Comparison with baselines.** To show the efficacy of LUMA, we compare it to the closely related prominent meta-learning frameworks, i.e., MAML (Finn et al., 2017) and Multimodal MAML (MMAML) (Vuorio et al., 2019), as well as additional baselines, namely, Conditional Neural Processes (CNP) (Garnelo et al., 2018a), Transformer Neural Processes (TNP-D) (Nguyen and Grover, 2022), and Deep Kernel Transfer (DKT) (Patacchiola et al., 2020). First, we consider a unimodal setting with the task distribution consisting of sinusoids of varying amplitude and phase. We use a neural network with 2 hidden layers, 40 neurons each, with a ReLU non-linearity, and a single GP structure of $\tilde{p}_\xi(f)$ for LUMA. In a unimodal setting, we also compare the results between training with an infinite amount of available sine tasks (infinite task dataset), and with a finite amount of available tasks (finite task dataset). We then extend the empirical analysis to a multimodal setting with training data consisting of sinusoids as well as lines with varying slopes. Details on the problems can be found in Appendix H, and the training and test details for unimodal and multimodal cases are in Appendix F and Appendix I, respectively.

As shown in Figure 2a, *the empirical evidence confirms that* LUMA *outperforms MAML, particularly in the low-data regime*; it achieves much better generalization particularly when we have a small number of context samples, $K$. Moreover, LUMA trained with a finite task dataset performs comparably to the one with an infinite dataset and predicts better than the baselines. This shows the robustness of LUMA, capturing the common features of the tasks even with limited meta-training. Examples of test predictions are in Figure 5 in the Appendix.

The same trend is further highlighted in the multimodal setting. Notably, LUMA with a single GP structure outperforms both MAML and MMAML in prediction. This highlights the strength of our probabilistic approach for multimodal meta-learning: even when the probabilistic assumptions are not perfectly aligned, the predictions remain accurate and surpass baseline methods. Moreover, this performance gap widens when incorporating a mixture structure into our framework.

Finally, we conducted a comparative analysis against probabilistic meta-learning baselines, PLATIPUS and BaMAML, as well as CNP, TNP-D, and DKT. As shown in Table 1, the results underscore the superior performance of our proposed method, which consistently achieves a lower prediction error across all eval-

uated tasks and datasets. In the unimodal infinite-task setting, neural processes (CNP, TNP-D) perform competitively, but their performance degrades substantially in the finite-task setting ($N = 10$), whereas LUMA maintains high accuracy, demonstrating superior sample efficiency across tasks. In the multimodal setting, our mixture model achieves the lowest error by explicitly capturing the task structure (see Table 5 in the Appendix for full results). We assess the reliability of their uncertainty estimates in the subsequent analysis section.

**Vision Regression Tasks.** In addition to our analysis with a shallow MLP, we extend our study to a more complex scenario using a deeper network for a unimodal vision meta-learning problem, ShapeNet1D (Gao et al., 2022), which aims to predict object orientations from the image. In this problem, each task consists of context data comprising images of the same object captured at different orientations, while the query inputs are additional images of the same object with unknown orientations to be predicted. Details on the problems and the datasets can be found in Appendix. As shown in Table 1, LUMA generalizes well to deeper networks. DKT excels at larger context sizes ($K \geq 10$) by leveraging its CNN backbone, while LUMA shows relatively more stable degradation in the extreme low-data regime ($K = 5$).

| | Unimodal | Multimodal | Vision |
|---|---|---|---|
| MAML | 0.2172 / 0.0314 | 0.5324 / 0.1041 | 22.41 / 17.42 |
| PLATIPUS | 0.2466 / 0.0680 | 0.4979 / 0.1160 | 57.74 / 55.08 |
| BaMAML | 0.4333 / 0.1359 | 0.9964 / 0.3292 | 21.80 / 17.64 |
| CNP | 0.0485 / 0.0189 | 0.1402 / 0.0311 | 22.31 / 19.70 |
| TNP-D | 0.1324 / 0.0186 | 0.1253 / 0.0196 | 89.75 / 88.74 |
| DKT | 3.2730 / 0.2122 | 2.4779 / 0.1760 | 21.32 / **3.920** |
| **Ours** | **0.0026** / **0.0015** | **0.0024** / **0.0012** | **18.94** / 7.684 |

Table 1: Prediction error comparison (MSE for regression; angular error for vision tasks) with $K$=5/10. Our method robustly outperforms the baselines on regression tasks, especially in low-data regimes (e.g., $K$=5).

**OoD detection performance.** To further examine the effectiveness of LUMA, we evaluate its OoD detection performance. Our framework provides a closed-form marginal likelihood for each task, which allows us to compute the AUC-ROC score using the negative log-likelihood of the context targets conditioned on the inputs under $\tilde{p}_\xi(f)$. We compare its reliability to the probabilistic baselines that work on sampling-based uncertainty estimation, as well as to CNP, TNP-D, and DKT. For this analysis, we consider a cluster of sine tasks, one of the linear tasks, and one of the quadratic tasks. In the unimodal setting, sine tasks are in-distribution (ID), while others are OoD. In the multimodal setting, sine and linear tasks are ID, with the quadratic task as OoD.

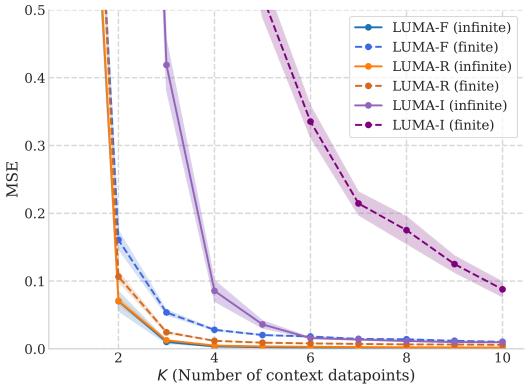
As illustrated in Table 2, across the different settings, the proposed framework achieves nearly perfect OoD detection accuracy even with a limited number of context data points (e.g., $K$=5). This further demonstrates the efficacy of our mixture-of-GPs structure in a multimodal setting. In the unimodal setting, CNP and TNP-D achieve comparable near-perfect detection. However, in the multimodal setting, their performance drops to near-random (AUC $\approx 0.5$), as their encoder maps context sets to a single latent representation, making it difficult to distinguish OoD tasks from in-distribution tasks of a different mode. DKT performs poorly across all settings, suggesting that its input-level uncertainty is not well-suited for task-level OoD detection. In contrast, our mixture-of-GPs formulation with tractable per-component likelihoods achieves near-perfect detection (AUC $> 0.99$) even in the multimodal case.

| Method | Unimodal | Multimodal |
|---|---|---|
| PLATIPUS | 0.8199 / 0.9438 | 0.6680 / 0.8363 |
| BaMAML | 0.6705 / 0.8474 | 0.5279 / 0.5786 |
| CNP | 0.9990 / **1.0000** | 0.5090 / 0.5110 |
| TNP-D | 0.9960 / **1.0000** | 0.5220 / 0.5340 |
| DKT | 0.2530 / 0.4710 | 0.1950 / 0.4230 |
| **Ours** | **1.0000** / **1.0000** | **0.9940** / **1.0000** |

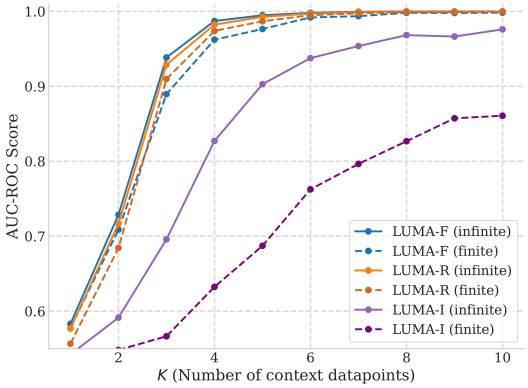
Table 2: Out-of-distribution detection performance comparison. AUC-ROC scores with $K$=5/10 are reported. LUMA achieves near-perfect scores across all settings. CNP and TNP-D perform well in the unimodal case but degrade to near-random in the multimodal setting.

**Trade-off analysis.** We provide an in-depth analysis comparing the performance between LUMA-I, LUMA-R, and LUMA-F to study a trade-off between learning a rich prior distribution over the weights and maintaining the full expressivity of the network. As shown in Figures 3a and 3b, LUMA-R and LUMA-F outperform LUMA-I, reflecting a higher-quality learned prior (see Appendix for details). For shallow networks, having a rich prior over the weights could be more beneficial than maintaining the full expressiveness of the feature map, making both LUMA-R and LUMA-F preferable. However, in deep networks (Figure 3c), preserving an expressive feature map becomes crucial, favoring LUMA-I and LUMA-F. Overall, the FIM-based approach, LUMA-F, is the most robust, consistently achieving superior performance.

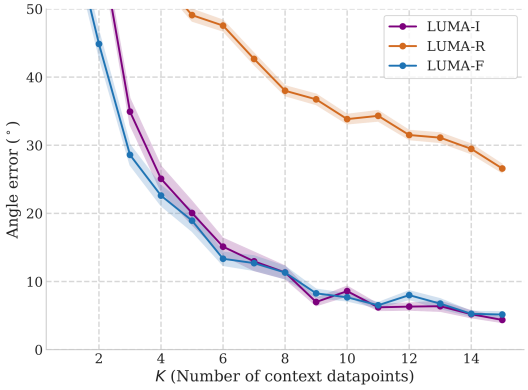
**Discussion.** Our experiments reveal that different meta-learning paradigms exhibit complementary strengths. Neural processes (CNP, TNP-D) are com-




(a) Sinusoid Task




(b) OoD Detection




(c) Vision Task

Figure 3: Trade-off analysis with LUMA variants.

petitive in the infinite-task setting but degrade substantially when meta-training tasks are limited. DKT leverages its deep feature extractor effectively on vision tasks with sufficient context. In contrast, LUMA demonstrates the most robust performance in low-data regimes and with limited meta-training data points. For OoD detection, our advantage is most pronounced in the multimodal setting, where all other baselines fail to reliably distinguish OoD tasks from in-distribution tasks of a different mode, thanks to our mixture-of-GPs formulation with tractable per-component likelihoods. These findings suggest that LUMA is particularly well suited to safety-critical few-shot regression where reliable uncertainty estimates and OoD detection are essential, such as autonomous vehicle dynamics adaptation and biomedical forecasting.

