EFFICIENT ESTIMATION OF KERNEL SURROGATE MODELS FOR TASK ATTRIBUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern AI systems, including LLMs, are trained on diverse tasks (e.g., translation, code generation, math reasoning, text prediction) simultaneously. A key challenge is to quantify the influence of individual training tasks on target task performance — a problem we term *task attribution*. A natural solution is leave-one-out retraining, where each task is removed and the model is retrained to measure its effect on target performance. However, this approach is computationally prohibitive at scale. We address this challenge using surrogate models that approximate a target task's performance given any subset of training tasks. While prior work has explored linear surrogates, these only capture first-order (linear) effects and do not model nonlinear task interactions such as synergy, antagonism, or XOR-type relationships. We introduce *kernel surrogate models*, which better capture these nonlinear relationships. To make kernel estimation tractable, we develop a gradient-based procedure leveraging a first-order approximation of pretrained models, and empirically validate this to be accurate. Experiments across various domains (math reasoning in transformers, in-context learning, and multi-objective reinforcement learning) validate the effectiveness of kernel surrogate models. We find that kernel surrogate models demonstrate a 25% higher correlation with the leave-one-out ground truth than linear surrogate models and influence functions (among other baselines), establishing a more accurate and scalable solution for task attribution. Using kernel surrogate models for downstream task selection leads to 40% improvement in demonstration selection for in-context learning and multi-objective reinforcement learning benchmarks.

1 INTRODUCTION

Modern AI systems are trained to perform well on multiple tasks simultaneously, from machine translation to code generation, object recognition, and mathematical reasoning. The multi-task capability of AI models raises the question of how we can better interpret their model behaviors. In this paper, we study the problem of quantifying the influence of individual training tasks on model performance, a problem that we term as *task attribution*.

Besides being a fundamental question for model interpretability, this problem has natural interpretations in many applications. In multi-task learning, understanding the task relationships can inform the design of neural network architectures and loss reweighting strategies (Yang et al., 2025). In multi-group learning (Deng & Hsu, 2024), better modeling between different groups can reveal how training on different demographic groups affects model behavior. Similarly, for in-context learning (Garg et al., 2022; Zhang et al., 2025), one may consider the question of adding or removing one demonstration example in the prompt and ask how that affects the model predictions. Other examples include multi-objective reinforcement learning (Yu et al., 2020), where one might care about how competing reward signals influence learned policies.

The natural way to estimate the statistical influence of one training task can be done via leave-one-out (LOO) retraining, which measures performance changes when one task is excluded from training. Notably, computing LOO scores requires retraining the model repeatedly; if we have K training tasks, computing the LOO scores for all K tasks requires training the model $K + 1$ times, which is computationally prohibitive when working with large-scale systems. More efficient proxies for the LOO scores involve influence functions (Koh & Liang, 2017; Grosse et al., 2023),

which measure the change of model predictions if one sample is added or removed in the training set. Notice that influence functions are often computed at the terminal state of model training, once it has reached a local minimum (Feldman & Zhang, 2020). Computing the influence functions (for all the K training tasks) again requires repeatedly computing Hessian-vector products as part of the Hessian approximation (Basu et al., 2021; Bae et al., 2022; Kwon et al., 2024). An alternative approach that has emerged, called datamodels (Ilyas et al., 2022) and data attribution (Park et al., 2023; Bae et al., 2024), involves building a surrogate function for approximating the outcome of a black-box model. For example, data models (Ilyas et al., 2022) work by learning a surrogate function that maps individual subsets of training subsets (that is, subsets of $\{1, 2, \ldots, K\}$ in our setting), to their trained outcomes if they are combined together. Notably, existing surrogate modeling methods focus on estimating linear surrogate models, which can be theoretically shown to capture first-order effects (Park et al., 2023; Li et al., 2023).

In this paper, we propose a *kernel surrogate model* instead, motivated by the need to capture non-linear task interactions. We begin by analyzing the residual errors of surrogate models via a second-order Taylor expansion. When the second-order terms in the expansion are negligible, we find that the linear surrogate model coefficients (estimated via minimizing the surrogate model loss) are approximately equal to the influence functions (up to third-order expansion errors). An implication of this result is that linear surrogate models do not capture second-order terms, which correspond to non-linear task interactions. Motivated by these observations, we propose to estimate kernel surrogate models such as the radial basis function (RBF) kernels. Estimating the kernel surrogates requires pre-computing model outcomes on multiple subsets. Instead, we design An efficient estimation procedure by using gradients as features, building on a first-order approximation of the model outputs. Empirically, we validate this first-order approximation to be accurate, incurring less than 1% relative error across a range of datasets, including CIFAR-10, modular arithmetic, in-context learning, and multi-objective RL benchmarks.

We validate the kernel surrogate modeling approach across a wide range of datasets and models. First, we test our central hypothesis on modular arithmetic reasoning tasks. This presents a non-linear learning problem, and we find that by using kernel surrogates, we can achieve a more accurate task attribution score by 42% relative to existing methods (Li et al., 2023; Ilyas et al., 2022; Park et al., 2023). Similarly, we conduct experiments on in-context learning and on a sequential decision-making benchmark, noting qualitatively similar results. By running inference on the Qwen3-8B model with sentiment classification and math reasoning tasks, the attribution scores using kernel surrogates improve over existing methods by 18%. By running soft actor-critic on the Meta-World MT10 benchmark (Yu et al., 2020), we can provide more accurate task attribution results with a 5% improvement for environments where data distributions shift dynamically during policy learning. Our method consistently outperforms existing approaches across all settings, achieving a higher correlation with LOO by 25%. Finally, we use the kernel surrogate model to perform downstream task selection, and find that our approach achieves 40% lower loss for both improved inference and improved optimization in the meta-world benchmark. Moreover, the running time of estimating kernel surrogates remains comparable to that of estimating linear surrogate models.

Taken together, this paper provides an efficient estimation algorithm for task attribution via kernel surrogate models. First, we analyze the residual errors of surrogate models, showing that the influence functions correspond to estimated coefficients of linear surrogate models. Second, we design kernel surrogate models for capturing nonlinear task relationships. Additionally, we provide an efficient estimation algorithm based on projected gradients. Overall, our approach is thus scalable and highly flexible for modeling task relationships in black-box systems. Finally, we evaluate this approach in a wide variety of relevant empirical settings for AI systems, including reasoning, LLM inference, and RL. We provide the code to replicate our results at https://anonymous.4open.science/r/KernelSM.

2 PRELIMINARIES

We begin by introducing the notation and a task weighting framework. Within this framework, we provide a formal definition of task attribution. Building on this foundation, we adapt influence functions and task modeling techniques from the data attribution literature, which serve as state-of-the-art benchmarks for evaluating task contributions without repeated retraining.

Preliminaries. Suppose our training dataset consists of K tasks, denoted by $S = \{T_1, T_2, \ldots, T_K\}$. Each task k contains n_k samples drawn from its task-specific distribution, i.e., $T_k = \{(x_{k,j}, y_{k,j})\}_{j=1}^{n_k}$. The total number of training samples is $n = \sum_{k=1}^{K} n_k$.

We consider a model f_W parameterized by $W \in \mathbb{R}^d$, which is shared across all tasks. For each task k, we denote the task-level loss by $\ell_k(f_W, T_k)$. To specify which tasks are included in training, we introduce a K-dimensional binary vector $\mathbf{s} = [s_1, \cdots, s_K]^\top$, where s_k indicates that task k is selected and $s_k = 0$ otherwise. The corresponding weighted empirical loss is then defined as

$$L(W, \mathbf{s}) = \frac{1}{\sum_{j=1}^{K} s_j} \sum_{k=1}^{K} s_k \ell_k(f_W, T_k). \tag{1}$$

For example, when $s_k = 1$ for all k, the weighted loss $L(W, \mathbf{s})$ reduces to the average loss over all tasks. We denote by $\hat{W}(\mathbf{s}) = \arg\min_W L(W, \mathbf{s})$ the minimizer of the weighted loss, and by $f_{\hat{W}(\mathbf{s})}$ the corresponding trained model.

Influence estimation. We aim to evaluate the influence of training tasks on a target test task T_{test}. To this end, we define a performance metric $F(\mathbf{s}) = \ell_{\text{test}}(f_{\hat{W}(\mathbf{s})}, T_{\text{test}})$, which measures the test loss of the model trained with subset of training tasks indicated by $\mathbf{s}$.

A natural way to quantify the contribution of a task is through leave-one-out (LOO) training, which measures the change in performance when task k is excluded from the training set

$$\mathcal{I}_k^{\text{LOO}} = F(\mathbf{1}_K) - F(\mathbf{1}_K - e_k),$$

where $\mathbf{1}_K$ is the all-ones vector corresponding to training on all tasks, and $e_k \in \{0, 1\}^K$ is the one-hot vector with a 1 in the k-th coordinate.

Definition 2.1. *We define the ground truth of task attribution of task k as $\mathcal{I}_k^{LOO}$.*

An alternative approach is based on extending classical influence functions (Koh & Liang, 2017) to task-weighted empirical loss. Given $\mathbf{s}$, the influence of task k on the metric $F(\mathbf{s})$ is

$$\mathcal{I}_k(\mathbf{s}) = \left[\nabla_W F(\mathbf{s})\right]^\top \left[\nabla_W^2 L(W, \mathbf{s})\right]^{-1} \nabla_W \left(\frac{s_k}{\sum_{j=1}^{K} s_j} \ell_k(f_W, T_k) \right).$$

This expression quantifies how the test performance changes when task k is infinitesimally up-weighted. The gradient term captures task k's direct contribution, the Hessian inverse accounts for curvature of the training loss, and the outer product with $\nabla_W F(\mathbf{s})$ translates the perturbation into its effect on the test metric.

