Scalable Fine-tuning from Multiple Data Sources: A First-Order Approximation Approach

Dongyue Li[†*] **Ziniu Zhang**[†*] **Lu Wang**[‡] **Hongyang R. Zhang**[†]
[†]Northeastern University, Boston, MA
[‡]University of Michigan, Ann Arbor, MI

Abstract

We study the problem of fine-tuning a language model (LM) for a target task by optimally using the information from n auxiliary tasks. This problem has broad applications in NLP, such as targeted instruction tuning and data selection in chain-of-thought fine-tuning. The key challenge of this problem is that not all auxiliary tasks are useful to improve the performance of the target task. Thus, choosing the right subset of auxiliary tasks is crucial. Conventional subset selection methods, such as forward & backward selection, are unsuitable for LM fine-tuning because they require repeated training on subsets of auxiliary tasks. This paper introduces a new algorithm to estimate model fine-tuning performances without repeated training. Our algorithm first performs multitask training using the data of all the tasks to obtain a meta initialization. Then, we approximate the model fine-tuning loss of a subset using functional values and gradients from the meta initialization. Empirically, we find that this gradient-based approximation holds with remarkable accuracy for twelve transformer-based LMs. Thus, we can now estimate fine-tuning performances on CPUs within a few seconds. We conduct extensive experiments to validate our approach, delivering a speedup of $30\times$ over conventional subset selection while incurring only 1% error of the true fine-tuning performances. In downstream evaluations of instruction tuning and chain-of-thought fine-tuning, our approach improves over prior methods that utilize gradient or representation similarity for subset selection by up to 3.8%.

1 Introduction

Fine-tuning a language model (LM) has emerged as an effective approach for knowledge transfer on text data. As the scale of LMs keeps growing, efficient and scalable fine-tuning methods are in great demand. For instance, parameter-efficient fine-tuning with adapters or low-rank parameterization can significantly reduce the memory usage of fine-tuning (Houlsby et al., 2019; Pfeiffer et al., 2020; Hu et al., 2021). In many applications, besides solving a target task of interest, one can access several related data sources, which could be used for data augmentation. Examples include multilingual systems such as neural machine translation (Neubig and Hu, 2018), parsing (Üstün et al., 2020), data selection (Xie et al., 2023), and targeted instruction tuning (Xia et al., 2024). A fundamental issue in these scenarios is selecting the beneficial data sources amongst all the available data sources, which can be formulated as a *subset selection* problem. In this paper, we develop efficient algorithms for subset selection in LM fine-tuning given n data sources, which can scale up to handle large numbers of n.

Classical subset selection methods, such as forward & backward stepwise selection, have been very effective for regression analysis practice (Hastie et al., 2009). However, these classical methods are unsuitable for LM fine-tuning because they require repeated training of the base LM on many subsets of tasks, which is not feasible for large n. More recently, data selection methods have been proposed based on selecting influential samples by comparing gradients (Xia et al., 2024) or feature representation during training (Ivison et al., 2023). These methods depend on the training procedure, leading to noisy outcomes for approximating true fine-tuning performances. In summary, existing subset selection methods require expensive computation or rely on noisy measurements of task-relatedness (Ruder, 2017; Vu et al., 2020, 2022; Zhang et al., 2023).

This paper presents a new approach to scale up subset selection by quickly estimating model fine-tuning losses without running the fine-tuning procedure. The key idea is first to perform multitask

[*]Equal Contribution. Emails: {li.dongyu, zhang.zini, ho.zhang}@northeastern.edu and wangluxy@umich.edu.

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 5608–5623
November 12-16, 2024 ©2024 Association for Computational Linguistics