## 7 CONCLUSION

We proposed LUMA, a meta-learning algorithm that models the underlying task distribution using a parametric and tunable distribution, leveraging Bayesian inference with linearized neural networks. By incorporating a Fisher-information-based parameterization, LUMA strikes an effective balance between scalability and expressivity. We demonstrated that (1) our approach makes efficient probabilistic predictions on in-distribution tasks, (2) it is capable of effectively detecting OoD context data, and that (3) both of these findings continue to hold in the multimodal task-distribution setting.

**Future work.** There are several interesting avenues for future work. Our current framework, built on a Gaussian likelihood, is tailored for regression tasks; generalizing our approach to non-Gaussian likelihoods would enable LUMA to be used for classification, a direction we believe is highly promising. Furthermore, while our experiments on synthetic and vision-based tasks demonstrate the core strengths of our method, future work could involve evaluation on a broader range of large-scale meta-learning benchmarks to further validate its effectiveness and scalability. Additionally, our mixture inference uses hard cluster assignment; developing tractable soft-weighting mechanisms that maintain posterior mixture weights across components could yield a more flexible model.

We hope this work encourages the community to further explore the intersection of linearized neural networks and probabilistic meta-learning. We believe the principles of tractable Bayesian inference presented here can serve as a strong foundation for developing more robust, uncertainty-aware, and general-purpose learning systems.

## Acknowledgments

This work was supported in part by the MIT-IBM Watson AI Lab, the MIT-Google Program for Computing Innovation, the MIT-Amazon Science Hub, Netflix, and Jane Street. The authors acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing computing resources that have contributed to the results reported within this paper.

## References


Abdollahzadeh, M., Malekzadeh, T., and Cheung, N.-M. M. (2021). Revisit multimodal meta-learning through the lens of multi-task learning. *Advances in Neural Information Processing Systems*, 34:14632–14644.

Azizan, N., Lale, S., and Hassibi, B. (2021). Stochastic mirror descent on overparameterized nonlinear models. *IEEE Transactions on Neural Networks and Learning Systems*.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186.

Farajtabar, M., Azizan, N., Mott, A., and Li, A. (2020). Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3762–3773. PMLR.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR.

Finn, C., Xu, K., and Levine, S. (2018). Probabilistic model-agnostic meta-learning. *Advances in Neural Information Processing Systems*, 31.

Fort, S., Dziugaite, G. K., Paul, M., Kharaghani, S., Roy, D. M., and Ganguli, S. (2020). Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. *Advances in Neural Information Processing Systems*, 33:5850–5861.

Gao, N., Ziesche, H., Vien, N. A., Volpp, M., and Neumann, G. (2022). What matters for meta-learning vision regression tasks? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14776–14786.

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. (2018a). Conditional neural processes. In *International conference on machine learning*, pages 1704–1713. PMLR.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. (2018b). Neural processes. *arXiv preprint arXiv:1807.01622*.

Garriga-Alonso, A., Rasmussen, C. E., and Aitchison, L. (2019). Deep convolutional networks as shallow gaussian processes. In *International Conference on Learning Representations*.

Harrison, J., Sharma, A., and Pavone, M. (2018). Meta-learning priors for efficient online bayesian regression. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 318–337. Springer.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, A. M., Araujo, A., and Gesmundo, A. (2019). Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning (ICML)*, pages 2790–2799.

Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. *ACL*, pages 328–339.

Immer, A., Korzepa, M., and Bauer, M. (2021). Improving predictions of bayesian neural nets via local linearization. In *International Conference on Artificial Intelligence and Statistics*, pages 703–711. PMLR.

Iwata, T. and Kumagai, A. (2022). Meta-learning for out-of-distribution detection via density estimation in latent space. *arXiv preprint arXiv:2206.09543*.

Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *Advances in Neural Information Processing Systems*, 31.

Jeong, T. and Kim, H. (2020). Ood-maml: Meta-learning for few-shot out-of-distribution detection and classification. *Advances in Neural Information Processing Systems*, 33:3907–3916.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. (2018). Deep neural networks as gaussian processes. In *International Conference on Learning Representations*.

Lester, B., Al-Rfou, R., and Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Liu, C., Zhu, L., and Belkin, M. (2020). On the linearity of large non-linear models: when and why the tangent kernel is constant. *Advances in Neural Information Processing Systems*, 33:15954–15964.

Maddox, W., Tang, S., Moreno, P., Wilson, A. G., and Damianou, A. (2021). Fast adaptation with linearized neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 2737–2745. PMLR.

Neal, R. M. (1996). Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pages 29–53. Springer.

Nguyen, T. and Grover, A. (2022). Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. *arXiv preprint arXiv:2207.04179*.

Patacchiola, M., Turner, J., Crowley, E. J., O'Boyle, M., and Storkey, A. J. (2020). Bayesian meta-learning for the few-shot setting via deep kernels. *Advances in Neural Information Processing Systems*, 33:16108–16118.

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Richards, S. M., Azizan, N., Slotine, J.-J., and Pavone, M. (2023). Control-oriented meta-learning. *The International Journal of Robotics Research*, 42(10):777–797.

Ritter, H., Botev, A., and Barber, D. (2018). A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. International Conference on Representation Learning.

Rochussen, T. and Fortuin, V. (2025). Sparse gaussian neural processes. *arXiv preprint arXiv:2504.01650*.

Sagun, L., Evci, U., Guney, V. U., Dauphin, Y., and Bottou, L. (2017). Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*.

Sharma, A., Azizan, N., and Pavone, M. (2021). Sketching curvature for efficient out-of-distribution detection for deep neural networks. In *Uncertainty in Artificial Intelligence*, pages 1958–1967. PMLR.

Tang, S., Sun, H., and Azizan, N. (2025). Meta-learning for adaptive control with automated mirror descent. In *Proceedings of the 7th Annual Learning for Dynamics & Control Conference*, volume 283 of *Proceedings of Machine Learning Research*, pages 1025–1037. PMLR.

Titsias, M. (2009). Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR.

Tropp, J. A., Yurtsever, A., Udell, M., and Cevher, V. (2017). Practical sketching algorithms for low-rank matrix approximation. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1454–1485.

Vuorio, R., Sun, S.-H., Hu, H., and Lim, J. J. (2019). Multimodal model-agnostic meta-learning via task-aware modulation. *Advances in Neural Information Processing Systems*, 32.

Yoon, J., Kim, T., Dia, O., Kim, S., Bengio, Y., and Ahn, S. (2018). Bayesian model-agnostic meta-learning. *Advances in Neural Information Processing Systems*, 31.

## Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Yes

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes
   (b) Complete proofs of all theoretical results. Yes
   (c) Clear explanations of any assumptions. Yes

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes
   (b) The license information of the assets, if applicable. Yes
   (c) New assets either in the supplemental material or as a URL, if applicable. Yes
   (d) Information about consent from data providers/curators. Not Applicable
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not Applicable
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable

# Tractable Uncertainty-Aware Meta-Learning (Supplementary Material)

## A Related Work

**Bayesian inference with linearized DNNs.** Bayesian inference with neural networks is often intractable because the posterior predictive has rarely a closed-form expression. Whereas LUMA linearizes the network to allow for practical Bayesian inference, existing work has used other approximations to tractably express the posterior. For example, it has been shown that in the infinite-width approximation, the posterior predictive of a Bayesian neural network behaves like a GP (Neal, 1996; Lee et al., 2018). This analysis can in some cases yield a good approximation to the Bayesian posterior of a DNN (Garriga-Alonso et al., 2019). It is also common to use Laplace's method to approximate the posterior predictive by a Gaussian distribution and allow practical use of the Bayesian framework for neural networks. This approximation relies in particular on the computation of the Hessian of the network: this is in general intractable, and most approaches use the so-called Gauss-Newton approximation of the Hessian instead (Ritter et al., 2018). Recently, it has been shown that Laplace's method using the Gauss-Newton approximation is equivalent to working with a certain linearized version of the network and its resulting posterior GP (Immer et al., 2021).

Bayesian inference is applied in a wide range of subjects. For example, recent advances in transfer learning have been possible thanks to Bayesian inference with linearized neural networks. Maddox et al. (2021) have linearized pre-trained networks and performed domain adaptation by conditioning the prior predictive with data from the new task: the posterior predictive is then used to make predictions. Our approach leverages a similar adaptation method and demonstrates how the prior distribution can be learned in a meta-learning setup.

**Meta-learning.** MAML is a meta-learning algorithm that uses as adaptation a few steps of gradient descent (Finn et al., 2017). It has the benefit of being model-agnostic (it can be used on any model for which we can compute gradients w.r.t. the weights), whereas LUMA requires the model to be a differentiable regressor. MAML has been further generalized to probabilistic meta-learning models such as PLATIPUS or BaMAML (Yoon et al., 2018; Finn et al., 2018), where the simple gradient descent step is augmented to perform approximate Bayesian inference. These approaches, like ours, learn (during meta-training) and make use of (at test-time) a prior distribution over the weights. **In contrast, however, LUMA performs analytically tractable Bayesian inference on a linearized model at test-time**. Therefore, unlike other probabilistic frameworks that estimate the posterior predictive distribution through sampling, our method yields an *analytically tractable* posterior distribution. MAML has also been improved for multimodal meta-learning via MMAML (Vuorio et al., 2019; Abdollahzadeh et al., 2021). Similarly to our method, they add a step to identify the cluster from which the task comes from (Vuorio et al., 2019). OoD detection in meta-learning has been studied by Jeong and Kim (2020), who built upon MAML to perform OoD detection in the classification setting, to identify unseen classes during training. Iwata and Kumagai (2022) also implemented OoD detection for classification, by learning a Gaussian mixture model on a latent space. LUMA extends these ideas to the regression case, aiming to identify when test data is drawn from an unfamiliar function.