Task modeling. Another approach to quantify task influence is through task modeling (Li et al., 2023). The key idea is to sample binary vectors $\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \ldots, \mathbf{s}^{(m)}$ from $\{0, 1\}^K$, each indicating a random subset of tasks. For each $\mathbf{s}^{(j)}$, we train the model on the corresponding weighted loss to obtain $f_{\hat{W}(\mathbf{s}^{(j)})}$ and record its performance $F(\mathbf{s}^{(j)})$. We then fit a linear surrogate model, parameterized by intercept $\alpha \in \mathbb{R}$ and coefficients $\beta = [\beta_1, \ldots, \beta_K]^\top \in \mathbb{R}^K$, to approximate $F(\mathbf{s})$

$$R(\alpha, \beta) = \mathbb{E}_{\mathbf{s} \sim \mathcal{D}} \left[\left(F(\mathbf{s}) - \alpha - \beta^\top \mathbf{s} \right)^2 \right], \tag{2}$$

where $\mathcal{D}$ is a distribution of $\mathbf{s}$. The optimal surrogate model is obtained by minimizing the empirical counterpart of $R(\alpha, \beta)$ using the m sampled pairs $\{(\mathbf{s}^{(j)}, F(\mathbf{s}^{(j)}))\}_{j=1}^{m}$.

3 METHODS

In this section, we present a kernel-based surrogate modeling approach for influence estimation. First, we analyze surrogate modeling together with influence functions using a second-order Taylor expansion in the task modeling space. We provide an approximation of the surrogate model and find that the first-order approximation is given by the influence function. Thus, linear surrogate models correspond to estimating the influence functions while ignoring non-linear interactions in the task composition. Second, we design a kernel-based surrogate modeling to capture the non-linear interactions between tasks. Finally, we also introduce a gradient estimation algorithm to efficiently find the kernel surrogate.

3.1 Analyzing surrogate modeling in the attribution space

Second-order expansion of surrogate modeling. We center our analysis around $\mathbf{s}^\star = \mathbf{1}_K$, which corresponds to the global model trained on all tasks. Then we can characterize task contributions with respect to this global model. Expanding around this anchor point $\mathbf{s}^*$, we approximate the performance function $F(\mathbf{s})$ using a second-order Taylor expansion:

$$F(\mathbf{s}) \approx F(\mathbf{s}^\star) + \underbrace{[\nabla_{\mathbf{s}} F(\mathbf{s}^\star)]^\top (\mathbf{s} - \mathbf{s}^\star)}_{\text{First-Order (Linear) Effects}} + \underbrace{\frac{1}{2}(\mathbf{s} - \mathbf{s}^\star)^\top \mathbf{H}_{\mathbf{s}} (\mathbf{s} - \mathbf{s}^\star)}_{\text{Second-Order (Interaction) Effects}}, \tag{3}$$

where the Hessian $\mathbf{H}_{\mathbf{s}} = \nabla^2_{\mathbf{s}} F(\mathbf{s}^\star)$ captures the non-linear interactions between tasks.

We next connect this expansion to linear task modeling, which approximates $F(\mathbf{s})$ with a linear surrogate of the form $g(\mathbf{s}) = \alpha + \beta^\top \mathbf{s}$. To analyze the behavior of this surrogate, we substitute the second-order expansion of $F(\mathbf{s})$ in Equation (3) directly into the surrogate model's optimization objective, $\mathbb{E}[(F(\mathbf{s}) - g(\mathbf{s}))^2]$. The following proposition formalizes the resulting characterization of the coefficients.

Proposition 3.1. *Let $F : \{0,1\}^K \to \mathbb{R}$ and fix $\mathbf{s}^\star = \mathbf{1}_K$. Suppose each coordinate in $\mathbf{s}$ is drawn independently from a Bernoulli distribution, $s_k \sim \text{Bernoulli}(p)$. Let $(\hat{\alpha}, \hat{\beta})$ be the parameters that minimize $R(\alpha, \beta)$ in equation (2). Then*

$$\hat{\beta} \approx \nabla_{\mathbf{s}} F(\mathbf{s}^\star) + \underbrace{(p-1)\mathbf{H}_{\mathbf{s}} \cdot \mathbf{1}_K}_{\textit{sampling shift}} + \underbrace{\tfrac{1}{2}(1-2p)\,\text{diag}(\mathbf{H}_{\mathbf{s}})}_{\textit{Bernoulli variance}}. \tag{4}$$

This result shows that the surrogate coefficients $\hat{\beta}$ recover the gradient of F at the fully trained model $\mathbf{s}^\star$, up to two terms induced by the sampling distribution. The first corresponds to a sampling shift due to the mean of the Bernoulli draw, while the second reflects the variance adjustment that accounts for randomness in task inclusion. A detailed proof of this proposition is in Appendix A.2.

Connection between influence functions and linear surrogate models. This result also shows the connection between the linear task modeling and influence functions. Specifically, under a first-order approximation ($\mathbf{H}_s = 0$), both methods are estimators of the same underlying quantity $\nabla_{\mathbf{s}} F(\mathbf{s}^\star)$. Under this assumption, the second-order bias terms in the equation for $\hat{\beta}$ from Proposition 3.1 vanish, directly showing that the linear task modeling coefficients approximate the gradient of the performance landscape: $\hat{\beta} \approx \nabla_{\mathbf{s}} F(\mathbf{s}^\star)$. The influence function, by contrast, is analytically derived to compute $\mathcal{I} = [\mathcal{I}_1(\mathbf{s}^\star), \mathcal{I}_2(\mathbf{s}^\star), \ldots, \mathcal{I}_K(\mathbf{s}^\star)]^\top = \nabla_{\mathbf{s}} F(\mathbf{s}^\star)$. This is stated precisely below.

Corollary 3.2. *Under a first-order assumption where task interactions are negligible, i.e., $\mathbf{H}_{\mathbf{s}} = \mathbf{0}$, the coefficients of a linear surrogate model are approximately equal to the influence function vector:*

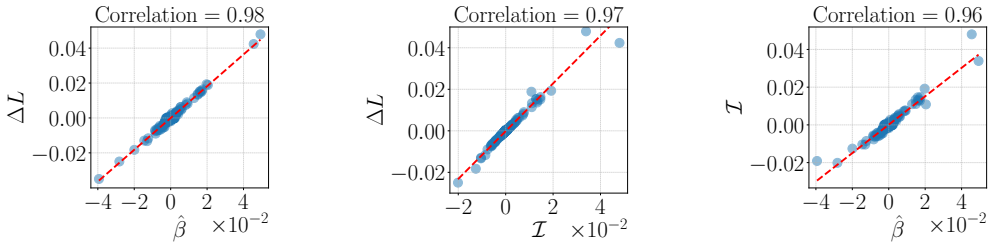
$$\hat{\beta} \approx \mathcal{I} = \nabla_{\mathbf{s}} F(\mathbf{s}^\star).$$

Thus, influence functions provide a first-order approximation of surrogate models, while linear surrogate models capture additive relations. In particular, more expressive attribution methods are needed to express higher-order relationships.

To empirically validate this connection, we experiment with a binary logistic regression setting, where the first-order approximation is expected to hold. We compared the estimates from both the linear task model and influence functions against the ground-truth leave-one-out (LOO) retraining scores. In Figure 1, we found that both methods produced estimates that aligned closely with the ground truth: the Linear Task Model and Influence Functions achieved high Pearson correlations of 0.98 and 0.97, respectively, with the LOO scores. The estimates from the two methods were also highly correlated with each other, with a Pearson correlation of 0.96. This demonstrates that when the performance landscape is nearly linear, both the empirical and analytical approaches indeed converge to the same underlying attribution scores. A detailed description of the experimental setup is available in Appendix C.1.

3.2 Learning kernel surrogate models

To build intuition for why a nonlinear data model is necessary, we present a numerical example in Figure 2. We use a binary classification task with a two-layer MLP as the base classifier. The



(a) Surrogate / LOO (b) Influence func. / LOO (c) Surrogate / Influence func.

Figure 1: We compare the influence functions (IF), leave-one-out (LOO) retraining, and linear surrogate models. Each point corresponds to the individual effect of removing one training example.

final goal for the surrogate model is to predict the MLP's output probability on a fixed test sample, given the subset of training data it was trained on. We specifically analyze the effect of different subsets of training data sampled from near the MLP's decision boundary. The detailed setting is in Appendix C.2.

The influence of a training subset exhibits strong non-linearity that cannot be decomposed into individual sample contributions. When specific samples combine in a training subset, they produce changes in the MLP learned function, causing transitions in test point predictions. The linear surrogate model does not capture these interactions. By contrast, the RBF kernel surrogate model captures these dependencies by modeling the joint influence of sample combinations. This demonstrates that non-linear surrogate models are necessary for capturing nonlinear interactions.

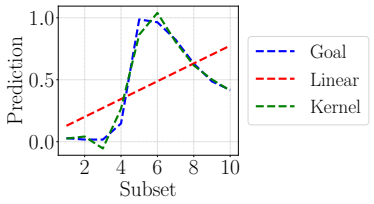


Figure 2: Illustrate linear vs. kernel models on a decision boundary.

Estimating surrogate model using kernel ridge regression. To address the limitations of linear models, we propose the kernel surrogate model algorithm, which replaces the linear mapping with a non-linear function learned through kernel ridge regression. This enables modeling complex, non-additive task interactions.

We learn a kernel surrogate function $g : \{0,1\}^K \to \mathbb{R}$ within a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$ by minimizing the following objective:

$$\min_{g \in \mathcal{H}_k} \sum_{i=1}^{m} \left(F(\mathbf{s}^{(i)}) - g(\mathbf{s}^{(i)})\right)^2 + \lambda \|g\|_{\mathcal{K}}^2, \tag{5}$$

where $\lambda > 0$ is a regularization parameter and $\mathcal{K}$ is an $m \times m$ kernel matrix with entries $\mathcal{K}_{ij} = k(\mathbf{s}^{(i)}, \mathbf{s}^{(j)})$. For example, since input vectors $\mathbf{s}$ are binary, we use the Radial Basis Function (RBF) kernel:

$$k(\mathbf{s}^{(a)}, \mathbf{s}^{(b)}) = \exp\left(-\gamma \|\mathbf{s}^{(a)} - \mathbf{s}^{(b)}\|^2\right). \tag{6}$$

By the representer theorem, the minimizer takes the form: $g(\mathbf{s}) = \sum_{i=1}^{m} c_i k\left(\mathbf{s}^{(i)}, \mathbf{s}\right)$. The coefficient vector $\mathbf{c} = [c_1, \ldots, c_m]^\top$ is computed as: $\mathbf{c} = (\mathcal{K} + \lambda\, \mathrm{Id})^{-1}\mathbf{F}$, where $\mathbf{F} = [F(\mathbf{s}^{(1)}), \ldots, F(\mathbf{s}^{(m)})]^\top$ is the vector of observed outcomes and Id is the identity matrix.