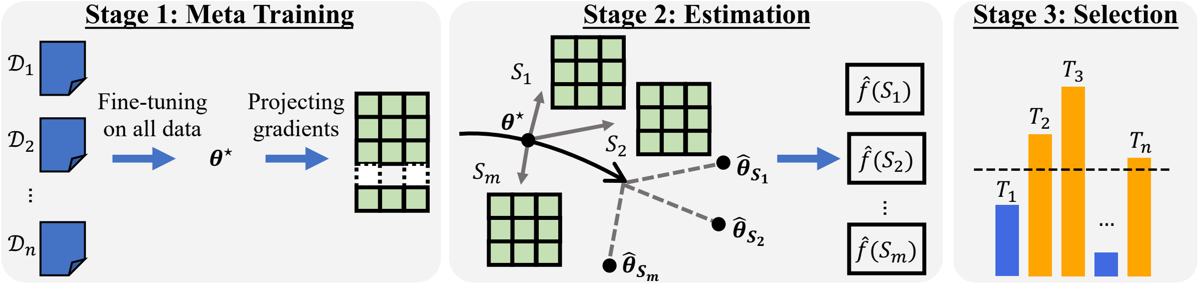


Figure 1: An overview of our approach: **(1)** Perform multitask training on a base LM using all the samples, leading to a meta-initialization $\theta^\star \in \mathbb{R}^p$. We store (randomly) projected gradients of $\theta^\star$ for every sample. **(2)** Estimate model fine-tuning performances on a list of task subsets using projected gradients as features in logistic regression. **(3)** Using the estimated results (denoted as $\hat{f}(S_1), \dots, \hat{f}(S_m)$), compute a score T_i for each auxiliary task for $i = 1, 2, \dots, n$, which indicates its relevance to the target task. Select a subset (depicted in blue color) using a threshold score and combine their data for training jointly with the target task.

training on the samples of all the tasks, leading to a *meta initialization* $\theta^\star$ (Finn et al., 2017). From $\theta^\star$, we compute all the samples' functional values and gradients, which can be used during inference. Second, given a subset of auxiliary tasks, we apply Taylor's expansion on the loss and approximate the loss value with the function values and the gradients of the samples in the subset. After applying this approximation, we will estimate the fine-tuning losses with logistic regression. Notice that this computation can be done on CPUs, and we use random projections to reduce the dimension of the regression down to a few hundred. Using this procedure, we can estimate the fine-tuning performance for each subset in just a few seconds, which is significantly faster than full-model fine-tuning. Importantly, the accuracy of this approach hinges on the quality of the first-order approximation. We find that across twelve transformer-based LMs, including Llama-3-8B, the approximation error of the gradient-based expansion is at the order of 10^{-5} to 10^{-3}, in regions close to $\theta^\star$. Based on this estimation, we can apply a classical subset selection method, such as forward selection, to every subset encountered during the procedure. In summary, our approach has three stages. See Figure 1 for an illustration.

We conduct extensive experiments to validate our approach using four benchmarks. We show that GRADEX can accurately approximate fine-tuned model losses within **1**% errors across various LMs. Meanwhile, GRADEX can accelerate forward selection with **30**× fewer FLOPs and **25**× less GPU hours by circumventing full model fine-tuning. GRADEX also speeds up random ensemble with **44**× fewer FLOPs and **46**× less GPU hours. We consider targeted instruction tuning and chain-of-thought fine-tuning for downstream evaluations of GRADEX. Our approach matches the accuracy of classical subset selection methods with full fine-tuning while costing only **0.5**% of the computation. On the ToxiGen and TruthfulQA benchmarks, GRADEX outperforms existing selection methods that rely on gradient or representation measurements (Xie et al., 2023; Xia et al., 2024) by **3.8**% on average. On StrategyQA and CommonsenseQA datasets, GRADEX improves chain-of-thought reasoning accuracy by **2.4**% over prior methods.

In conclusion, this paper introduces a new algorithm for subset selection in LM fine-tuning given multiple data sources. The algorithm can quickly estimate model fine-tuning performances without performing fine-tuning on each subset. We identify a linearization property of LMs (after meta-training), which is empirically verified on twelve transformer-based LMs. This empirical finding may be relevant for future developments of fast inference methods of LLMs. Our estimation algorithm unlocks classical subset selection methods such as forward & backward selection to LMs. Our extensive experiments show that our approach can deliver significant speedups without losing downstream accuracy. In particular, our approach outperforms recent subset selection methods for instruction tuning and chain-of-thought fine-tuning. The code repository and detailed instructions for reproducing our findings are available at `https://github.com/VirtuosoResearch/Scalable-finetuning`.

2 Problem Formulation

Problem setup: Consider fine-tuning a language model to solve a target task. Suppose we also have access to n source tasks. Because the information from different data sources may conflict with each other or hurt the target performance (Wu et al., 2020), we would like to select a subset of the n tasks. This often happens in transfer learning, for instance, working with low-resource neural machine translation (Zoph et al., 2016). Because there are 2^n possible choices of subsets, performing the best subset selection is very costly.