ALPaCA is a Bayesian meta-learning algorithm for neural networks, where only the last layer is Bayesian (Harrison et al., 2018). This framework yields an exact linear regression that uses as feature map the activations right before the last layer. **Our work is a generalization of ALPaCA**, in the sense that LUMA restricted to the last layer matches ALPaCA's approach. The link between these methods is further discussed in Appendix C. More broadly, control-oriented meta-learning approaches have been developed, e.g., by Richards et al. (2023); Tang et al. (2025), though they are not probabilistic.

**Algorithm 2** LUMA-I: meta-training with identity prior covariance

Initialize $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$.
**for all** epoch **do**
  Sample $n$ tasks $\{\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)\}_{i=1}^{i=n}$
  **for all** $\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)$ **do**
    $NLL_i \leftarrow \textsc{GaussNLL}(\boldsymbol{Y}^i; \boldsymbol{J}\boldsymbol{\mu},\ \boldsymbol{J}\boldsymbol{J}^\top + \boldsymbol{\Sigma}_\varepsilon)$ ▷ $\boldsymbol{J} = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)$
  **end for**
  Update $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$ with $\nabla_{\boldsymbol{\theta}_0 \cup \boldsymbol{\mu}} \sum_i NLL_i$
**end for**

**Algorithm 3** LUMA-R and LUMA-F: meta-training with a learnt covariance

**if** using random projections **then**
  Find random projection $\boldsymbol{Q}$
  Initialize $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$, $\boldsymbol{s}$
**else if** using FIM-based projections **then**
  Find intermediate $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$ with LUMA-I ▷ see Alg. 2
  Find $\boldsymbol{Q}$ via FIMProj(r); initialize $\boldsymbol{s}$. ▷ see Alg. 4
**end if**
**for all** epoch **do**
  Sample $n$ tasks $\{\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)\}_{i=1}^{i=n}$
  **for all** $\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)$ **do**
    $NLL_i \leftarrow \textsc{GaussNLL}(\boldsymbol{Y}^i; \boldsymbol{J}\boldsymbol{\mu},\ \boldsymbol{J}\boldsymbol{Q}^\top \mathrm{diag}(\boldsymbol{s}^2)\boldsymbol{Q}\boldsymbol{J}^\top + \boldsymbol{\Sigma}_\varepsilon)$ ▷ $\boldsymbol{J} = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)$
  **end for**
  Update $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$, $\boldsymbol{s}$ with $\nabla_{\boldsymbol{\theta}_0 \cup \boldsymbol{\mu} \cup \boldsymbol{s}} \sum_i NLL_i$
**end for**

**Sparse Gaussian neural processes.** Rochussen and Fortuin (2025) extend the sparse variational GP (Titsias, 2009) framework to the meta-learning setting by training an encoder function that predicts task-specific inducing points from context data, enabling rapid sparse GP inference on new tasks without per-task optimization. While both SGNP and our LUMA yield analytically tractable posterior predictive distributions, the two methods are fundamentally different: SGNP predicts inducing points that summarize the context data, whereas LUMA directly infers a posterior distribution over functions in the weight space of a linearized neural network. Another notable difference is that LUMA provides an analytically tractable prior predictive (equation 5), enabling principled task-level OoD detection. In contrast, SGNP does not compute the likelihood of context data; thus, it's not suitable for OoD detection.

**Meta-learning vs. fine-tuning approaches** Foundation models are large-scale pre-trained neural networks trained on vast amounts of unlabeled data from diverse domains (Devlin et al., 2019; Brown et al., 2020; Bommasani et al., 2021). These models serve as general-purpose backbones for a wide range of downstream tasks and have significantly influenced research in few-shot learning.

A widely used approach for adapting foundation models is fine-tuning, where the model's parameters are further updated on task-specific data (Howard and Ruder, 2018; Houlsby et al., 2019; Lester et al., 2021). Although this method is straightforward and often effective, it can be computationally expensive and prone to performance degradation, particularly when only a small number of labeled examples are available. Alternatively, in-context learning provides examples of the desired task within the models input prompt, enabling generalization to new queries without updating model parameters (Brown et al., 2020). However, meta-learning offers a more principled framework for adapting to families of related tasks, allowing for rapid generalization even in low-data regimes. Additionally, meta-learning approaches tend to be more robust to domain shifts, as the meta-training phase involves adaptation to diverse tasks and conditions, explicitly preparing the model to generalize effectively.

## B A tractable way of finding the perturbation directions in weight space that impact the most the predictions of an entire dataset

Deep neural networks have a large number of parameters, making the feature map $\phi_{\boldsymbol{\theta}_0}$ high-dimensional. However, recent work has shown that only a small subspace of the weight space is impactful. For example, to perform continual learning, Farajtabar et al. (2020) leverage the fact that it is sufficient to update the parameters or-

thogonally to a few directions only to avoid catastrophic forgetting. Sagun et al. (2017) have shown that the Hessian of a deep neural network can be summarized in a few number of directions, due to rapid spectral decay. This encourages finding a method to extract these meaningful directions of the weight space.

### B.1 Link with the Fisher Information Matrix

We define these main directions as the ones that have the most impact on the predictions of a whole dataset. To find them, we first find a way to quantify the influence of an infinitesimal weight perturbation. Using the second-order approximation of that quantity, we then describe in the deep-learning context a tractable way to find the directions.

**Setting** We take the same setting as Section Background, and we describe a method to quantify the influence of a parameter perturbation $\tilde{\boldsymbol{\theta}}$ on the predictions of a dataset of tasks $\mathcal{D}$. To do so, we leverage a probabilistic interpretation of the model: we assume a Gaussian pdf over the observations for given inputs and parameters $p_{\boldsymbol{\theta}}(\boldsymbol{Y}|\boldsymbol{X}) \sim \mathcal{N}(g(\boldsymbol{\theta}, \boldsymbol{X}),\ \boldsymbol{\Sigma}_\varepsilon)$, where the covariance of the noise $\boldsymbol{\Sigma}_\varepsilon$ is diagonal $\boldsymbol{\Sigma}_\varepsilon = \sigma_\varepsilon^2 \boldsymbol{I}$ (just as in Section Background).

**Perturbation of the prediction of a batch of inputs.** Before quantifying the influence of the perturbation on the predictions of the whole task dataset, we do it for the prediction of a batch of inputs $g(\boldsymbol{\theta}_0, \boldsymbol{X})$.

We borrow the method from Sharma et al. (2021): we quantify the influence of a parameter perturbation by computing the Kullback-Leibler divergence between $p_{\boldsymbol{\theta}_0}(\boldsymbol{Y}|\boldsymbol{X})$ and $p_{\boldsymbol{\theta}_0+\tilde{\boldsymbol{\theta}}}(\boldsymbol{Y}|\boldsymbol{X})$. The expansion is:

$$\delta(\boldsymbol{\theta}_0, \boldsymbol{X})(\tilde{\boldsymbol{\theta}}) := D_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}(\boldsymbol{Y}|\boldsymbol{X}) \| p_{\boldsymbol{\theta}_0+\tilde{\boldsymbol{\theta}}}(\boldsymbol{Y}|\boldsymbol{X})) \approx \tilde{\boldsymbol{\theta}}^\top \boldsymbol{F}(\boldsymbol{\theta}_0, \boldsymbol{X}) \tilde{\boldsymbol{\theta}} + o(\|\tilde{\boldsymbol{\theta}}\|^2) \tag{7}$$

where $\boldsymbol{F}(\boldsymbol{\theta}_0, \boldsymbol{X}) := \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X})^\top \boldsymbol{\Sigma}_\varepsilon^{-1} \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}) = \sigma_\varepsilon^{-2} \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X})^\top \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}) \in \mathbb{R}^{P \times P}$ is the empirical Fisher Information Matrix (FIM) of the batch of inputs $\boldsymbol{X}$, computed on the parameters $\boldsymbol{\theta}_0$.

**Generalization: perturbation of the prediction of a dataset.** Now we can define the influence of a parameter perturbation on the whole training dataset $\mathcal{D}$, by generalizing the previous definition:

$$\delta(\boldsymbol{\theta}_0, \mathcal{D})(\tilde{\boldsymbol{\theta}}) := \frac{1}{N} \sum_{i=1}^{N} \delta(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)(\tilde{\boldsymbol{\theta}})$$

Using equation 7, this quantity verifies:

$$\delta(\boldsymbol{\theta}_0, \mathcal{D})(\tilde{\boldsymbol{\theta}}) \approx \tilde{\boldsymbol{\theta}}^\top \left( \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{F}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i) \right) \tilde{\boldsymbol{\theta}} + o(\|\tilde{\boldsymbol{\theta}}\|^2) \tag{8}$$

which gives a natural definition for the FIM of the whole dataset by analogy with equation 7:

$$\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D}) := \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{F}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i) = \frac{1}{N\sigma_\varepsilon^2} \sum_{i=1}^{N} \boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)^\top \boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i) \in \mathbb{R}^{P \times P}$$

The expansion in equation 8 shows that the FIM of the dataset is a second-order approximation describing the influence of a parameter perturbation over the entire dataset. In particular, the eigenvectors of $\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D})$ with the highest eigenvalues are the directions that impact the most the predictions.

### B.2 Computing the top eigenvectors of the Fisher Information Matrix of the dataset in a deep learning context

Naively computing the top eigenspace of $\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D})$ requires processing a $P \times P$ matrix, which is intractable in the deep-learning context (where $P$ can surpass $10^6$). Instead, we decide to use the method used by Sharma et al. (2021), which leverages a low-rank-approximation-based technique (namely matrix sketching) developed by Tropp et al. (2017).