This kernel provides flexibility for capturing complex relationships while maintaining computational tractability. Hyperparameters λ and γ are selected via cross-validation. Unlike linear models that assign fixed coefficients to individual tasks, the kernelized task model learns a global non-linear function that predicts performance for any task combination, thereby capturing intricate inter-dependencies between tasks.

We empirically validated this choice by comparing the RBF kernel with linear and polynomial kernels with degrees $\{1, 2, 3\}$ in Table 1. We use ResNet-9 classifiers trained on the CIFAR-10 dataset (Krizhevsky et al., 2009) as the base model. The detailed description is in Appendix C.4. We find that the RBF kernel achieves a much lower residual error compared to the linear surrogate model. We discuss more kernels in Appendix C.4.

Table 1: We investigate the surrogate model performance with different kernels.

Residual error	Linear	RBF
CIFAR-10	$4.4_{\pm 0.9}$	$1.0_{\pm 0.0}$
Modular arithmetic	$4.6_{\pm 1.3}$	$1.5_{\pm 0.4}$
In-context learning	$0.8_{\pm 0.2}$	$0.4_{\pm 0.1}$
Multi-objective RL	$0.2_{\pm 0.1}$	$0.1_{\pm 0.1}$

Table 2: Relative approximation error of $\epsilon_W(x)$, tested on four different tasks.

	Relative error
CIFAR-10	$1.02_{\pm 0.69}\%$
Modular arithmetic	$2.40_{\pm 2.17}\%$
In-context learning	$0.51_{\pm 0.04}\%$
Multi-objective RL	$0.43_{\pm 0.73}\%$

3.3 Efficient estimation via projected gradients

One downside of surrogate modeling is that it involves training m separate models, one on each randomly-sampled subset. Instead of training the model to get $f_{W^{(i)}}$ on each subset $\mathbf{s}^{(i)}$, one can approximate the model output via a first-order Taylor expansion with perturbed parameters $W = \hat{W} + Z$ as:

$$f_W(x) = f_{\hat{W}}(x) + \langle \nabla_W f_W(x), Z \rangle + \epsilon_W(x),$$

where $Z \in \mathbb{R}^d$ is the parameter perturbation vector and $\nabla_{\hat{W}} f_W(x)$ is the gradient of the model logits with respect to its parameters, evaluated at $\hat{W}$, and $\epsilon_W(x)$ is the first-order Taylor's expansion error of f_W on input x. We empirically evaluate relative approximation error $\left(\epsilon_W(x)/f_{\hat{W}}(x)\right)^2$ in Table 2.

By substituting this approximation into the empirical loss, we can estimate the optimal parameter perturbation $\Delta W^\star_{\mathbf{s}^{(i)}}$. We now get an approximated objective on each subset $\mathbf{s}^{(i)}$ as:

$$Z^\star_{\mathbf{s}^{(i)}} = \arg\min_{Z \in \mathbb{R}^d} \sum_{k=1}^{k} \sum_{(x,y) \in T_k} s_k^{(i)} \cdot \ell\left(f_{\hat{W}}(x) + \langle \nabla_W f_W(x), Z \rangle, y\right),$$

assuming the expansion error of $\epsilon_W(x)$ is negligibly small. This corresponds to a regularized multinomial logistic regression problem, where the gradient serves as features for the regression weights Z. Now, to minimize the above approximated loss on all the subsets $\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \ldots, \mathbf{s}^{(m)}$, one only needs to compute the function values and the gradients at the initialization $\hat{W}$ (e.g., a well-trained LLM), including $f_{\hat{W}}(x)$ for all x in the training set, as well as $\nabla f_{\hat{W}}(x)$ for all possible x.

We estimate the model's output on a test sample $(x_{\text{test}}, y_{\text{test}})$ using the same linear approximation:

$$\hat{f}_{\hat{W}(\mathbf{s}^{(i)})}(x_{\text{test}}) = f_{\hat{W}}(x_{\text{test}}) + \langle \nabla f_{\hat{W}}(x_{\text{test}}), Z^\star_{\mathbf{s}^{(i)}} \rangle.$$

The performance of the subset model is then evaluated as the loss $\ell(\hat{f}_{\hat{W}(\mathbf{s}^{(i)})}(x_{\text{test}}), y_{\text{test}})$ on the test set. The primary challenge in solving this objective is the high dimensionality of the parameter space. We address this by applying random projection to project the gradients into a low-dimensional subspace. In practice, solving the regression with the projected gradients takes a few seconds, allowing for quickly estimating the performance of multiple subsets. See the detailed procedure in Algorithm 2, Appendix B. Taken together, we summarize the overall procedure in Algorithm 1 below.

4 Experiments

Our experiments investigate three key questions: (1) How does KernelSM compare to existing attribution techniques on complex tasks such as modular arithmetic reasoning, in-context learning, and multi-objective reinforcement learning? (2) Can the attributions generated by KernelSM be effectively leveraged for downstream task selection? (3) How does a complex loss landscape impact the performance of KernelSM compared to methods relying on linear surrogate models? We evaluate our approach across three settings, including arithmetic reasoning, in-context learning, and multi-objective reinforcement learning. We find that KernelSM can improve over existing influence estimation methods by 25%. When applied to downstream task selection, KernelSM can improve the performance of baselines by 41%. Various robustness checks and ablation studies validate the consistency of KernelSM under different optimizers and sample sizes.

Algorithm 1 Estimating Kernel Surrogate Models (KERNELSM)

Input: Set of training tasks $\{T_1, \ldots, T_K\}$, test performance metric $F(\cdot)$, number of subsets m, subset sampling distribution D, kernel function $k(\cdot, \cdot)$, ℓ_2-regularization parameter $\lambda > 0$
Output: Regression coefficient vector $\mathbf{c}$
/* *Generating sampled subsets* */
Initialize surrogate dataset $\mathcal{D}_{\text{surrogate}} \leftarrow \emptyset$
Compute initial logits $\{f_{\hat{W}}(z)\}$ and gradients $\{\nabla_{\hat{W}} f_W(z)\}$ for all relevant data points z
for $i = 1, \ldots, m$ **do**
    Sample a task subset, represented by a binary vector $\mathbf{s}^{(i)} \sim D$, where $s_k^{(i)} = 1$ if task T_k is in the subset, and 0 otherwise
    Obtain $F(W^{(i)}) \leftarrow \texttt{GradientEstimation}(\mathbf{s}^{(i)}, \{f_{\hat{W}}(z)\}, \{\nabla_{\hat{W}} f_W(z)\}, F)$
    $\mathcal{D}_{\text{surrogate}} \leftarrow \mathcal{D}_{\text{surrogate}} \cup \{(\mathbf{s}^{(i)}, F(W^{(i)}))\}$
end for
/* *Kernel ridge regression* */
Construct the $m \times m$ kernel matrix $\mathcal{K}$ where $\mathcal{K}_{ij} = k(\mathbf{s}^{(i)}, \mathbf{s}^{(j)})$
Construct the outcome vector $\mathbf{y} = [F(W^{(1)}), \ldots, F(W^{(m)})]^\top$.
Solve for the KRR coefficients $\mathbf{c} \leftarrow (\mathcal{K} + \lambda \,\mathrm{Id})^{-1}\mathbf{y}$
return $g(\mathbf{s}) = \sum_{i=1}^{m} c_i k(\mathbf{s}^{(i)}, \mathbf{s})$

4.1 EXPERIMENT SETUP

Datasets and models. Our primary controlled environment is an arithmetic reasoning task, where we train a small decoder-only Transformer. Consider learning a transformer model to perform modular arithmetic operations over two numbers, in the form of $a \circ b \pmod p = c$, where $\circ$ is an arithmetic operation and p is a prime. Inputs are composed of four tokens: a, $\circ$, b, and $=$, with a and b generated between 0 and $p-1$. The label is the result of the arithmetic operation c. Specifically, we use the addition task $a + b \pmod p = c$ and the quadratic task $a^2 + b^2 + ab \pmod p = c$, where we set $p = 97$.

For more complex scenarios, we evaluate a Qwen3-8B model on in-context classification tasks. We construct prompts with $k = 4$ in-context examples followed by a query. For attribution computation, we treat each example's input embedding as the sample input, and the query's cross-entropy loss as the model score $F(\mathbf{s})$. We evaluate on two datasets: SST-2 and coin flip.

We also evaluate attribution in multi-objective reinforcement learning using the MetaWorld MT10 benchmark. It contains ten different manipulation tasks. We adopt the Soft Actor-Critic (SAC) algorithm as our training protocol. Each task is treated as a sample for attribution analysis, and we evaluate the model score by the average rewards of all tasks.

Baselines. We compare KERNELSM with existing data attribution methods. Influence functions (Koh & Liang, 2017) employ implicit differentiation, computing the inverse Hessian via LISSA. TracIn (Pruthi et al., 2020) traces prediction changes throughout training by computing gradient products between training and test examples. Linear surrogate models (Ilyas et al., 2022) train linear models that predict test behavior from binary indicators of training sample inclusion across multiple model retrainings. Trak (Park et al., 2023) applies random projection and the one-step Newton method to approximate the datamodels' results without retraining.

Evaluation. We evaluate the data attribution methods by comparing them to the linear datamodeling score (LDS) (Ilyas et al., 2022). We compute LDS in the following steps: (1) Sample m fixed subsets of the training set, which are represented by $\mathbf{s}^{(i)}, \ldots, \mathbf{s}^{(m)} \in \{0, 1\}^K$. We sample $\mathbf{s}$ from a Bernoulli distribution with probability p. (2) For each $\mathbf{s}^{(i)}$, we train a model $F(\mathbf{s}^{(i)})$, and use a task attribution method to obtain an estimation $\hat{F}(\mathbf{s}^{(i)})$. (3) We compute the LDS from the Spearman correlation ρ between the real model output and the task attribution estimation.