In this paper, we develop a much faster approach for subset selection that is suitable for running on LMs. Concretely, given a base fine-tuning procedure on top of an LM (such as LoRA or QLoRA (Dettmers et al., 2023)), let $f(S)$ be the model fine-tuning loss on the target task trained together with a given subset of auxiliary tasks $S \subseteq \{1, 2, \dots, n\}$. A lower value of f indicates that S is more relevant to the target task. Best subset selection corresponds to finding a subset that minimizes $f(S)$.

2.1 Prior approaches for measuring $f(S)$

Experimental setup: We will present a case study using the Alpaca dataset (Taori et al., 2023), which involves $52,000$ instruction-following data generated by GPT-3, each containing an instruction as a task description and the corresponding task input and output. The dataset can be pre-processed into 38 tasks based on the instruction types (Wang et al., 2023b). Each task is identified by the verb in the instruction, such as "edit" and "describe." We randomly sample 10% as the validation set. We use TinyLlama-1.1B as the base LM and LoRA as the base protocol (Hu et al., 2021).

$\diamondsuit$ First, we find the existence of negative transfers, which remains consistent with prior studies of multitask learning in NLP (Vu et al., 2020, 2022). Specifically, we compare (A1) fine-tuning the base LM on each task and (A2) fine-tuning the base LM on each task with *one of* the remaining 37 tasks as auxiliary tasks. For $18/38$ tasks, A1 performs better than A2, indicating negative transfers from auxiliary tasks to the target task.

$\diamondsuit$ Second, we also find that $f(S)$ is not monotone, in the sense that adding an extra task to S may not necessarily reduce the value of f (although it does more data). We pick a target task t corresponding to "arrange" in Alpaca and find that $f(S)$ starts to increase after we add more than one task to S, which confirms our hypothesis. This suggests directly optimizing f over the space of subsets remains challenging because f is complex.

$\diamondsuit$ Third, we also find that methods that utilize model gradients (Fifty et al., 2021) or representation similarity (Wu et al., 2020; Vu et al., 2020) to select tasks do not correlate well with actual values of f. For every pair of tasks i and j, we evaluate the correlation between the cosine similarity of the averaged gradient of tasks i, j, and $f(\{i, j\})$. We find it is less than 0.2.

2.2 Problem statement

Having described the nuances of f and the potential complexity in optimizing f directly (see also Zhang et al. (2023) for further discussion on this topic), we now pose the following research question: *Given a list of subsets $S_1, \dots, S_m \subseteq \{1, \dots, n\}$, can we efficiently estimate $f(S_1), \dots, f(S_m)$, without fine-tuning the model on each possible subset?* Next, we give two examples to illustrate the choices of subsets.

Example 2.1 (Forward stepwise selection)**.** In forward selection (Hastie et al., 2009), one starts with an empty subset $S_1 = \{\}$. Then, enumerate through all singleton sets, $f(\{1\}), \dots, f(\{n\})$, and pick the best one. Suppose i_1 is chosen. Then, evaluate $f(\{i_1, 1\}), \dots, f(\{i_1, n\})$ except when i_1 is repeated, and pick the best one. Until f peaks.

Example 2.2 (Random ensemble)**.** Random ensembling is a highly effective approach for data attribution (Ilyas et al., 2022), which can also be used for subset selection in multitask learning (Li et al., 2023b,a, 2024b). We choose m random subsets from $\{1, 2, \dots, n\}$ with a fixed size α. For example, if $\alpha = 2$, f measures pairwise affinity scores between two tasks (Fifty et al., 2021). If $\alpha > 2$, $f(S_i)$ measures higher-order affinity scores between multiple tasks (Li et al., 2023a).

If we can quickly estimate the values of $f(S_1), f(S_2), \dots, f(S_m)$, then we can still apply the above subset selection methods. In the next section, we present our estimation approach.

3 Our Approach

Our algorithm involves two stages. First, we run a meta-training procedure to obtain an initialization $\theta^\star$ by fine-tuning an LM on all the samples, similar to performing multitask learning (Wang et al., 2018). Second, we estimate model fine-tuning losses by solving a logistic regression problem to get $\hat{\theta}_{S_i}$ for every $i = 1, 2, \dots, m$. Notably, the second estimation stage can be run entirely on CPUs, which will be very fast.