**Sketching the FIM of the dataset** The key idea behind this technique is to build two small random sketches of the FIM, $(\boldsymbol{Y}, \boldsymbol{W}) := \mathcal{S}\left(\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D})\right)$, that with high probability contain enough information to reconstruct the top $r$ eigenvectors of $\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D})$. The linearity of $\mathcal{S}$ simplifies the sketching process by breaking down the computation into individual sketches:

$$(\boldsymbol{Y}, \boldsymbol{W}) := \mathcal{S}(\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D})) = \frac{1}{N\sigma_\varepsilon^2} \sum_{i=1}^{N} \mathcal{S}\left(\boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)^\top \boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)\right) =: \frac{1}{N\sigma_\varepsilon^2} \sum_{i=1}^{N} (\boldsymbol{Y}^i, \boldsymbol{W}^i)$$

In particular, the sketch $(\boldsymbol{Y}, \boldsymbol{W})$ can be updated in-place and does not require to store all the individual sketches. Given a sketch budget $k + l$ (Sharma et al. (2021) recommends choosing $k := 2r + 1$ and $l := 4r + 3$) and two random normal matrices $\boldsymbol{\Omega} \in \mathbb{R}^{k \times P}$ and $\boldsymbol{\Psi} \in \mathbb{R}^{l \times P}$, the random individual sketches $(\boldsymbol{W}^i, \boldsymbol{Y}^i)$ are defined as:

$$\begin{cases} \boldsymbol{Y}^i & := & ((\boldsymbol{\Omega}\boldsymbol{J}_i^\top)\boldsymbol{J}_i)^\top & \in \mathbb{R}^{P \times k} \\ \boldsymbol{W}^i & := & (\boldsymbol{\Psi}\boldsymbol{J}_i^\top)\boldsymbol{J}_i & \in \mathbb{R}^{l \times P} \end{cases}$$

where $\boldsymbol{J}_i := \boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)$.

**Sketch-based computation of the top eigenspace** Once the sketches are computed, the function FixedRankSymApprox by Tropp et al. (2017) computes the first eigenvectors and eigenvalues of the FIM of the dataset. Overall, the memory footprint to find the sketches and the top eigenspace is $O(P(r + D_y M))$, where $r$ is the number of queried eigenvectors and $M$ is the size of $\widetilde{\boldsymbol{X}}^i$. As long as $r \ll P$, this computation is tractable, given that $D_y M \ll P$ is usual deep learning contexts. Algorithm 4 summarizes the process that yields the FIM-based projections via sketching. We drop the scaling coefficient $\sigma_\varepsilon^{-2}$ as it doesn't affect the computation, given that we only want orthogonal eigenvectors and eigenvalues.

**Algorithm 4** Computing the FIM-based projections

**Require:** $r$ (desired size of the subspace)
$k \leftarrow 2r + 1$
$l \leftarrow 4r + 3$
Draw $\boldsymbol{\Omega} \in \mathbb{R}^{k \times P}, \boldsymbol{\Psi} \in \mathbb{R}^{l \times P}$, two random normal matrices
Initialize $\boldsymbol{Y} = 0 \in \mathbb{R}^{P \times k}, \boldsymbol{W} = 0 \in \mathbb{R}^{l \times P}$
**for all** training task $\mathcal{T}^i$ **do**
    $\boldsymbol{J}_i \leftarrow \boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)$
    $\boldsymbol{Y} \leftarrow \boldsymbol{Y} + 1/N((\boldsymbol{\Omega}\boldsymbol{J}_i^\top)\boldsymbol{J}_i)^\top$
    $\boldsymbol{W} \leftarrow \boldsymbol{W} + 1/N(\boldsymbol{\Psi}\boldsymbol{J}_i^\top)\boldsymbol{J}_i$
**end for**

### B.3 Computational Complexity Analysis

The computational complexity of our method, LUMA, scales **linearly with the number of model parameters ($P$)**, which is comparable to the scaling of MAML. The primary additional costs arise from Gaussian Process (GP) matrix operations, which remain manageable in typical meta-learning settings. Below, we provide a detailed breakdown of the computational costs for both the training and inference phases.

To recap, the variables used in this analysis are defined as:

- $P$: The total number of parameters in the neural network.
- $K$: The number of context points in a given task.
- $N_y$: The output dimensionality of the model.
- $r$: The rank of the low-rank covariance approximation, where $r \ll P$.

#### Training Cost

The training cost is dominated by the computation of the Jacobian and the GP prior covariance. The cost varies depending on the chosen covariance parameterization.

- **LUMA with Low-Rank Covariance (LUMA-F and -R):** The primary operations are the GP prior mean computation ($O(N_yKP)$) and the prior covariance computation. The total cost is dominated by the Jacobian-vector products and matrix multiplications involving the rank $r$ factor.

$$\text{Training Cost (Low-Rank)} = O(N_yKrP)$$

- **LUMA with Full Covariance:** Without low-rank approximations, the training cost scales quadratically with the number of parameters $P$ due to the manipulation of the full $P \times P$ covariance matrix.

$$\text{Training Cost (Full)} = O(N_yKP^2)$$

This comparison highlights the significant efficiency gains from our proposed low-rank parameterizations, making the framework scalable to larger networks.

**Inference Cost**

During inference, the cost is determined by the posterior predictive computation, which involves kernel matrix calculations and a matrix inversion. The inference cost is the same for all variants of LUMA.

The key computational steps are:

1. **Kernel Matrix Computation:** Calculating the kernel matrices $k(x, X)$ and $k(X, X)$ requires Jacobian-vector products, resulting in a cost of $O(N_y^2K^2P)$.
2. **GP Inverse Computation:** The inversion of the $N_yK \times N_yK$ kernel matrix has a cost of $O((N_yK)^3)$.

Combining these steps, the total inference cost is:

$$\text{Inference Cost} = O(N_y^2K^2P + (N_yK)^3)$$

In typical meta-learning scenarios, the number of context points $K$ and the output dimension $N_y$ are small, making the $O((N_yK)^3)$ term manageable and often negligible compared to the term that scales with the model size $P$. Furthermore, the kernel computations involving the context set $X$ can be pre-computed and cached, accelerating predictions for multiple query points.

## C LUMA as a generalization of ALPaCA

**Restraining the linearization to the last layer** Remember the linear regression of equation 3, that we obtained by linearizing the network with all its layers. Let's separate the parameters of the network $\boldsymbol{\theta}$ into two: the parameters of all the layers but the last one $\boldsymbol{\lambda}$, and the parameters of the last layer $\boldsymbol{\rho}$: $\boldsymbol{\theta} = \boldsymbol{\lambda} \cup \boldsymbol{\rho}$.

We assume that the last layer is dense with biases: we will note $\boldsymbol{\rho}^w$ the weight matrix and $\boldsymbol{\rho}^b$ the biases of this last layer. $N_\psi$ will stand as the dimension of the activations right before the last layer: in particular, $\boldsymbol{\rho}^w \in \mathbb{R}^{D_y \times N_\psi}$ and $\boldsymbol{\rho}^b \in \mathbb{R}^{D_y}$. $P'$ will stand as the size of $\boldsymbol{\rho}$: in our case, $P' = N_\psi \times D_y + D_y$. We implicitly vectorize $\boldsymbol{\rho}$, such that:

$$\boldsymbol{\rho} = \begin{pmatrix} \operatorname{vec} \boldsymbol{\rho}^w \\ \boldsymbol{\rho}^b \end{pmatrix} \in \mathbb{R}^{N_\psi D_y + D_y} = \mathbb{R}^{P'}$$

We now restrain the linear regression to the last layer as follows:

$$\boldsymbol{y} = \boldsymbol{J}'(\boldsymbol{\rho}_0, \psi_{\boldsymbol{\lambda}}(\boldsymbol{X}))(\boldsymbol{\rho} - \boldsymbol{\rho}_0) + \varepsilon \tag{9}$$

where:

- $\psi_{\boldsymbol{\lambda}}(\cdot) : \mathbb{R}^{D_x \times K} \to \mathbb{R}^{N_\psi \times K}$ stands for the function that maps the inputs and the activations right before the last layer;

- $\boldsymbol{J}'(\cdot,\cdot) : \mathbb{R}^{P'} \times \mathbb{R}^{N_\psi \times K} \to \mathbb{R}^{D_y K \times P'}$ stands for the jacobian of the last layer *with respect to the parameters* $\boldsymbol{\rho}$. We can write the jacobian in closed-form due to the linearity of the last layer:

$$\boldsymbol{J}'(\boldsymbol{\rho}_0, \psi_{\boldsymbol{\lambda}}(\boldsymbol{X})) = \begin{pmatrix} \psi_{\boldsymbol{\lambda}}(\boldsymbol{X})^\top \otimes \boldsymbol{I}_{D_y} & \boldsymbol{I}_{D_y} \end{pmatrix} \in \mathbb{R}^{D_y K \times P'}$$

  Note that $\boldsymbol{J}'(\boldsymbol{\rho}_0, \psi_{\boldsymbol{\lambda}}(x))$ does not depend on the linearization point $\boldsymbol{\rho}_0$ (could be expected, given the linearity of the last layer).
- $\boldsymbol{\rho} - \boldsymbol{\rho}_0 \in \mathbb{R}^{P'}$ is the *correction* to the last parameters. We will note the correction to the weights as $\hat{\boldsymbol{\rho}}^w := \boldsymbol{\rho}^w - \boldsymbol{\rho}_0^w$ and the correction to the bias as $\hat{\boldsymbol{\rho}}^b := \boldsymbol{\rho}^b - \boldsymbol{\rho}_0^b$.