4.2 EXPERIMENTAL RESULTS

Estimation results. Our results uniformly demonstrate that by moving beyond the linear assumptions of prior work, KERNELSM achieves a more accurate estimation of task attribution. The ad-

Table 3: Summary of comparison results on task attribution. We report the mean and standard deviation from five independent runs.

Methods	Modular arithmetic reasoning		In-context learning		Multitask RL
	Addition	Quadratic	SST-2	Coin flip	Metaworld
Influence functions	$0.03_{\pm 0.01}$	$0.01_{\pm 0.01}$	$0.16_{\pm 0.05}$	$0.05_{\pm 0.10}$	$0.71_{\pm 0.11}$
TracIn	$0.17_{\pm 0.03}$	$0.32_{\pm 0.09}$	$0.21_{\pm 0.05}$	$0.32_{\pm 0.09}$	$0.45_{\pm 0.15}$
TRAK	$0.14_{\pm 0.01}$	$0.28_{\pm 0.06}$	$0.11_{\pm 0.08}$	$0.23_{\pm 0.12}$	$0.42_{\pm 0.19}$
Linear surrogate models	$0.18_{\pm 0.02}$	$0.44_{\pm 0.09}$	$0.33_{\pm 0.05}$	$0.43_{\pm 0.05}$	$0.76_{\pm 0.04}$
KERNELSM	$\mathbf{0.30}_{\pm 0.12}$	$\mathbf{0.52}_{\pm 0.08}$	$\mathbf{0.37}_{\pm 0.02}$	$\mathbf{0.54}_{\pm 0.01}$	$\mathbf{0.80}_{\pm 0.04}$

Table 4: We evaluate KERNELSM on the downstream task selection. We measure performance by the loss in ICL tasks and the target rewards in multi-objective RL tasks. We measure the running time in minutes.

Loss (↓)	SST-2	Coin flip	Time (↓)	Rewards (↑)	MT 10	Time (↓)
Influence func.	$0.23_{\pm 0.21}$	$1.62_{\pm 1.33}$	$17_{\pm 1}$	Influence func.	$16.3_{\pm 1.4}$	$2_{\pm 1}$
Linear surrogate	$0.20_{\pm 0.11}$	$0.05_{\pm 0.03}$	$2_{\pm 1}$	Linear surrogate	$13.1_{\pm 4.3}$	$1_{\pm 1}$
KERNELSM	$0.16_{\pm 0.08}$	$0.02_{\pm 0.03}$	$2_{\pm 1}$	KERNELSM	$18.8_{\pm 5.6}$	$1_{\pm 1}$

vantage is most pronounced in tasks with complex, non-linear structures. For example, in modular arithmetic reasoning, KERNELSM improves the LDS by over 42% on average compared to linear surrogate models. Similarly, on the reasoning Coin Flip ICL task, where the language model exhibits intricate prompt-following behavior that falters linear approximations, our method's ability to capture higher-order interactions yields a 25% relative improvement in LDS. In multi-objective reinforcement learning, where the task's more transparent structure allows the linear baseline to perform well already (LDS of 0.76), KERNELSM still shows an improvement to 0.80.

Task selection results. Next, we show that the accurate attributions from KERNELSM are applicable to downstream optimization tasks. We apply task attribution methods to find optimal groupings for downstream performance. For prompt selection in ICL, we select the top-4 examples with the highest attribution score. For synergistic task selection in RL, we co-train each target task with the top-3 tasks identified by our method to encourage positive transfer.

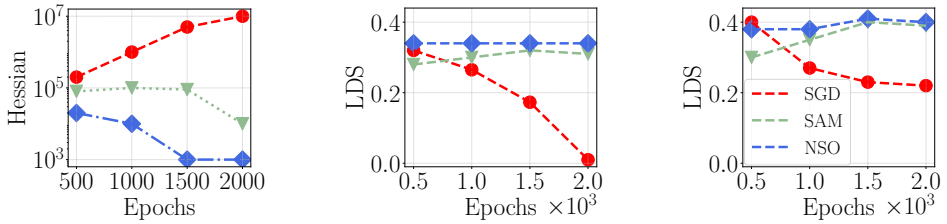
To derive an individual attribution for each item from the kernel surrogate model, we adopt a random ensemble method. We sample multiple subsets and use our kernel surrogate to obtain a performance prediction for each. The final attribution for any given item is then calculated as the average prediction score of all sampled subsets in which that item was included.

As shown in Table 4, KERNELSM achieves a 40% lower loss in the prompt selection task in ICL, and improves the target rewards by 15%. These results demonstrate the benefit of using KERNELSM over linear surrogate models.

4.3 REGULARIZATION OF HESSIAN

Recall from equation (4) that the error of surrogate models is influenced by the second-order terms. Next, we compare the accuracy of linear task modeling by measuring the curvature of the loss Hessian $\nabla^2_W \hat{L}(\hat{W})$, by controlling it with different Hessian regularization. We report results on a modular quadratic task trained using a two-layer transformer. We compared three methods, including the standard SGD without explicit Hessian regularization, and two Hessian regularization methods: sharpness-aware minimization (SAM) (Foret et al., 2021), and noise stability optimization (NSO) (Zhang et al., 2024). We measure the Hessian trace on the global model, and the LDS and residual error of the linear surrogate model are measured on subsets with $\alpha = 0.9$, at 500, 1000, 1500, and 2000 training epochs.

In SGD, the Hessian trace increases continuously throughout training. This worsens the linear surrogate model's performance, characterized by a decrease in LDS. Eventually, the LDS dropped to 0.01. By contrast, both SAM and NSO constrain the Hessian trace. This regularization results in improved model fit, whose LDS remained relatively consistent throughout training.



(a) Hessian trace (b) Linear LDS (c) Kernel LDS

Figure 3: We investigate both linear and kernel surrogate models' fit under different Hessian regularization during model training. The linear surrogate model does not fit model outcomes when using SGD, and only works when its Hessian is regularized. By contrast, kernel surrogate models remain robust when trained with various optimizers and regularization.

5 Related Work

Our work is related to the literature on training data attribution methods. Data attribution methods have been extensively studied to quantify the influence of individual training samples on model behavior (Koh & Liang, 2017; Feldman & Zhang, 2020; Ilyas et al., 2022). In this work, we extend these concepts to task attribution, where we measure the contribution of groups of samples that form different training tasks.

The first line of approach is based on the influence function (Koh & Liang, 2017). These methods compute the influence function. The main challenge is that the influence function involves Hessian computation. Instead of computing the exact Hessian inverse, recent approaches use the Gauss-Newton Hessian approximation (Choe et al., 2024). Influence functions can also be applied to non-decomposable loss functions, such as contrastive loss and preference loss, allowing for data attribution across a broader class of models (Deng et al., 2025). The second line of approach studies the trajectory of model training, rather than just examining the final state of the trained model. These methods aim to trace the training dynamics and predict the counterfactual trajectory that would result from a different training set (Bae et al., 2024). Recent work calculates the exact influence of training data on a single, deterministic model instance by leveraging large-scale meta-gradient computation (Engstrom et al., 2025; Ilyas & Engstrom, 2025). The third line of research involves using a surrogate model to approximate the behavior of the original complex model and then computing the surrogate model. Recent work shows that the model output can be linearized in a certain local area (Li et al., 2024). Under this assumption, one can rephrase the model output as a first-order Taylor expansion on the parameters. This method provides an efficient estimation for the leave-one-out score (Li et al., 2023) and for datamodels (Park et al., 2023). Building on top of surrogate model methods, our work aims to improve upon this by using a kernel-based model to better capture the non-linear interactions between tasks.

6 Conclusion

In this paper, we study the problem of task attribution, which aims to quantify the influence of different training tasks on a model's final performance. We first unified the theories of existing influence functions and linear surrogate models, highlighting their limitations in handling non-linear interactions between tasks. To overcome this challenge, we proposed a new kernel-based attribution method. This approach effectively captures complex task interactions and does not rely on specific assumptions about the training process. This makes it broadly applicable to emerging paradigms, including in-context learning and reinforcement learning. Furthermore, we introduced an efficient gradient approximation technique that avoids expensive retraining, ensuring our method is scalable. We validated our method's effectiveness with experiments across several distinct domains. The experiment results demonstrate that our approach provides more accurate attribution than existing methods in a diverse range of settings that are relevant to modern AI systems.

ETHICS STATEMENT

Our study does not involve human subjects, personally identifiable information, or sensitive private data. All datasets used are publicly available and have been processed in compliance with their respective licenses. We do not foresee harmful applications or discriminatory outcomes stemming from this work. Our research complies with the ICLR Code of Ethics and institutional research integrity guidelines.

REPRODUCIBILITY STATEMENT

We have taken several steps to ensure reproducibility. The main paper includes detailed descriptions of our model architecture, training procedure, and evaluation settings. A full account of assumptions and theoretical background is provided in the appendix. All datasets used are publicly available and described with preprocessing details in the supplementary materials. To further support reproducibility, we have provided an anonymous repository containing source code, hyperparameters, and scripts to reproduce all experiments.

THE USE OF LARGE LANGUAGE MODELS

In this paper, we have used the LLM to polish the English writing.

REFERENCES

Juhan Bae, Nathan Ng, Alston Lo, Marzyeh Ghassemi, and Roger B Grosse. If influence functions are the answer, then what is the question? *Advances in Neural Information Processing Systems*, 35:17953–17967, 2022.

Juhan Bae, Wu Lin, Jonathan Lorraine, and Roger Baker Grosse. Training data attribution via approximate unrolling. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Samyadeep Basu, Phil Pope, and Soheil Feizi. Influence functions in deep learning are fragile. In *International Conference on Learning Representations*, 2021.

Sang Keun Choe, Hwijeen Ahn, Juhan Bae, Kewen Zhao, Minsoo Kang, Youngseog Chung, Adithya Pratapa, Willie Neiswanger, Emma Strubell, Teruko Mitamura, et al. What is your data worth to gpt? llm-scale data valuation with influence functions. *arXiv preprint arXiv:2405.13954*, 2024.