The key idea of why this works is a first-order approximation property that we have found empiri-

Table 1: We empirically find that the *first-order approximation holds with very high accuracy within 10% relative distance to the meta initialization $\theta^\star$*. As shown in the table below, the RRSS is at the order of 10^{-5}-10^{-3} when X is between 2%-10% distance of $\theta^\star$. We attribute this behavior to the highly overparameterized nature of LLMs. We report the average over 50 random task subsets to ensure statistical significance. The reference for each LM can be found in Appendix D.

Dist.	**Pythia-70M**	**BERT-Base**	**RoBERTa-Base**	**GPT-2**	**FLAN-T5-Base**	**BloomZ-560M**
2%	$9_{\pm 1.4} \times 10^{-4}$	$3_{\pm 0.2} \times 10^{-4}$	$4_{\pm 0.4} \times 10^{-4}$	$2_{\pm 0.2} \times 10^{-4}$	$1_{\pm 0.2} \times 10^{-4}$	$2_{\pm 0.4} \times 10^{-4}$
4%	$1_{\pm 0.2} \times 10^{-4}$	$5_{\pm 0.9} \times 10^{-4}$	$5_{\pm 0.5} \times 10^{-4}$	$6_{\pm 0.7} \times 10^{-4}$	$8_{\pm 0.4} \times 10^{-4}$	$5_{\pm 1.5} \times 10^{-4}$
6%	$3_{\pm 0.3} \times 10^{-4}$	$9_{\pm 1.0} \times 10^{-4}$	$6_{\pm 0.9} \times 10^{-4}$	$8_{\pm 0.6} \times 10^{-4}$	$2_{\pm 0.3} \times 10^{-4}$	$9_{\pm 0.5} \times 10^{-4}$
8%	$4_{\pm 0.9} \times 10^{-4}$	$3_{\pm 0.4} \times 10^{-3}$	$9_{\pm 1.3} \times 10^{-4}$	$3_{\pm 0.3} \times 10^{-3}$	$3_{\pm 0.6} \times 10^{-4}$	$7_{\pm 0.6} \times 10^{-4}$
10%	$7_{\pm 1.4} \times 10^{-3}$	$5_{\pm 1.4} \times 10^{-3}$	$5_{\pm 0.5} \times 10^{-3}$	$5_{\pm 0.4} \times 10^{-3}$	$5_{\pm 1.2} \times 10^{-3}$	$5_{\pm 2.2} \times 10^{-3}$

Dist.	**TinyLlama-1.1B**	**GPT-Neo-1.3B**	**OPT-1.3B**	**Gemma-2-2B**	**Mistral-7B**	**Llama-3-8B**
2%	$6_{\pm 0.5} \times 10^{-5}$	$3_{\pm 0.4} \times 10^{-5}$	$7_{\pm 0.2} \times 10^{-5}$	$4_{\pm 0.3} \times 10^{-5}$	$9_{\pm 0.5} \times 10^{-5}$	$3_{\pm 0.3} \times 10^{-5}$
4%	$1_{\pm 0.3} \times 10^{-4}$	$3_{\pm 0.3} \times 10^{-4}$	$7_{\pm 1.0} \times 10^{-5}$	$2_{\pm 0.1} \times 10^{-4}$	$2_{\pm 0.2} \times 10^{-4}$	$6_{\pm 0.9} \times 10^{-5}$
6%	$3_{\pm 0.7} \times 10^{-4}$	$6_{\pm 0.6} \times 10^{-4}$	$8_{\pm 0.2} \times 10^{-5}$	$3_{\pm 0.4} \times 10^{-4}$	$3_{\pm 0.4} \times 10^{-4}$	$3_{\pm 0.6} \times 10^{-4}$
8%	$4_{\pm 0.9} \times 10^{-4}$	$1_{\pm 0.2} \times 10^{-3}$	$1_{\pm 0.1} \times 10^{-4}$	$8_{\pm 0.7} \times 10^{-4}$	$4_{\pm 0.2} \times 10^{-4}$	$4_{\pm 0.4} \times 10^{-4}$
10%	$5_{\pm 0.8} \times 10^{-3}$	$5_{\pm 0.8} \times 10^{-3}$	$5_{\pm 0.1} \times 10^{-3}$	$4_{\pm 0.5} \times 10^{-3}$	$4_{\pm 0.1} \times 10^{-3}$	$5_{\pm 0.4} \times 10^{-4}$