Using Kronecker's product identities, the linear regression in equation 9 can be rewritten:

$$\boldsymbol{y} = \hat{\boldsymbol{\rho}}^w \psi_{\boldsymbol{\lambda}}(\boldsymbol{X}) + \hat{\boldsymbol{\rho}}^b + \varepsilon \tag{10}$$

Also, as a side note, another way of getting this linear regression (equation 10) is to rewrite the initial linearization of Section Background, but with respect to $\boldsymbol{\rho}$ only:

$$\begin{aligned} f(\boldsymbol{\lambda} \cup \boldsymbol{\rho}, \boldsymbol{x}_t) - f(\boldsymbol{\lambda} \cup \boldsymbol{\rho}_0, \boldsymbol{x}_t) &= \boldsymbol{\rho}^w \psi_{\boldsymbol{\lambda}}(x) + \boldsymbol{\rho}^b - (\boldsymbol{\rho}_0^w \psi_{\boldsymbol{\lambda}}(x) + \boldsymbol{\rho}_0^b) \\ &= (\boldsymbol{\rho}^w - \boldsymbol{\rho}_0^w)\psi_{\boldsymbol{\lambda}}(x) + (\boldsymbol{\rho}^b - \boldsymbol{\rho}_0^b) \\ &= \hat{\boldsymbol{\rho}}^w \psi_{\boldsymbol{\lambda}}(x) + \hat{\boldsymbol{\rho}}^b \end{aligned}$$

Doing it this way has the benefit to show that the linearization is exact when restricted to the last layer. For non-linear neural networks, the linearization is always an approximation.

**Adapting LUMA to the last layer** Just like what we did in the general case, we use the GP theory to make Bayesian inference on this linear regression. The new parameters of the GP $\tilde{p}_\xi(f)$ are now $\xi = (\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the parameters of the Gaussian prior over $\hat{\boldsymbol{\rho}}$. Note how $\boldsymbol{\rho}_0$ have disappeared from $\xi$: contrary to the general case, the linearization point is not optimized, as it does not impact the computation. Also note that $\boldsymbol{\lambda}$ has replaced $\boldsymbol{\rho}_0$, as it parameterizes the feature map.

**Comparison with ALPaCA** In the ALPaCA setting, the linear regression is:

$$\boldsymbol{y}_a = \hat{\boldsymbol{\rho}}^w \psi_{\boldsymbol{\lambda}}(\boldsymbol{X}) + \varepsilon \tag{11}$$

Note that $\hat{\boldsymbol{\rho}}^w$ still plays the same role in the linear regression, but is not any *correction* anymore. Subtracting 10 from 11 yields:

$$\boldsymbol{y} - \boldsymbol{y}_a = \hat{\boldsymbol{\rho}}^b$$

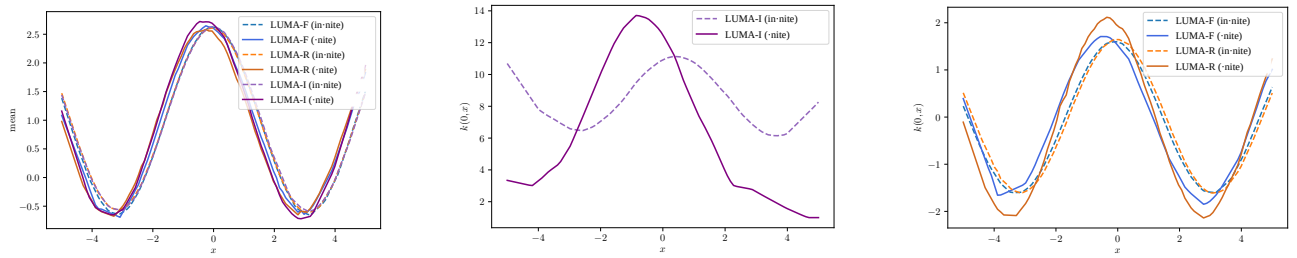
LUMA restricted to the last layer is closely related to ALPaCA, in that they both perform a linear regression with the same kernel: the only difference lies in the additional bias term that is not considered in ALPaCA. Thus, we can think of LUMA as a generalization of ALPaCA to all the layers of the network.

## D Comparison with Deep Kernel Learning

Table 3 summarizes the key representational differences between Deep Kernel Transfer (DKT) and our method.

| **Aspect** | **Meta Deep Kernel Learning** | **Ours** |
|---|---|---|
| **Kernel** | Over feature *outputs*: $k(h_\theta(x_1), h_\theta(x_2))$ | In *weight space* via NTK: $k_\Sigma(x_1, x_2) = J(x_1)^\top \Sigma J(x_2)$ |
| **Meta-learned** | Feature extractor weights $\theta$ | Prior distribution $(\mu, \Sigma)$ over network weights |
| **Uncertainty** | Input-level (distance in feature space) | Task-level (prior predictive over context) |

Table 3: Comparison of representational differences between DKT and LUMA.




(a) Mean $\mathbb{E}(f)$ (LUMA-I, LUMA-R, LUMA-F)

(b) Covariance $\text{cov}(f(0), f(x))$ (LUMA-I)

(c) Covariance $\text{cov}(f(0), f(x))$ (LUMA-R, LUMA-F)

Figure 4: Mean (4a) and covariance (4b, 4c) functions of $\tilde{p}_\xi(f)$ after different meta-trainings on the sine cluster (LUMA-I, LUMA-R and LUMA-F, with a finite or infinite dataset). Note the scale, and how LUMA-I has a less accurate covariance function that LUMA-R and LUMA-F.

# E Additional results

## E.1 Additional results (single-cluster case)

**Quality of the priors** To qualitatively analyze the prior after meta-training on the sines, we plot the mean functions and the covariance functions of the resulting GP, that is $\tilde{p}_\xi(f)$ (Figure 4). All the trainings yield a similar mean for $\tilde{p}_\xi(f)$ (that is a cosine with amplitude 1.5 and offset 1) (Figure 4a), which is close to the theoretical mean of $p(f)$ (a cosine with amplitude 2.5 and offset 1). The covariance function of LUMA-R and LUMA-F (Figure 4c) resembles what we would expect for $p(f)$ (e.g., periodicity, negative correlation between 0 and $\pi$, etc.), but it is not the case for LUMA-I (Figure 4b). This empirical analysis confirms the quantitative comparison in terms of OoD detection and prediction performance carried in Section Numerical Analysis.

**Examples of predictions** Figure 5 summarizes the predictions of the model meta-trained on sines, for a varying number of context inputs $K$, breaking down all the different cases of training.

## E.2 Additional results (multi-cluster case)

**Quality of the priors** Figure 6 and Figure 7 show the mean and covariance functions of the GP (when training with a single GP), and the mean and covariance functions of the GPs composing the mixture(when training with a mixture of GPs). We note that in the case of the mixture, both of the Gaussians composing the mixture have correctly captured the common features shared by each of the clusters (respectively the linear and the sine cluster). For instance, the mean of the line cluster is the zero function, which matches Figure 6b, and the correlation between $x = 1$ and the other inputs is correctly rendered for linear tasks (Figure 7b).

When learning with a single GP, the learnt mean and covariance do not match any of the two clusters. For example, the mean has an intermediate offset between the offset of the sine cluster and the line cluster. This empirical analysis comforts the conclusions of Section Numerical Analysis: the mixture model yields better results than the single GP case.

## E.3 LUMA yields effective predictions on large-scale vision problems

We consider a vision meta-learning problem from Gao et al. (2022), Shapenet1D, aiming to predict object orientations in space. In this problem, each task consists of a different object of which we want to predict the orientation. For each task, the context data consists of some images of the same object, but with different orientations; the query inputs are other images of the same object, with unknown orientations. Details on the problems and the datasets can be found in Appendix F.

We train a deep learning model on Shapenet1D, and we compare the performances on the test set between LUMA-I, LUMA-R and LUMA-F (Figure 9). More training and test details can be found in the Appendix J.

Both LUMA-I and LUMA-F yield better performances than MAML, achieving low angle errors. LUMA-R

however gives poor results, worse than that of MAML.

In terms of the trade-off of Section Methods, our conclusion from small networks does not scale up to deep models. Here, the loss of valuable features is perceptible (that is what happens with LUMA-R, when the randomness of the directions may drop such features), and not learning a rich prior over the weights is not burdensome (LUMA-I). However, LUMA-F plays a role of compromise, by learning the prior covariance while keeping the few important features of the jacobian, as it gives comparable results to LUMA-I.

# F Details on the regression problems

## F.1 Simple regression problems

Our simple regression problems are inspired by Vuorio et al. (2019)'s work. They consists of three clusters of different types of tasks, and varying offset. The first cluster consists of sines with constant frequency and offset, but with a varying amplitude and phase:

$$\{x \mapsto A\sin(x+\varphi)+1 | A \in [0.1, 5], \varphi \in [0, \pi]\}$$

The second cluster consists of lines with a varying slope and no offset:

$$\{x \mapsto ax | a \in [-1, 1]\}$$

The last cluster consists of quadratic functions, with a varying quadratic coefficient and phase, and with a constant offset:

$$\left\{x \mapsto a(x-\varphi)^2 + 0.5 | a \in [-0.2, 0.2], \varphi \in [-2, 2]\right\}$$

In all these clusters, we add an artificial Gaussian noise on the context observations $\mathcal{N}(0,\ 0.05)$. The query datapoints remain noiseless, to remain coherent with the assumption from Section Background borrowed from Rasmussen and Williams (2005).

## F.2 Vision problem

We consider the meta-learning vision, regression problem recently created by Gao et al. (2022) (namely Shapenet1D), that consists in estimating objects orientation from images. The objects have a vertical angular degree of freedom, and Figure 8 shows an example of such objects in different positions.

We use the same training and test datasets as Gao et al. (2022), with no kind of augmentation (in particular, no artificial noise on the ground-truth angles). In particular, we use the same intra-category (IC) evaluation dataset (that is, objects from the same categories as the objects used for training) and cross-category (CC) evaluation dataset (that is, objects from different categories as the ones used for training).