Junwei Deng, Weijing Tang, and Jiaqi W. Ma. A versatile influence function for data attribution with non-decomposable loss. In *Forty-second International Conference on Machine Learning*, 2025.

Samuel Deng and Daniel Hsu. Multi-group learning for hierarchical groups. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 10440–10487, 2024.

Logan Engstrom, Andrew Ilyas, Benjamin Chen, Axel Feldmann, William Moses, and Aleksander Madry. Optimizing ml training with metagradient descent. *arXiv preprint arXiv:2503.13751*, 2025.

Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation. *Advances in Neural Information Processing Systems*, 33:2881–2891, 2020.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021.

Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in neural information processing systems*, 35:30583–30598, 2022.

Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, et al. Studying large language model generalization with influence functions. *arXiv preprint arXiv:2308.03296*, 2023.

Andrew Ilyas and Logan Engstrom. Magic: Near-optimal data attribution for deep learning. *arXiv preprint arXiv:2504.16430*, 2025.

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. 2022.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894. PMLR, 2017.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Yongchan Kwon, Eric Wu, Kevin Wu, and James Zou. Datainf: Efficiently estimating data influence in lora-tuned llms and diffusion models. In *The Twelfth International Conference on Learning Representations*, 2024.

Dongyue Li, Huy Nguyen, and Hongyang Ryan Zhang. Identification of negative transfers in multitask learning using surrogate models. *Transactions on Machine Learning Research*, 2023.

Dongyue Li, Ziniu Zhang, Lu Wang, and Hongyang Zhang. Scalable fine-tuning from multiple data sources: A first-order approximation approach. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 5608–5623, 2024.

Sung Min Park, Kristian Georgiev, Andrew Ilyas, Guillaume Leclerc, and Aleksander Madry. Trak: Attributing model behavior at scale. In *International Conference on Machine Learning*, pp. 27074–27113. PMLR, 2023.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33: 19920–19930, 2020.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32, 2019.

Fan Yang, Hongyang R Zhang, Sen Wu, Christopher Re, and Weijie J Su. Precise high-dimensional asymptotics for quantifying heterogeneous transfers. *Journal of Machine Learning Research*, 26 (113):1–88, 2025.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pp. 1094–1100. PMLR, 2020.

Hongyang R. Zhang, Dongyue Li, and Haotian Ju. Noise stability optimization for finding flat minima: A hessian-based regularization approach. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

Ziniu Zhang, Zhenshuo Zhang, Dongyue Li, Lu Wang, Jennifer Dy, and Hongyang R Zhang. Linear-time demonstration selection for in-context learning via gradient estimation. *Empirical methods in natural language processing*, 2025.

A Complete Proofs

A.1 Derivation of influence functions

Let $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ be a dataset including n samples from an unknown data distribution. Let f_W denote a model with parameters $W \in \mathbb{R}^d$. Let $\hat{L}(f_W)$ denote the empirical training loss of the model f_W, averaged over the n training data samples. The textbook definition of influence, as defined by how much a trained model changes after adding or removing one sample z, is given by $\left(\nabla^2 \hat{L}(f_W)\right)^{-1} \nabla \hat{L}(z, f_W)$. To derive this result, let the perturbed parameter after adding input z with step size ϵ be given by:

$$\hat{W}\epsilon, z = \arg\min_{W \in W} \left(\hat{L}(W) + \epsilon L(z, W)\right) \tag{7}$$

Define the parameter change $\Delta_\epsilon = \hat{W}_{\epsilon,z} - \hat{W}$, and note that, as $\hat{W}$ does not depend on ϵ, we can thus write:

$$\frac{d\hat{W}_{\epsilon,z}}{d\epsilon} = \frac{d\Delta_\epsilon}{d\epsilon}.$$

Based on first-order optimality conditions, from Eq. (7), we get:

$$\nabla \hat{L}(\hat{W}_{\epsilon,z}) + \epsilon \nabla L(z, \hat{W}_{\epsilon,z}).$$

Since $\hat{W}_{\epsilon,z} \to \hat{W}$ as $\epsilon \to 0$, we perform Taylor's expansion with the anchor set at $\hat{W}$ up to first-order as:

$$\left(\nabla \hat{L}(\hat{W}) + \epsilon \nabla L(z, \hat{W})\right) + \left(\nabla^2 \hat{L}(\hat{W}) + \epsilon \nabla^2 L(z, \hat{W})\right) \Delta_\epsilon \approx 0, \tag{8}$$

which is approximately zero after dropping the second-order term. After setting ϵ to approaching zero, and solving for Δ_ϵ, we get:

$$\Delta_\epsilon = -[\nabla^2 \hat{L}(\hat{W})]^{-1} \left(\nabla \hat{L}(\hat{W}) + \epsilon \nabla L(z, \hat{W})\right) = -\epsilon [\nabla^2 \hat{L}(\hat{W})]^{-1} \nabla L(z, \hat{W}), \tag{9}$$

since $\nabla \hat{L}(\hat{W}) = 0$. Through the chain rule, the influence on any differentiable quantity F becomes:

$$\mathcal{I} = [\nabla_W F(f_{\hat{W}})]^\top \left[\nabla^2 \hat{L}(f_W)\right]^{-1} \nabla \hat{L}(z, f_W). \tag{10}$$

A.2 Proof of Proposition 3.1

Proof. We let $\mu = \mathbb{E}[\mathbf{s}] = p\,\mathbf{1}_K$. First, we derive the population OLS coefficients. Since features are independent, the population OLS coefficient for feature $\mathbf{s}_k$ is

$$\hat{\beta}_k = \frac{\mathrm{Cov}(\mathbf{s}_k, F(\mathbf{s}))}{\mathrm{Var}(\mathbf{s}_k)}.$$

We decompose the covariance using the Taylor expansion:

$$\mathrm{Cov}(\mathbf{s}_k, F(\mathbf{s})) \approx \underbrace{\mathrm{Cov}(\mathbf{s}_k, F(\mathbf{s}^\star))}_{=0} + \underbrace{\mathrm{Cov}\big(\mathbf{s}_k, g^\top(\mathbf{s} - \mathbf{s}^\star)\big)}_{\text{linear}} + \underbrace{\mathrm{Cov}\big(\mathbf{s}_k, \tfrac{1}{2}(\mathbf{s} - \mathbf{s}^\star)^\top H_s (\mathbf{s} - \mathbf{s}^\star)\big)}_{\text{quadratic}}.$$

The linear term reduces to $g_k \,\mathrm{Var}(\mathbf{s}_k)$ by independence. For the quadratic term, write $\mathbf{s} - \mathbf{s}^\star = (\mathbf{s} - \mu) + (\mu - \mathbf{s}^\star)$ and expand:

$$Q = \tfrac{1}{2}\Big[(\mathbf{s} - \mu)^\top H_s (\mathbf{s} - \mu) + 2(\mu - \mathbf{s}^\star)^\top H_s (\mathbf{s} - \mu) + (\mu - \mathbf{s}^\star)^\top H_s (\mu - \mathbf{s}^\star)\Big].$$

The constant part vanishes in covariance. For the mixed part,

$$\mathrm{Cov}\big(\mathbf{s}_k, (\mu - \mathbf{s}^\star)^\top H_s (\mathbf{s} - \mu)\big) = \mathrm{Var}(\mathbf{s}_k) \sum_j H_{kj}(\mu_j - \mathbf{s}_j^\star),$$

using symmetry of H_s. For the centered quadratic,

$$(\mathbf{s} - \mu)^\top H_s (\mathbf{s} - \mu) = \sum_i H_{ii}(\mathbf{s}_i - \mu_i)^2 + 2 \sum_{i<j} H_{ij}(\mathbf{s}_i - \mu_i)(\mathbf{s}_j - \mu_j).$$

All cross-covariances with $i \neq j$ vanish by independence and zero mean of $(\mathbf{s}_j - \mu_j)$, leaving

$$\tfrac{1}{2}\operatorname{Cov}\big(\mathbf{s}_k, (\mathbf{s}-\mu)^\top H_s(\mathbf{s}-\mu)\big) = \tfrac{1}{2}\,H_{kk}\operatorname{Cov}\big(\mathbf{s}_k, (\mathbf{s}_k-\mu_k)^2\big).$$

With $X_k = \mathbf{s}_k - \mu_k$, $\operatorname{Cov}(\mathbf{s}_k, X_k^2) = \mathbb{E}[X_k^3]$, and for Bernoulli$(p)$, $\mathbb{E}[X_k^3] = p(1-p)(1-2p)$. Hence

$$\operatorname{Cov}(\mathbf{s}_k, Q) = \operatorname{Var}(\mathbf{s}_k)\sum_j H_{kj}(\mu_j - \mathbf{s}_j^\star) + \tfrac{1}{2}\,H_{kk}\,p(1-p)(1-2p),$$

which yields

$$\hat\beta_k \approx g_k + \sum_j H_{kj}(\mu_j - \mathbf{s}_j^\star) + \tfrac{1}{2}H_{kk}(1-2p).$$

Stacking over k gives (4).