cally around the initialization LM. The intuition is that for a highly over-parameterized network, the geometry around a local minimum solution tends to be flat (Zhang et al., 2024), leading fine-tuning to behave like kernel regression locally (Malladi et al., 2023). To aid this approximation, we hypothesize that after meta-training on all tasks, this initialization can adapt quickly to subsets of tasks. This has also been observed in MAML (Finn et al., 2017), as depicted in Figure 1. The difference is that we further utilize the first-order approximation of large language models after meta-training. We have empirically observed that the linearization property holds across twelve LMs, as described next.

3.1 Multitask training on all tasks

Let the output of a network be $h_X(s, y)$, where s is an input (e.g., sentence) and y is the prediction label. For example, in binary classification, h_X is the log loss. Here, $X \in \mathbb{R}^p$ denotes the trainable parameters of all layers, while $\theta^\star \in \mathbb{R}^p$ denotes trained parameters.

Note that $f(S)$ is equal to the averaged $h_X(s, y)$ over a set of samples (s, y). We examine first-order Taylor's expansion of $h_X(s, y)$ centered at $\theta^\star$:

$$h_X(s,y) \approx h_{\theta^\star}(s,y) + [\nabla_X h_{\theta^\star}(s,y)]^\top (X - \theta^\star) \\ + \epsilon.$$

Our key observation is that ϵ remains negligible for X around $\theta^\star$. We report empirical measurements of ϵ across twelve LMs, evaluated on the GLUE benchmark (with $n = 9$ tasks) for BERT and RoBERTa, and the Alpaca dataset with $n = 38$ tasks for the rest of ten LMs. We use LoRA as the base fine-tuning procedure and see similar results with full fine-tuning. We compute the relative residual sum of squares (RRSS):

$$\frac{\Big(h_X(s,y) - h_{\theta^\star}(s,y) - \nabla_X h_{\theta^\star}(s,y)^\top (X - \theta^\star)\Big)^2}{h_X(s,y)^2}.$$

Table 1 reports the results, averaged over 50 randomly sampled subsets with fixed sizes (3 for GLUE and 19 for Alpaca). Remarkably, across a range of relative distance values (between X and $\theta^\star$) from 2% to 10%, the RRSS is at the range of 10^{-5} to 10^{-3}.

3.2 Fast estimation for each task subset

Next, we describe the inference of model fine-tuning performances on each subset S_i. Importantly, we will achieve this using the functional values and the projected gradients obtained at the end of the first stage *without* performing fine-tuning.

We illustrate the idea in binary classification, where y is 1 or -1. While the same works for multi-class and generative tasks. Consider the log-loss for binary classification:

$$\ell(X) = \log\left(1 + \exp\left(-y \cdot h_X(s,y)\right)\right).$$

Our key idea is to replace $h_X(s, y)$ above using $h_{\theta^\star}(s, y)$ plus the first-order term, and this is valid as long as ϵ is negligible, which is generally true for fine-tuning since X will be close to $\theta^\star$. Let $b_s = -y \cdot h_{\theta^\star}(s, y)$ and let $g_s = \nabla h_{\theta^\star}(s, y)$. We can approximate $\ell(X)$ with

$$\hat{\ell}(X) = \log\Big(1 + \exp\Big(b_s - y \cdot g_s^\top (X - \theta^\star)\Big)\Big).$$