# G Full Results with Confidence Intervals

We report the full results with 95% confidence intervals for all experiments.

| $K$ | **Unimodal (Infinite)** **5 / 10** | **Unimodal (Finite)** **5 / 10** |
|---|---|---|
| CNP | 0.0485±0.0077 / 0.0189±0.0030 | 1.9816±0.1902 / 1.5777±0.1806 |
| TNP-D | 0.1324±0.1045 / 0.0186±0.0014 | 2.5239±0.2130 / 1.6384±0.1461 |
| DKT | 3.2730±0.4264 / 0.2122±0.0524 | 3.4938±0.3593 / 0.3008±0.1193 |
| **Ours** | **0.0026±0.0002 / 0.0015±0.0001** | **0.0204±0.0018 / 0.0105±0.0010** |

Table 4: Unimodal regression MSE with 95% CIs. Mean ± CI reported for $K$=5/10.

| | Multimodal | |
|---|---|---|
| $K$ | **5** | **10** |
| CNP | 0.1402 ± 0.0490 | 0.0311 ± 0.0083 |
| TNP-D | 0.1253 ± 0.0517 | 0.0196 ± 0.0020 |
| DKT | 2.4779 ± 0.3681 | 0.1760 ± 0.0525 |
| Ours (Single) | 0.0454 ± 0.0084 | 0.0027 ± 0.0003 |
| **Ours (Mixt)** | **0.0024 ± 0.0002** | **0.0012 ± 0.0001** |

Table 5: Multimodal regression MSE with 95% CIs. LUMA (Mixture) achieves the lowest error.

| | Vision Regression Tasks (Angular Error °) | | |
|---|---|---|---|
| $K$ | **5** | **10** | **15** |
| CNP | 22.31 ± 1.930 | 19.70 ± 1.250 | 21.00 ± 2.310 |
| TNP-D | 88.74 ± 1.830 | 89.75 ± 1.760 | 89.96 ± 1.660 |
| DKT | 21.32 ± 1.840 | **3.920 ± 0.4600** | **1.790 ± 0.1500** |
| **Ours** | **18.94 ± 3.280** | 7.684 ± 1.150 | 5.157 ± 0.9000 |

Table 6: Vision regression (angular error) with 95% CIs. DKT excels with sufficient context ($K \geq 10$), while LUMA is relatively more stable in the extreme low-data regime ($K = 5$).

## H Training and test details for the single cluster case

### H.1 Training details of LUMA (single cluster case)

The model is a neural network with 2 hidden layers, 40 neurons each, with a ReLU non-linearity. In our single-cluster experiment, the cluster is the sine cluster (see Appendix F).

In the case where there are an infinite number of available sine tasks during training (*ie* $N = +\infty$), the training is performed with $n = 24$ tasks per epoch, and at each epoch the context inputs are randomly drawn from $[-5, 5]$ (which means that $M = +\infty$). In the case where we restrict the available tasks to a finite number, we randomly choose $N = 10$ tasks and $M = 50$ context datapoints per task before training (they are shared among all the "finite" trainings) and perform the trainings with $n = 6$ tasks per epoch.

For all the trainings, the number of context inputs seen during training is $K = 10$.

For all trainings, we train LUMA on $60,000$ epochs. When dealing with the LUMA-F case, we allow half of the epochs $(30,000)$ to the training *before* finding the intermediate $\boldsymbol{\theta}_0$ and $\boldsymbol{\mu}$ (see Algorithm 1), and the other half *after*.

In the LUMA-F case with infinite available tasks, a finite number of tasks is needed to compute the FIM: thus we build an artificial finite dataset of $N = 100$ and $M = P$ (arbitrary, but chosen so that $\boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)\boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)^\top$ gets a chance to be full-rank *ie* contain as much information as possible), that is used only for that computation.

In the LUMA-F and the LUMA-R cases, the subspace size is $r = 10$.

For all trainings, the meta-optimizer is Adam with an initial learning rate of 0.001. The noise $\sigma_\varepsilon = 0.05$, equal to the noise added to the context data. We compute the NLL using Rasmussen and Williams (2005)'s implementation.

### H.2 Training details of MAML (single cluster case)

We train MAML baselines to compare our results with that of MAML. A large part of these hyperparameters is directly inspired from Finn et al. (2017)'s work.

The model is a neural network with 2 hidden layers, 40 neurons each, with a ReLU non-linearity. In our single-cluster experiment, the cluster is the sine cluster (see Appendix F).

In the case where there are an infinite number of available sine tasks during training (*ie* $N = +\infty$), the training is performed with $n = 24$ tasks per epoch, and at each epoch the context and query inputs are randomly drawn from $[-5, 5]$. In the case where we restrict the available tasks to a finite number, we randomly choose $N = 10$

tasks and $M = 50$ datapoints per task (used for both context and query batches) before training (they are shared among all the "finite" trainings) and perform the trainings with $n = 6$ tasks per epoch.

For all trainings, the number of context datapoints is $K = 10$, and the number of query datapoints is $L = 10$. We meta-train for $70,000$ epochs.

For all trainings, the meta-optimizer is Adam with an initial learning rate of 0.001. The inner learning-rate is kept constant, at 0.001. The number of inner updates is 5 during training, and 10 at test time.

### H.3 Test details (single-cluster case)

For the OoD detection evaluation, we plot the AUC as a function of the number of the context inputs $K$. The AUC is computed using the NLL of the context inputs with respect to $\tilde{p}_\xi(f)$ (our uncertainty metric). The true-positives are the OoD tasks (lines and quadratic tasks) flagged as such; the false-positives are the in-distribution tasks (sines) flagged as OoD.

For the predictions, we plot the average and 95% CI of the MSE on 1,000 tasks (100 queried inputs each). In the LUMA-R case, we also compute the average and 95% CI on 5 different random projections trainings.

## I Training and test details for the multi cluster case

### I.1 Training details of LUMA (multi-cluster case)

The model is a neural network with 2 hidden layers, 40 neurons each, with a ReLU non-linearity. In our multi-cluster experiment, $\alpha = 2$: the clusters consist of the sine cluster and the linear cluster (see Appendix F).

For all the trainings, the training is performed with $n = 24$ tasks per epoch (with an infinite number of available sine tasks and linear tasks during training *ie* $N = +\infty$), and at each epoch the context inputs are randomly drawn from $[-5, 5]$ (which means that $M = +\infty$). In accordance with our equal probability assumption from Section Methods, at each epoch $n/2 = 12$ tasks come from the sine cluster and $n/2 = 12$ tasks come from the linear cluster.

For all trainings, the number of context inputs seen during training is $K = 10$.

For all trainings, we train LUMA algorithm on $60,000$ epochs: because we deal with the LUMA-F case, we allow half of the epochs ($30,000$) to the training *before* finding the intermediate $\boldsymbol{\theta}_0$ and $\{\boldsymbol{\mu}_j\}_{j=1}^{j=\alpha}$ (see Algorithm 1), and the other half *after*.

In the LUMA-F case, a finite number of tasks is needed to compute the FIM: thus we build an artificial finite dataset of $N = 100$ and $M = P$ (arbitrary, but chosen so that $\boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)\boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)^\top$ gets a chance to be full-rank *ie* contain as much information as possible), that is used only for that computation.

For all trainings, the subspace size is $r = 10$.

For all trainings, the meta-optimizer is Adam with an initial learning rate of 0.001. The noise $\sigma_\varepsilon = 0.05$, equal to the noise added to the context data. We compute the NLL using Rasmussen and Williams (2005)'s implementation.

When training with MIXT, we make sure to initialize $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ randomly with $\mathcal{N}(\boldsymbol{0},\ 0.5\boldsymbol{I})$, so that the meta-learning can effectively differentiate the two clusters. Also, in the MIXT case, the mean $\boldsymbol{\mu}$ is unique *before* computing the FIM. Once we have found the projection directions, we initialize $(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2)$ with the intermediate $\boldsymbol{\mu}$, thus yielding two Gaussians.

### I.2 Training details of MAML (multi-cluster case)

We reimplement and train a MAML baseline to compare our results with that of MAML. A large part of these hyperparameters is directly inspired from Finn et al. (2017)'s work.

The model is a neural network with 2 hidden layers, 40 neurons each, with a ReLU non-linearity. In our multi-cluster experiment, $\alpha = 2$: the clusters consist of the sine cluster and the linear cluster (see Appendix F).

The training is performed with $n = 24$ tasks per epoch (with an infinite number of available sine and linear tasks

during training *ie* $N = +\infty$) and at each epoch the context and query inputs are randomly drawn from $[-5, 5]$. In accordance with our equal probability assumption from Section Methods, at each epoch $n/2 = 12$ tasks come from the sine cluster and $n/2 = 12$ tasks come from the linear cluster.

For all trainings, the number of context datapoints is $K = 10$, and the number of query datapoints is $L = 10$. We meta-train for $70,000$ epochs.

For all trainings, the meta-optimizer is Adam with an initial learning rate of 0.001. The inner learning-rate is kept constant, at 0.001. The number of inner updates is 5 during training, and 10 at test time.

### I.3 Training details of MMAML (multi-cluster case)

We train a MMAML by using Vuorio et al. (2019)'s code, using the best setting mentioned in their paper (FiLM). We adapt their code to train it on our sine and linear clusters: we add an offset to their sine cluster (via their parameter `bias`), change the phase from $\sin(x - \varphi)$ to $\sin(x + \varphi)$ and specify via the arguments the slope range ($[-1, 1]$) and the y-intercept range ($[0, 0]$, because it does not vary in our case) of the line tasks. We also change the number of context inputs that we set to $K = 10$, to remain coherent with the rest of the trainings. The rest of the hyperparameters are kept identical to the command specified in the repository of MMAML. Finally, at test time, we make the query ground-truths noiseless, to remain coherent with the rest of the test conditions.