Intercept. Using $\hat\beta_0 = \mathbb{E}[F(\mathbf{s})] - \hat\beta^\top\mu$ and $\mathbb{E}[\mathbf{s}-\mu] = 0$,

$$\mathbb{E}[F(\mathbf{s})] \approx F(\mathbf{s}^\star) + g^\top(\mu - \mathbf{s}^\star) + \tfrac{1}{2}\Big((\mu-\mathbf{s}^\star)^\top H_s(\mu-\mathbf{s}^\star) + \operatorname{Tr}\big(H_s\operatorname{Cov}(\mathbf{s})\big)\Big).$$

Residual mean-squared error. Let $X = \mathbf{s} - \mu$ (zero mean, independent coordinates). The optimal linear predictor removes the constant and all components in the span of $\{X_k\}$. Therefore

$$\mathcal{E} = \operatorname{Var}\Big[Q - \mathbb{E}[Q] - \sum_k b_k X_k\Big] = \operatorname{Var}(Q) - \sum_k \frac{\operatorname{Cov}[X_k, Q]^2}{\operatorname{Var}[X_k]}.$$

Because $\operatorname{Cov}[X_k, Q] = \operatorname{Cov}[\mathbf{s}_k, Q]$ and $\operatorname{Var}[X_k] = \operatorname{Var}[\mathbf{s}_k] = p(1-p)$, this gives

$$\mathcal{E} \;\equiv\; \min_{\beta_0,\beta}\mathbb{E}\big[(F(\mathbf{s}) - \beta_0 - \beta^\top\mathbf{s})^2\big] \;\approx\; \operatorname{Var}(Q) - \sum_{k=1}^{K}\frac{\operatorname{Cov}(\mathbf{s}_k, Q)^2}{\operatorname{Var}(\mathbf{s}_k)}. \tag{11}$$

It remains to compute $\operatorname{Var}[Q]$. Since $Q = \frac{1}{2}X^\top H_s X$,

$$X^\top H_s X = \sum_i H_{ii}X_i^2 + 2\sum_{i<j} H_{ij}X_iX_j$$
$$\Rightarrow \quad \operatorname{Var}(X^\top H_s X) = \sum_i H_{ii}^2\operatorname{Var}[X_i^2] + 4\sum_{i<j} H_{ij}^2\operatorname{Var}[X_iX_j],$$

by independence and mean zero, eliminating mixed covariances. Thus

$$\operatorname{Var}(Q) = \tfrac{1}{4}\operatorname{Var}(X^\top H_s X).$$

For Bernoulli(p), $X_i \in \{1-p, -p\}$ with $\mathbb{E}[X_i^2] = p(1-p)$, $\mathbb{E}[X_i^4] = p(1-p)^4 + (1-p)p^4$, so

$$\operatorname{Var}(X_i^2) = \mathbb{E}[X_i^4] - \mathbb{E}[X_i^2]^2 = p(1-p)(1-2p)^2,$$
$$\operatorname{Var}(X_iX_j) = \mathbb{E}[X_i^2]\mathbb{E}[X_j^2] = [p(1-p)]^2 \;(i \neq j).$$

Substituting gives

$$\operatorname{Cov}(\mathbf{s}_k, Q) = p(1-p)\sum_j H_{kj}(\mu_j - \mathbf{s}_j^\star) + \tfrac{1}{2}\,H_{kk}\,p(1-p)(1-2p), \tag{12}$$

$$\operatorname{Var}(Q) = \tfrac{1}{4}\left[\sum_i H_{ii}^2\,p(1-p)(1-2p)^2 + 4\sum_{i<j} H_{ij}^2\,\big(p(1-p)\big)^2\right]. \tag{13}$$

□

A.3 Proof of Corollary 3.2

Proof. Next, we show that the influence function estimates the gradients of the performance function. Our goal is to show that the vector of influence functions $\mathcal{I} = (\mathcal{I}_1, \ldots, \mathcal{I}_K)^\top$ is equal to the gradient vector $\nabla_{\mathbf{s}} F(\mathbf{s})$.

The optimal parameters $\hat{W}(\mathbf{s}) = \arg\min_W \hat{L}(W, \mathbf{s})$. The empirical risk minimizer corresponds to the uniform weight vector $\mathbf{s}^* = \mathbf{1}^K$. Let $F(\mathbf{s})$ be any differentiable performance metric that depends on the optimal parameters, e.g., the loss on a test point, $F(\mathbf{s}) = \ell(f_{\hat{W}(\mathbf{s})}(x_{\text{test}}), y_{\text{test}})$. The gradient of this function at the standard training point, $\nabla_{\mathbf{s}} F(\mathbf{s}^\star)$.

The original influence function uses an additive perturbation ε for a single sample i. The perturbed objective is:

$$L_{\text{pert}}(W, \epsilon) = \left(\frac{1}{K} \sum_{j=1}^{K} \ell(f_W(x_j), y_j) \right) + \frac{\epsilon}{n} \ell(f_W(x_i), y_i). \tag{14}$$

We can rewrite this expression by collecting terms for each sample:

$$L_{\text{pert}}(W, \epsilon) = \frac{1}{K} \left((1+\epsilon)\ell(f_W(x_i), y_i) + \sum_{j \neq i} \ell(f_W(x_j), y_j) \right). \tag{15}$$

This shows that the additive perturbation framework is a specific instance of the general task weight framework. It corresponds to a path in the n-dimensional weight space $\mathbf{s}$, where the path is defined as:

$$\mathbf{s}(\epsilon) = (1, 1, \ldots, \underbrace{1+\epsilon}_{i\text{-th pos.}}, \ldots, 1)^\top = \mathbf{s}^\star + \epsilon \cdot e_i. \tag{16}$$

Here, e_i is the i-th standard basis vector.

The influence function for sample i, $\mathcal{I}_i(F)$, is defined as the derivative of the performance function F with respect to ε, evaluated as $\varepsilon = 0$. We can compute this derivative using the multivariate chain rule on our general function $F(\mathbf{s})$:

$$\mathcal{I}_i(F) = \left. \frac{dF(\mathbf{s}(\epsilon))}{d\epsilon} \right|_{\epsilon=0}. \tag{17}$$

The chain rule states:

$$\frac{dF(\mathbf{s}(\epsilon))}{d\epsilon} = \nabla_{\mathbf{s}} F(\mathbf{s}(\epsilon))^\top \cdot \frac{d\mathbf{s}(\epsilon)}{d\epsilon}. \tag{18}$$

From our definition, $\mathbf{s}(\epsilon) = \mathbf{s}^* + \epsilon \cdot e_i$, the derivative of the path vector is simply $\frac{d\mathbf{s}(\epsilon)}{d\epsilon} = e_i$. Substituting this in, we get:

$$\frac{dF(\mathbf{s}(\epsilon))}{d\epsilon} = \nabla_{\mathbf{s}} F(\mathbf{s}(\epsilon))^\top \cdot e_i = \frac{\partial F(\mathbf{s}(\epsilon))}{\partial \mathbf{s}_i}. \tag{19}$$

This means the derivative with respect to the perturbation parameter ε is exactly the partial derivative with respect to the weight $\mathbf{s}_i$.

Finally, we evaluate this at $\varepsilon = 0$. At this point, $\mathbf{s}(0) = \mathbf{s}^*$. Therefore:

$$\mathcal{I}_i(F) = \left. \frac{\partial F(\mathbf{s}(\epsilon))}{\partial \epsilon} \right|_{\epsilon=0} = \left. \frac{\partial F(\mathbf{s})}{\partial \mathbf{s}_i} \right|_{\mathbf{s}=\mathbf{s}^\star}. \tag{20}$$

Since this holds for all $i = 1, \ldots, K$, the vector of influence functions is identical to the gradient vector of the performance function:

$$\mathcal{I} = (\mathcal{I}_1, \ldots, \mathcal{I}_K)^\top = \nabla_{\mathbf{s}} F(\mathbf{s}^\star). \tag{21}$$

Next, we examine the optimal solution of task modeling, which estimates the gradients of the performance function. Denote the quality of a linear data model $\beta \in \mathbb{R}^K$ over training set B given by:

$$R(\alpha, \beta) = \mathbb{E}_{s\sim\mathcal{D}}\left[\left(F(\mathbf{s}) - \alpha - \sum_{i=1}^{K} \beta_i s_i\right)^2\right] \tag{22}$$

Let $\mu = \mathbb{E}[\mathbf{s}] \in \mathbb{R}^K$, $\mu_F = \mathbb{E}[F(\mathbf{s})] \in \mathbb{R}$, $\Sigma = \mathrm{Cov}(\mathbf{s}) = \mathbb{E}[(\mathbf{s}-\mu)(\mathbf{s}-\mu)^\top]$, and $c = \mathrm{Cov}(\mathbf{s}, F) = \mathbb{E}[(\mathbf{s}-\mu)(F(\mathbf{s})-\mu_F)]$.

Then the minimizer of

$$R(\alpha, \beta) = \mathbb{E}\big[(F(\mathbf{s}) - \alpha - \beta^\top \mathbf{s})^2\big]$$

is

$$\hat{\beta} = \Sigma^\dagger c, \quad \hat{\beta}_0 = \mu_F - \hat{\beta}^\top \mu.$$

The minimal residual (risk) is

$$R(\hat{\alpha}, \hat{\beta}) = \mathrm{Var}(F(\mathbf{s})) - c^\top \Sigma^\dagger c.$$

With the first-order Taylor expansion on $F(\mathbf{s}^\star)$:

$$F(\mathbf{s}) = F(\mathbf{s}^\star) + \left[\frac{\partial F(\mathbf{s})}{\partial \mathbf{s}}\Big|_{\mathbf{s}=\mathbf{s}^\star}\right]^\top (\mathbf{s} - \mathbf{s}^\star) + r(\mathbf{s}). \tag{23}$$

Define the first-order expansion

$$F(\mathbf{s}) = F(\mathbf{s}^\star) + g^\top(\mathbf{s} - \mathbf{s}^\star) + R(\mathbf{s}),$$

where $g = \nabla_{\mathbf{s}} F(\mathbf{s}^\star)$. Neglecting quadratic remainder terms,

$$c \approx \Sigma g, \qquad \mu_F \approx F(\mathbf{s}^\star) + g^\top(\mu - \mathbf{s}^\star).$$

Therefore,

$$\hat{\beta} \approx \Sigma^\dagger \Sigma\, g = \Pi_\Sigma g, \qquad \hat{\beta}_0 \approx F(\mathbf{s}^\star) - g^\top \mathbf{s}^\star + (g - \Pi_\Sigma g)^\top \mu,$$

where $\Pi_\Sigma := \Sigma^\dagger\Sigma = \Sigma\Sigma^\dagger$ projects onto $\mathrm{range}(\Sigma)$. In particular, if Σ is full rank (or $g \in \mathrm{range}(\Sigma)$), then $\hat{\beta} \approx g$ and $\hat{\beta}_0 \approx F(\mathbf{s}^\star) - g^\top \mathbf{s}^\star$. Here the full rank of Σ follows from the random sampling of subsets, and we need the number of subsets to be large enough so that it's full rank (See Section 3.2, Li et al. (2023)).