Algorithm 1 GRADEX: FAST ESTIMATION of LM Fine-Tuning Losses Using **Grad**i**e**nts

```
Input: $n$ training sets; $m$ subsets $S_1, S_2, \dots, S_m$ of $\{1, 2, \dots, n\}$; a validation set of the target task
Require: LM $h_{\theta_0}$; Projection dimension $d$
/*                Stage 1: Meta-training                */
$\theta^\star \leftarrow$ fine-tune $h_{\theta_0}$ on $\mathcal{D}_{\{1,2,\dots,n\}}$
$P \leftarrow p$ by $d$ isotropic Gaussian random matrix
for $(s, y) \in \mathcal{D}_{\{1,2,\dots,n\}}$ do
    $\tilde{g} \leftarrow P^\top \nabla h_{\theta^\star}(s, y)$   ▷ project the gradient
    $b \leftarrow -y \cdot h_{\theta^\star}(s, y)$
end for
/*                Stage 2: Estimation                */
for $i \leftarrow 1, \dots, m$ do
    $\hat{X}_d \leftarrow \min \hat{L}_{S_i}(X)$ with $\mathcal{D}_{S_i}$
    $\hat{\theta}_{S_i} \leftarrow \theta^\star + P\hat{X}_d$
    $\hat{f}(S_i) \leftarrow$ evaluate $h_{\hat{\theta}_{S_i}}$ on the target val set
end for
Return $\hat{f}(S_i)$, for every $i = 1, 2, \dots, m$
```

For a subset $S \subseteq \{1, 2, \dots, n\}$, let $\mathcal{D}_S$ denote the combined samples from all of S, and let n_S denote the total number of samples in $\mathcal{D}_S$. We estimate the model fine-tuning loss by minimizing the averaged $\hat{\ell}(X)$ over $X \in \mathbb{R}^p$:

$$\frac{1}{n_S} \sum_{(s,y)\in\mathcal{D}_S} \log\left(1 + \exp(b_s - y g_s^\top (X - \theta^\star))\right).$$

Denote the above as $\hat{L}_S(X)$, which varies by S. Let $\hat{\theta}_S \in \mathbb{R}^p$ be the solution from minimizing $\hat{L}_S(X)$. In practice, the dimension of X can be huge; thus, we apply random projection to reduce dimension, which can provably preserve the accuracy of the regression problem through the Johnson-Lindenstrauss lemma (Johnson, 1984).

We sample a p by d (e.g., for $d = 100$) Gaussian random matrix P with each entry drawn from $N(0, d^{-1})$ and project the gradient as $P^\top h_{\theta^\star}(s, y)$. We insert the projected gradient as $\tilde{g}$ into $\hat{\ell}(X)$. Then, after solving the regression problem in dimension d, we map the minimizer $\hat{X}_d$ from dimension d to p using $P\hat{X}_d$. This step only takes *a few seconds*, which is extremely fast and is much faster than full fine-tuning, since it does not compute the gradient as both b and g are already computed after meta-training. Thus, this step will work using CPUs. Algorithm 1 summarizes our procedure.

3.3 Accelerating subset selection for LMs

We now describe several use cases of the estimation algorithm, expanding on Examples 2.1 and 2.2.

GRADEX-FS: During forward selection, we apply GRADEX to every subset encountered in the selection procedure. After performing subset selection, we output the subset and use their data for augmentation.

GRADEX-RE: For random ensembles, we first get a list of estimates $\hat{f}(S_1), \hat{f}(S_2), \dots, \hat{f}(S_m)$ for every random subset. Then, compute a score T_i for each task i as the averaged result over all subsets that include i:

$$T_i = \frac{1}{n_i} \sum_{1\le k\le m:\ i\in S_k} \hat{f}(S_k), \text{ for } 1 \le i \le n, \quad (1)$$

where n_i is the number of subsets that include i. Then, select a subset $\{i \mid T_i < \gamma, i = 1, \dots, n\}$ using a threshold γ adjusted via cross-validation.

GRADEX-DS: In data selection, one would like to pick a data subset from a collection of raw data. To apply our technique, one can preprocess the raw data by clustering it into n groups. Then, use the above procedures to choose a subset from the n groups.

Examples. We illustrate our approach in a noisy addition example. Consider adding two digits of length 5:

IN $\underline{6\,7\,0\,1\,3}$ + $\underline{2\,3\,9\,2\,4}$
OUT $0\,\underline{7}$ | $0\,\underline{3\,7}$ | $0\,\underline{9\,3\,7}$ | $1\,\underline{0\,9\,3\,7}$ | $\underline{9\,0\,9\,3\,7}$

One can write down the intermediate calculations plus carry bits. We thus generate a synthetic addition set by generating input pairs of length 5 between 0 and 9. We create 10 groups; five are correct, while the rest involve randomly generated output digits. Then, we aim to separate the noisy groups from the correctly-labeled groups.