### I.4 Test details (multi-cluster case)

For the OoD detection evaluation, we plot the AUC as a function of the number of context inputs $K$. The AUC is computed using the NLL of the context inputs with respect to $\tilde{p}_\xi(f)$ (our uncertainty metric). The true-positives are the OoD tasks (quadratic tasks) flagged as such; the false-positives are the in-distribution tasks (lines and sines) flagged as OoD.

For the predictions, we plot the average and 95% CI of the MSE on 1,000 tasks (100 queried inputs each): half of them are sines and half of them are lines.

## J Training and test details for the vision problem

The model is a deep neural network, identical to the one used by Gao et al. (2022) except for the last layer: instead of doing a one-dimensional regression (where the output stands for an angle prediction), we perform a two-dimensional regression (where the output stands for a cosine and sine prediction). This choice is motivated by the fact that the Gaussian noise assumed in Section Background cannot capture the complexity of an angle error (e.g., predicting 361° should yield a low error when compared to the ground-truth angle 0°), but better renders the MSE that can be applied on cosine and sine.

### J.1 Training details of LUMA (vision case)

For all the trainings, there are $n = 10$ tasks per epoch and $K = 15$ context inputs per task. When dealing with the LUMA-F case, we allow half of the epochs ($5,000$) to the training *before* finding the intermediate $\boldsymbol{\theta}_0$ and $\boldsymbol{\mu}$ (see Algorithm 1), and the other half after.

In the LUMA-F and LUMA-R cases, the subspace size is $r = 100$.

For all the trainings, the meta-optimizer is Adam with an initial learning rate of 0.001. The noise is $\sigma_\varepsilon = 0.01$. We compute the NLL using Rasmussen and Williams (2005)'s implementation.

### J.2 Training details of MAML (vision case)

We train a MAML baseline to compare our results with that of MAML. A large part of these hyperparameters is directly inspired from Gao et al. (2022).

The training is performed with $n = 10$ tasks per epoch. The number of context datapoints is $K = 15$, and the number of query datapoints is $L = 10$. We meta-train for $50,000$ epochs.

For all trainings, the meta-optimizer is Adam with an initial learning rate of 0.0005. The inner learning-rate is kept constant, at 0.002. The number of inner updates is 5 during training, and 20 at test time.

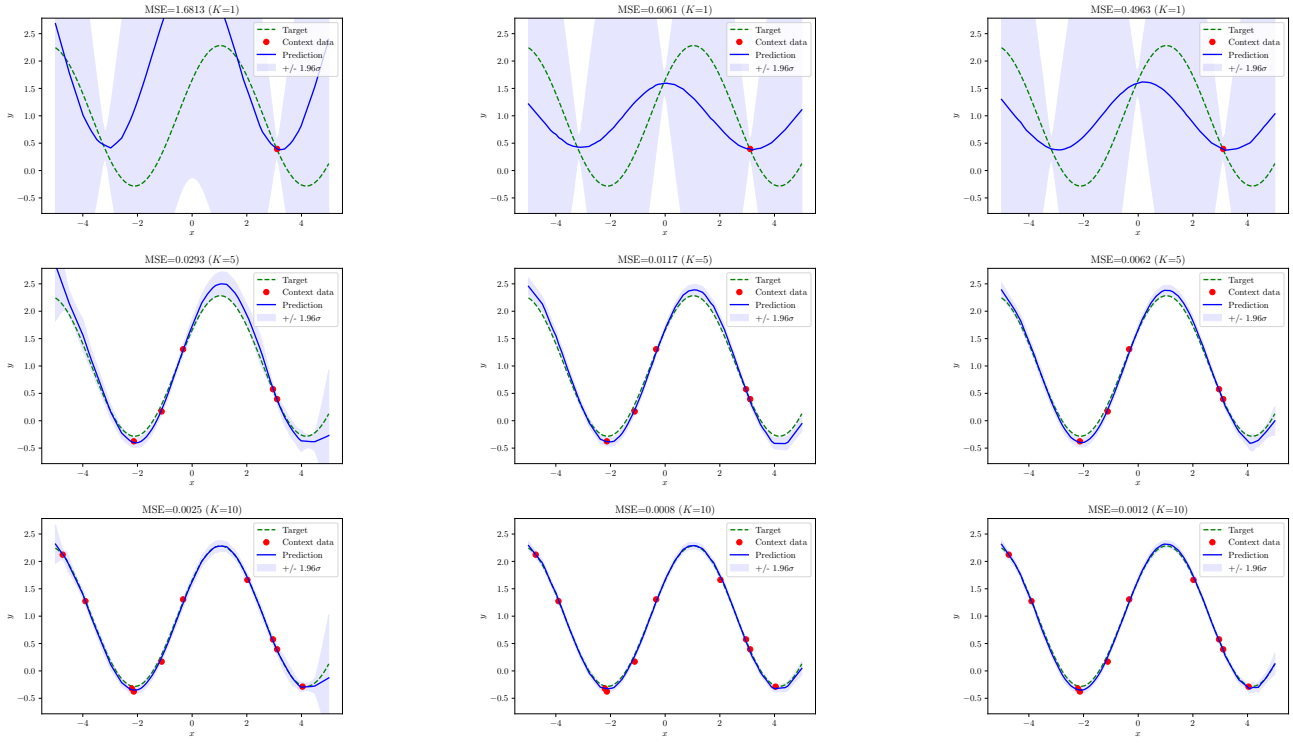
### J.3 Test details (vision problem)

At test time, we wrap our model with the arctan function to convert the predictions to angles. Then, we compare the angle predictions with the ground-truth angles. To do so, we use the following error from Gao et al. (2022) to compare two angles:

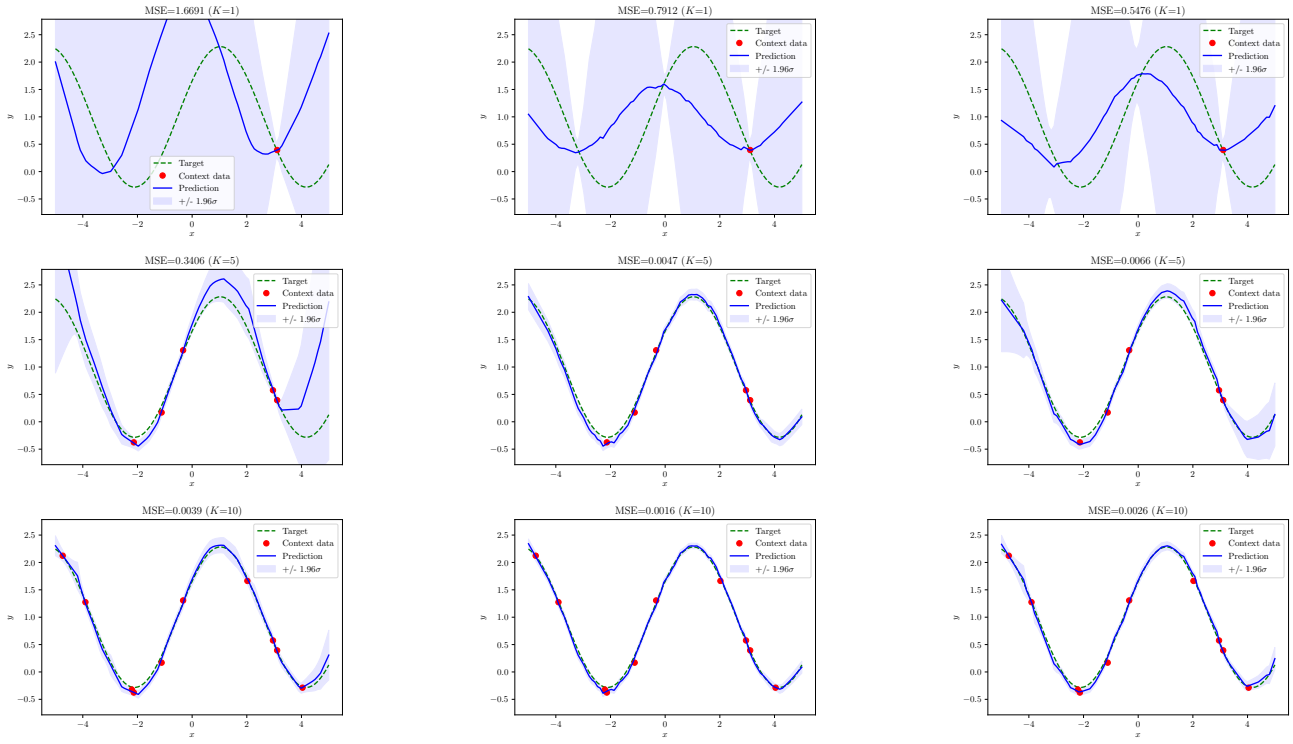
$$\mathcal{E}(\beta, \beta^*) = \min\{\mathcal{E}_{\beta^+,\beta^*}, \mathcal{E}_{\beta,\beta^*}, \mathcal{E}_{\beta^-,\beta^*}\}$$

where $\mathcal{E}_{\beta^\pm,\beta^*} = |y \pm 360 - y^*|$ and $\mathcal{E}_{\beta,\beta^*} = |y - y^*|$.

When plotting the performance (Figure 9), we plot the average and 95% CI on 100 tasks (15 queried inputs each). For LUMA-R, the average and 95% CI are computed on 5 different random projection trainings.