By combining these two results, we have proven the proposition that when the respective assumptions for both methods are satisfied, they are numerically equivalent, $\hat{\beta} = g$. □

B ALGORITHM DETAILS

Random Projection. This method directly addresses the high dimensionality of the gradient by projecting it into a low-dimensional subspace. We employ a random matrix $P \in \mathbb{R}^{k\times d}$ (where $k \ll d$) to map the gradient $\nabla f_W(x) \in \mathbb{R}^d$ to a compressed representation $\nabla \tilde{f}_W(x) = P\nabla f_W(x) \in \mathbb{R}^{C\times k}$. Motivated by the Johnson-Lindenstrauss Lemma, which guarantees that pairwise distances are approximately preserved, we then solve the perturbation objective using these low-dimensional features. The optimization is performed over a k-dimensional vector, making the problem tractable and independent of the original parameter count d.

Gradient estimation for in-context learning. For our experiments on the ICL task, we define the loss function and compute gradients in the embedding space. This approach enables us to avoid perturbation-based objective optimization methods, as we already know the difference in embeddings. Thus, we can directly perform a first-order estimation of the logit outputs for different prompts based on the gap between their embeddings. Consider the model output of a prompt subset S on an input x, denoted as $f_W(\phi(S, x))$. Given an anchor prompt S_0, the first-order approximation of f_W around the embedding vector $\phi(S_0, x)$ is given by:

$$f_W(\phi(S,x)) = f_W(\phi(S_0,x)) + \langle \nabla_\phi f_W(\phi(S_0,x)), \phi(S,x) - \phi(S_0,x)\rangle + \epsilon_{S,x}. \tag{24}$$

Algorithm 2 `GradientEstimation`

Input: Task subset vector $\mathbf{s}^{(i)}$; pre-computed logits $\{f_{\hat{W}}(z)\}$; pre-computed gradients $\{\nabla_{\hat{W}} f_W(x)\}$; test task T_{test}
Requires: ℓ_2 regularization $\lambda > 0$; random projection matrix $P \in \mathbb{R}^{k \times d}$
Output: Estimated performance on the test task, $y^{(i)}$
Construct the sample subset $S^{(i)}_{\text{train}} = \{z_j = (x_j, y_j)\}_{j=1}^{|S|}$ from tasks in $\mathbf{s}^{(i)}$
/* *Step 1: Project gradients and solve for low-dimensional regression* */
Compute projected gradients for all samples in the subset: $g_j = P \nabla f_W(x_j)$
Define the low-dimensional objective over $Z \in \mathbb{R}^k$:

$$\mathcal{L}_{\text{proj}}(Z) = \sum_{j=1}^{|S|} \ell_{\text{CE}} \left(f_{\hat{W}}(x_j) + \langle g_j, Z \rangle, y_j \right) + \frac{\lambda}{2} \|Z\|_2^2$$

Compute the optimal low-dimensional regression by minimizing this convex objective:

$$Z^\star = \arg\min_{Z \in \mathbb{R}^k} \mathcal{L}_{\text{proj}}(Z)$$

/* *Step 2: Estimate performance* */
$y^{(i)} \leftarrow 0$
for each $z_{\text{test}} = (x_{\text{test}}, y_{\text{test}}) \in T_{\text{test}}$ **do**
    Project the test gradient: $g_{\text{test}} = P \nabla f_{\hat{W}}(x_{\text{test}})$
    Approximate the test logits using the low-dimensional projections:

$$\hat{f}_{\hat{W}(S^{(i)})}(x_{\text{test}}) = f_{\hat{W}}(x_{\text{test}}) + \langle g_{\text{test}}, Z^\star \rangle$$

    Accumulate the loss: $y^{(i)} \leftarrow y^{(i)} + \ell(\hat{f}_{\hat{W}(S^{(i)})}(x_{\text{test}}), y_{\text{test}})$
end for
$y^{(i)} \leftarrow y^{(i)} / |T_{\text{test}}|$ ▷ Average loss over the test task
return $y^{(i)}$

C Experiment Details

C.1 Correlation between surrogate models and influence functions

We designed an experiment using an ℓ_2-regularized logistic regression model (with a 10^{-2} ℓ_2 penalty) on the Wisconsin Breast Cancer dataset. The data was standardized and split into a training set of 455 samples and a test set. Our analysis focused on quantifying the influence of each training point on the loss of a single, randomly selected test point.

We computed three distinct valuation scores for every training point: the ground-truth leave-one-out (LOO) score, the first-order influence function (IF) approximation, and the coefficients from a linear surrogate model. The LOO scores were obtained by exhaustively retraining the model from scratch (cold-start) after removing each training point individually and recording the change in test loss. The IF scores were calculated as a first-order approximation using the Hessian of the full training loss, stabilized with a 10^{-3} damping term. To build the surrogate model, we generated 1000 subsets, each containing 430 training points, and retrained a model on each to measure the resulting test loss change. These loss changes were then used as targets in a final linear regression, whose coefficients, solved via ordinary least squares (OLS), provided the surrogate model scores.

C.2 Non-linear numerical simulation

We consider a binary classification task in a feature space $\mathcal{X} \in \mathbb{R}^2$ with labels $\mathcal{Y} \in \{0, 1\}$. The base learning algorithm, denoted by $\mathcal{A}$, is a two-layer MLP. The network architecture consists of an input layer, two hidden layers with 16 neurons each and ReLU activation functions, and a final output layer with a softmax activation.

Table 5: We investigate the surrogate model performance with different kernels. We run each experiment with five random seeds to report the standard deviations.

Residual error	Linear regression	Degree-1	Degree-2	Degree-3	RBF
CIFAR-10	$4.4_{\pm 0.9}$	$3.8_{\pm 0.8}$	$1.6_{\pm 0.1}$	$2.7_{\pm 0.2}$	$1.0_{\pm 0.0}$
Modular arithmetic	$4.6_{\pm 1.3}$	$2.2_{\pm 0.5}$	$1.7_{\pm 0.4}$	$3.3_{\pm 0.7}$	$1.5_{\pm 0.4}$
In-context learning	$0.8_{\pm 0.2}$	$0.6_{\pm 0.1}$	$0.5_{\pm 0.1}$	$0.5_{\pm 0.1}$	$0.4_{\pm 0.1}$
Multi-objective RL	$0.2_{\pm 0.1}$	$0.2_{\pm 0.1}$	$0.1_{\pm 0.1}$	$0.1_{\pm 0.1}$	$0.1_{\pm 0.1}$

To analyze the influence of different training samples, we construct a series of related but distinct training subsets. We first define a small, fixed set of N anchor data points, $S_{\text{anchor}} = \{(x_i, y_i)_{i=1}^N\}$. Next, we define a candidate set of K additional data points, $\mathcal{C} = \{c_1, c_2, \ldots, c_K\}$, which are sampled along the path that traverses a critical region near this boundary. The experiment then focuses on a series of K training subsets, where each subset S_j is formed by combining the fixed anchor set with exactly one candidate point from $\mathcal{C}$: $S_j = S_{\text{anchor}} \cup \{(c_j, y_c)\}, \quad \text{for } j = 1, \ldots, K$. We select a fixed test point $x_{\text{test}} \in \mathcal{X}$ whose prediction is sensitive to the location of the decision boundary. The goal is to attribute the model's prediction on x_{test} to the choice of the candidate point c_j.

C.3 Ablation studies

Kernel methods often require a higher sample complexity to converge to an optimal solution compared to linear models. We conduct an ablation study to investigate this trade-off. We compared the performance of the kernel surrogate model with that of the linear surrogate model, varying the size of the data subset used for their construction, sampling from 20% to 80% of the total subsets. The results in Figure 4 show that the kernel surrogate model consistently outperforms the linear surrogate model across all tested subset sizes. Notably, even when the number of samples was small (e.g., at the 20% subset level), the kernel method still demonstrated a better performance.

C.4 Comparison of kernels

We use the CIFAR-10 dataset and the ResNet-9 classifier for our experiments. The dataset consists of 60,000 32×32 color images in 10 classes, with 6,000 images per class. We utilize the standard training set of 50,000 images. To align with the scope of our task attribution methodology, we partition the 50,000 training samples into 50 disjoint tasks. Each task is constructed by randomly sampling 1,000 data points from the training set without replacement. This results in 50 distinct tasks, which form the basis of our analysis. For the training of our surrogate model, we randomly select 30 of these 50 tasks.

We evaluate the performance of different kernels. Let $\mathbf{s}^{(a)}$ and $\mathbf{s}^{(b)}$ denote different subset indices. The kernels evaluated in our study are:

- Polynomial Kernel (Degree-1, 2, 3): The general form of the polynomial kernel is
$$k(\mathbf{s}^{(a)}, \mathbf{s}^{(b)}) = (\mathbf{s}^{(a)\top}\mathbf{s}^{(b)} + c)^d.$$
In our experiments, we set the constant $c = 0$ and test for degrees $d \in \{1, 2, 3\}$.
- Radial Basis Function (RBF) Kernel: The RBF kernel is defined by the following equation:
$$k(\mathbf{s}^{(a)}, \mathbf{s}^{(b)}) = \exp\big(-\gamma \|\mathbf{s}^{(a)} - \mathbf{s}^{(b)}\|^2\big).$$
In our experiments, we set $\gamma = 10^{-5}$.

In Table 5, we find that the linear surrogate model (Task modeling) and the degree-1 kernel have a large residual error. The degree-2 kernel achieves the lowest residual error among the three polynomial kernels. And the RBF kernel achieves the lowest residual error among all the kernels.

C.5 Omitted experimental setup

Modular arithmetic tasks. For modular arithmetic tasks, we consider the following functions, with $p = 97$: $a + b \pmod p = c$, and $a^2 + ab + b^2 \pmod p = c$. For each task, we generate a complete

dataset by iterating through all possible pairs of (a, b), where $a, b \in \{0, 1, \ldots, p-1\}$. This results in a dataset of $97 \times 97 = 9,409$ unique equations for each function. Each equation is formatted as a sequence of tokens: `a`, `op`, `b`, `=`, `c`, where `op` is + for addition and a placeholder token for the quadratic function. We randomly split the whole dataset into a training set comprising 90% of the data and a test set comprising the remaining 10%.

We use a standard two-layer decoder-only transformer with embedding dimension 128 as the classifier for all modular experiments. The model is trained to predict the correct token for the result `c`, given the preceding sequence (`a`, `op`, `b`, `=`). The cross-entropy loss and gradients are computed based on the model's output logits at the final token position.

To support our task attribution analysis, we employ a grouped subset sampling strategy that partitions the training data based on the values of both operands, a and b. We first partition the range of possible values for each operand, $\{0, 1, \ldots, 96\}$, into 5 disjoint intervals. Specifically, a training example (a, b, c) is assigned to a group $G_{i,j}$ where $i = \lfloor a/20 \rfloor$ and $j = \lfloor b/20 \rfloor$. This creates a 5×5 grid of 25 groups, where each group corresponds to a specific rectangular region in the input space. Subsets of the training data are then constructed by selecting specific combinations of these groups. This method enables a fine-grained analysis of how the model learns from different regions of the input space, allowing for the systematic construction of task vectors. We sample 50 subsets with a sampling ratio 0.9, and use 40 subsets as the surrogate training set.

In-context learning. For in-context learning, we use the Qwen3-8B as the base model and evaluate it on a sentiment classification dataset, SST-2, and a reasoning task, the coin flip. The SST-2 dataset is a binary sentiment classification dataset composed of movie reviews labeled as either positive or negative from the GLUE benchmark. The number of queries is 450. The Coin-Flip dataset is an arithmetic reasoning task where the model reads a natural language description of a sequence of fair coin flips and must predict the final outcome (heads or tails). The number of queries is 869. For each dataset, we use the first 50 candidate demonstrations as the data to be attributed. We sample 200 subsets, each containing 4 prompts, and use 80 of these subsets as the surrogate training set.

Multi-objective reinforcement learning. We use MT10 from the Meta-World benchmark (Yu et al., 2020), which consists of 10 diverse robotic manipulation tasks. The agent's observation includes the environment state and a one-hot vector that specifies the current task. A sparse reward for moving the objective to its goal position. The 10 tasks in MT10 are: reach, push, pick-place, door-open, drawer-open, drawer-close, button-press-topdown, peg-insert-side, window-open, and box-open.

For the task attribution evaluation, we use the Soft Actor-Critic (SAC) as the training algorithm, and take the reward of each task as the $F(\mathbf{s})$. We sample 50 subsets, each containing 7 tasks, and use 40 subsets as the surrogate training set.

C.6 Baselines

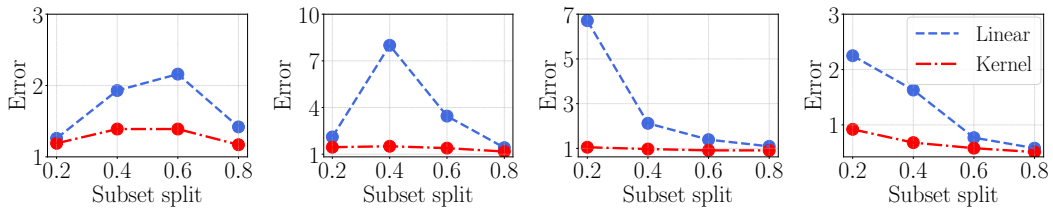
Influence functions. Influence functions (Koh & Liang, 2017), originally proposed for individual samples, can be adapted to estimate the influence of entire tasks by considering a task-weighted loss function. The influence of infinitesimally up-weighting task T_k on the test performance is given by the formula from the main text:

$$\mathcal{I}_k(\mathbf{s}) = \left[\nabla_W F(\mathbf{s})\right]^\top \left[\nabla_W^2 L(W, \mathbf{s})\right]^{-1} \nabla_W \left(\frac{s_k}{\sum_{j=1}^K s_j} \ell_k(f_W, T_k) \right).$$

Here, $H = \nabla_W^2 L(W, \mathbf{s})$ is the Hessian of the total training loss. In our implementation, we approximate the inverse Hessian-vector product $H^{-1}v$ (where v is the gradient of the task-specific loss term) by solving a Lasso regression problem, which avoids direct matrix inversion.

TracIn. TracIn (Pruthi et al., 2020) traces the influence of training data through the optimization trajectory, avoiding Hessian computation. We adapt this method from the sample level to the task level by measuring the correlation between task gradients over the course of training. The influence of a training task T_k on a test task T_{test} is approximated by summing the dot products of their respective loss gradients across various training checkpoints:

$$\text{TracIn}(T_k, T_{\text{test}}) = \sum_{t=1}^{T} \eta_t \nabla_W f_{\text{test}}^\top \nabla_W \ell_k(f_{W_t}, T_k),$$



(a) Modular addition (b) Modual quadratic (c) SST-2 (d) Coin flip

Figure 4: We investigated how the size of the surrogate training split affects the residual error of the linear and kernel models. We observed that across a range of training splits from 20% to 80%, the kernel model consistently yields a lower residual error than the linear model.

Table 6: Top positively and negatively influential prompt samples identified on the Coin flip task.

Query	**Top Positively Prompt Sample**	**Top Negatively Prompt Sample**
Q: A coin is heads up. kielmeyer **flips** the coin. jevgenij **does not flip** the coin. Is the coin still heads up? **(no)**	Q: A coin is heads up. erdener **flips** the coin. ismari **does not flip** the coin. Is the coin still heads up? **(no)**	Q: A coin is heads up. pachl *flips* the coin. lissett *flips* the coin. Is the coin still heads up? *(yes)*
Q: A coin is heads up. bonnitta **does not flip** the coin. ellise **does not flip** the coin. Is the coin still heads up? **(yes)**	Q: A coin is heads up. brittingham **does not flip** the coin. alilet **does not flip** the coin. Is the coin still heads up? **(yes)**	Q: A coin is heads up. kimyetta *flips* the coin. raynel **does not** flip the coin. Is the coin still heads up? *(no)*
Q: A coin is heads up. roeland **does not flip** the coin. joeliz **flips** the coin. Is the coin still heads up? **(no)**	Q: A coin is heads up. kulju **does not flip** the coin. afrodisio **flips** the coin. Is the coin still heads up? **(no)**	Q: A coin is heads up. pachl *flips* the coin. lissett **flips** the coin. Is the coin still heads up? *(yes)*

where W_t are the model parameters and η_t is the learning rate at checkpoint t.

TRAK. TRAK (Park et al., 2023) offers an efficient method for data attribution by linearizing the model and using random projections. We adapt it to attribute influence at the task level. The core idea is to represent each task by an average of its constituent samples' projected gradients. Specifically, for each sample z in a task, a feature vector is computed from the gradient of a model output function f_W. To get a feature vector for an entire task T_k, we average these features over all its samples. This task-level feature is then projected into a low-dimensional space using a random matrix $\mathbf{P}$. The attribution score for the test task T_{test} is then computed as: $\phi(T_{\text{test}})^\top(\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{Q}$. Here, $\phi(T_{\text{test}})$ is the projected feature vector for the test task, $\mathbf{\Phi}$ is the $K \times k$ matrix of stacked projected features for the K training tasks, and $\mathbf{Q}$ is a $K \times K$ diagonal weighting matrix. The final scores are averaged over an ensemble of models to ensure robustness.

Linear surrogate models. Linear surrogate models (Ilyas et al., 2022; Li et al., 2023) directly learn a linear mapping from data weights to model predictions. Given a set of models trained on different data subsets, this approach fits a linear function $f : \{0,1\}^n \to \mathbb{R}$ that predicts test performance from training set inclusion indicators. The coefficients of this linear model serve as attribution scores, capturing the marginal contribution of each training example across multiple training runs.

C.7 Qualitative Results

In Table 6, we present the top positively and negatively prompt samples obtained by KernelSM on the Coin flip dataset. We use **bold** to show information that is the same as the query, and *italics* for information that is different from the query.

C.8 Extensions

Meta-learning. Model-agnostic meta-learning (MAML) seeks to learn a model initialization that is optimized for rapid adaptation to new tasks, typically with only a few gradient steps. Our work on

task attribution, which measures a model's sensitivity to its training data, is connected to this goal. The error of first-order attribution methods is directly governed by the Hessian. For attribution, a large Hessian signifies a highly curved, non-linear performance landscape where linear approximations are unreliable, leading to significant attribution error. In contrast, MAML leverages this same curvature as a signal. The meta-gradient update in MAML involves differentiating through an inner-loop gradient step, a calculation that explicitly depends on the Hessian of the inner-loop loss with respect to the model parameters. There also exists a mathematical parallel between the derivation of influence functions and the formulation of implicit MAML (iMAML) (Rajeswaran et al., 2019). Influence functions can be derived by applying the implicit function theorem to the first-order optimality condition of the perturbed loss, yielding an expression for the parameter change that involves the inverse Hessian. Analogously, iMAML uses the implicit function theorem to derive an analytical expression for the meta-gradient that depends only on the solution of the inner optimization, not the path taken to reach it.

Multi-group learning. Multi-group learning aims to train a single predictor that performs robustly across a predefined set of subgroups, addressing the common failure mode where high average accuracy masks poor performance on critical subpopulations (Deng & Hsu, 2024). The influence of task k on the performance of a model evaluated on task j can be directly interpreted as the marginal contribution of group k's training data to the model's performance on group j. A positive influence value for a loss-based metric provides a clear signal of negative transfer. Standard influence functions, being first-order approximations, capture pairwise, additive effects and struggle to model more complex issues. For example, a model may perform poorly on an intersectional group (e.g., women of a specific race) due to higher-order interactions between the data subgroups that linear methods cannot detect. By using a non-linear RBF kernel, our model learns a global function that captures combinatorial effects between groups. This can also inform the design of robust learning algorithms. For example, the MGL-Tree algorithm for hierarchical groups decides whether to use a general parent-level predictor or a specialized child-level predictor by comparing their empirical risks. Our method could provide a more principled, causal signal to guide this choice.