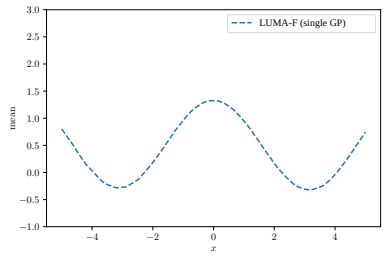



(a) LUMA-I (inf) (b) LUMA-R (inf) (c) LUMA-F (inf)

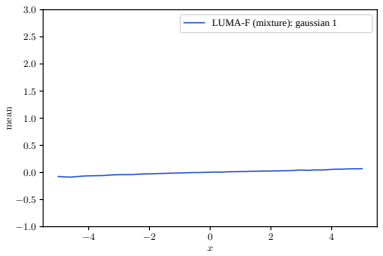



(d) LUMA-I (fin) (e) LUMA-R (fin) (f) LUMA-F (fin)

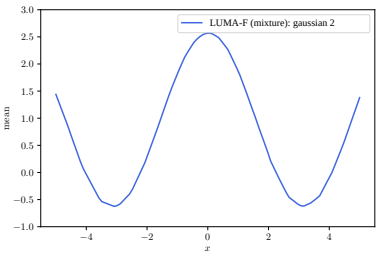
Figure 5: Example of predictions for a varying number of context inputs $K$, after different meta-trainings on the sine cluster (LUMA-I, LUMA-R and LUMA-F, in the infinite and finite case). Standard deviation is from the posterior predictive distribution. Note how LUMA-R and LUMA-F perform better than LUMA-I when it comes to reconstructing the sine with a smaller amount of context inputs.




(a) Mean function $\mathbb{E}(f)$ of the GP (LUMA-F with a single GP).

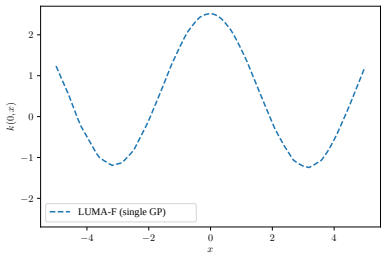



(b) Mean function $\mathbb{E}(f)$ of the first GP of the mixture (LUMA-F with a mixture of GPs).

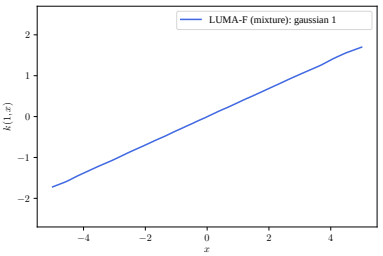



(c) Mean function $\mathbb{E}(f)$ of the second GP of the mixture (LUMA-F with a mixture of GPs).

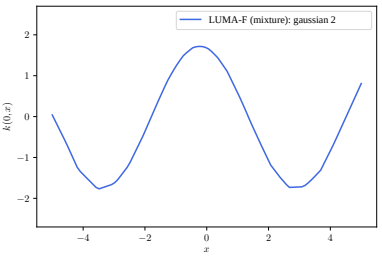
Figure 6: Mean functions of the GP (when learning with a GP) / the GPs composing the mixture (when learning a mixture of GPs), after training on both the sine and line cluster with LUMA-F. Note how the mean of the single GP is intermediate between the ones of the mixture.




(a) Covariance function $\text{cov}(f(0), f(x))$ of the GP (LUMA-F with a single GP).

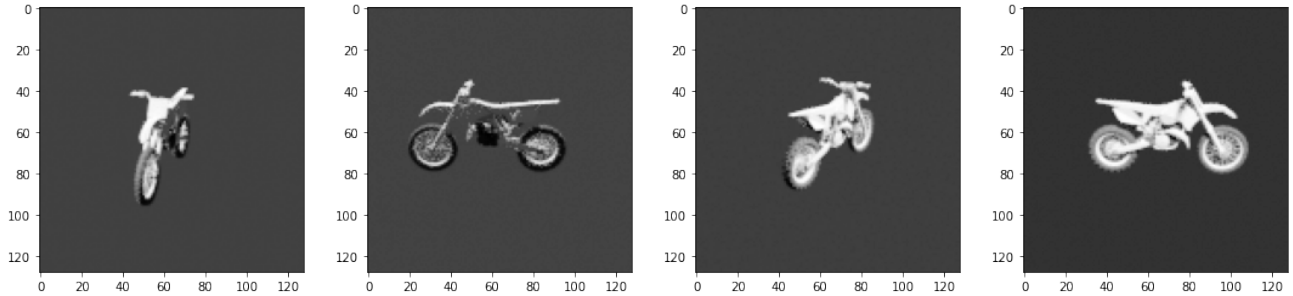



(b) Covariance function $\text{cov}(f(1), f(x))$ of the first GP of the mixture (LUMA-F with a mixture of GPs)

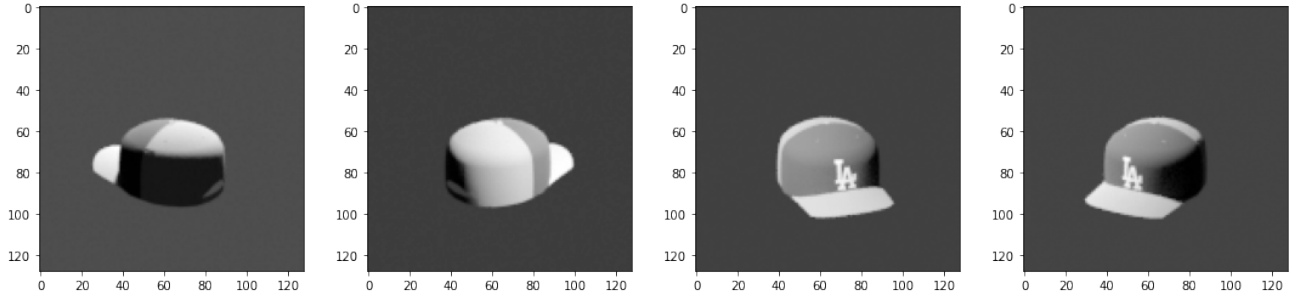



(c) Covariance function $\text{cov}(f(0), f(x))$ of the second GP of the mixture (LUMA-F with a mixture of GPs)

Figure 7: Covariance functions of the GP (when learning with a GP) / the GPs composing the mixture (when learning a mixture of GPs), after training on both the sine and line cluster with LUMA-F. Note how the covariance of the single GP is not accurate for any of the two clusters.

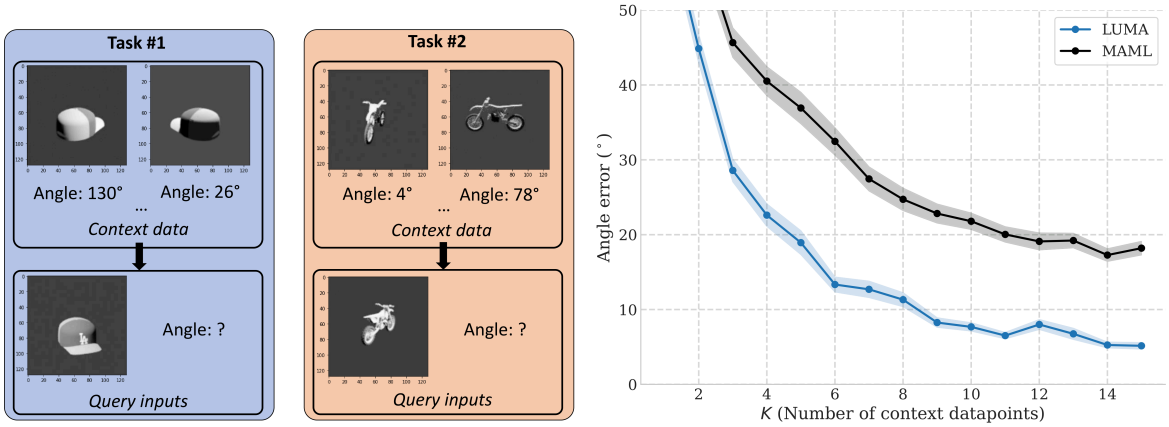


(a) Angle: 4° (c) Angle: 78° (e) Angle: 202° (g) Angle: 261°



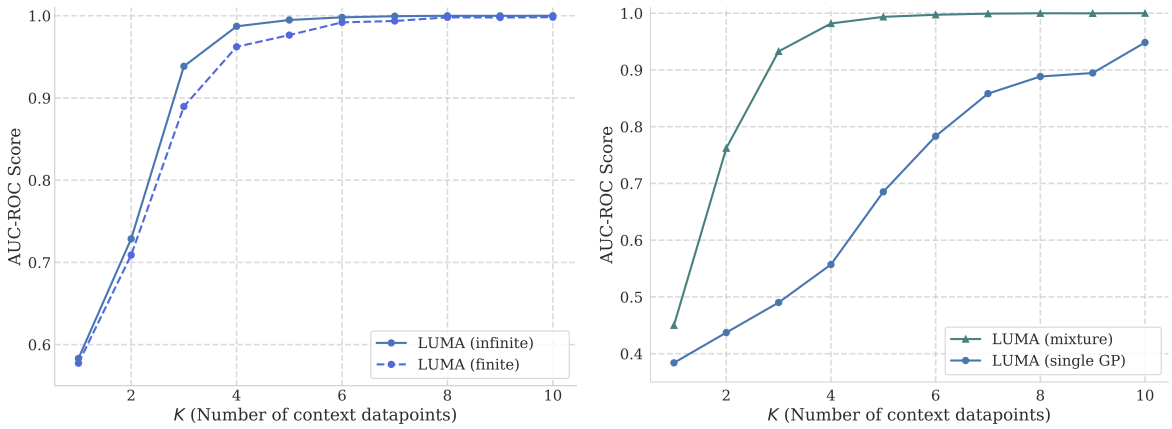
(b) Angle: 26° (d) Angle: 130° (f) Angle: 243° (h) Angle: 285°

Figure 8: Examples of images and ground-truth angles from the Shapenet1D dataset (Gao et al., 2022)




(a) Vision Task (b) Angle Error

Figure 9: (a) Graphical depiction of the vision tasks. (b) Angle error with varying $K$. LUMA provides more accurate predictions than the baseline.




(a) Unimodal (b) Multimodal

Figure 10: OoD detection performance: AUC-ROC score is evaluated with varying $K$. LUMA achieves high accuracy even with a limited number of context datapoints, in both unimodal and multimodal settings. In the multimodal setting, the mixture model further improves performance, as expected.