We apply GRADEX-RE to train a GPT-2 transformer network. For comparison, we also report the results from measuring n-gram features (Xie et al., 2023), feature similarities inside the transformer (Ivison et al., 2023), and gradient similarities of the fine-tuned model (Xia et al., 2024). We report our findings in Figure 2, showing that our procedure can lead to a better separation than the baseline measurements.

3.4 Comparison with prior approaches

We discuss the memory and runtime complexity of our algorithm. Regarding memory requirement,

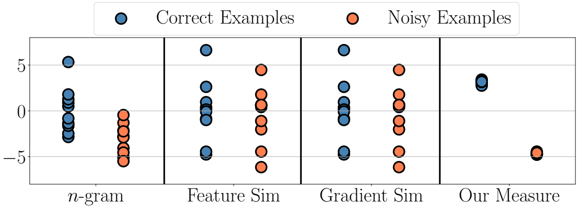


Figure 2: Illustrating the separation between correctly-labeled and noisy examples, using estimated values of T in equation (1), compared to several existing measures. Our measure from T (rightmost) leads to a much more clear separation of correct vs. noisy examples compared to three measures.

our algorithm matches the base fine-tuning method and inherits the same number of trainable parameters as LoRA. The runtime of our algorithm involves meta-training on n tasks, computing the gradients of all the n tasks, and solving logistic regression in dimension d on m subsets.

In particular, our algorithm reduces m model fine-tuning runs to a single meta-model training. As mentioned earlier, the additional estimation stage incurs very little overhead; after dimension reduction, solving each logistic regression problem takes less than 2 seconds per task subset. This estimation stage takes less than 10% of the total computation cost in our overall procedure. As for the meta-training stage, our algorithm uses comparable computational costs to existing data selection methods such as Xie et al. (2023) and Xia et al. (2024). Table 2 summarizes this comparison, including the number of forward passes required by each approach.

Methods	**Runtime**	**# Forward**
Forward Selection (FS)	$O(n^3)$	$\frac{1}{6}n^3$
Random Ensemble (RE)	$O(n \log n)$	$\alpha n \log n$
DSIR (Xie et al., 2023)	$O(n)$	n
DEFT (Ivison et al., 2023)	$O(n)$	$3n$
LESS (Xia et al., 2024)	$O(n)$	$3n$
GRADEX-FS	$O(n)$	$3n$
GRADEX-RE	$O(n)$	$3n$

Table 2: Summary of runtime complexity between our algorithm and existing solutions for subset selection, as a function of the number of data sources n. We describe the constants in terms of forward passes each method takes. For every method, the number of backward passes equals the forward passes. Here, α denotes the size of subsets sampled in random ensembles. Note that the number of forward passes is for one training step; for the calculation, see Appendix C.

4 Experiments

We now validate GRADEX and its use cases across various datasets and models, focusing on the following key questions. Does the estimation procedure accurately approximate the true model fine-tuning losses? How much computational cost does it save relative to classical subset selection methods that require repeated model fine-tuning? How effective are the subset selection methods using the estimation results in downstream evaluation?

Our experiments show that GRADEX approximates fine-tuned model losses within **1**% error, tested on five different LMs including Llama-3-8B. GRADEX reduces the number of FLOPs by up to **43**× and GPU hours by **46**× compared to conventional subset selection. Next, we evaluate GRADEX for *instruction tuning* and *chain-of-thought fine-tuning*. In both cases, our algorithm performs on par with conventional subset selection that uses full fine-tuing results, while incurring less than **0.5**% computation costs. Our algorithm outperforms existing selection methods (Xie et al., 2023; Xia et al., 2024) on ToxiGen by **3.8**% and TruthfulQA by **2.4**%, while using comparable computation costs.

4.1 Experimental setup

Our algorithm is broadly applicable to estimating model fine-tuning performances; We focus on instruction tuning and chain-of-thought fine-tuning in this section. For instruction tuning, given a set of source tasks and a target task, our goal is to select source tasks relevant to the target task. For chain-of-thought fine-tuning, we fine-tune an LM to generate chain-of-thought reasoning steps and answers to answer a question. Several explanations are possible, while some of them are incorrect. We aim to select explanations pertinent to the reasoning task using subset selection.

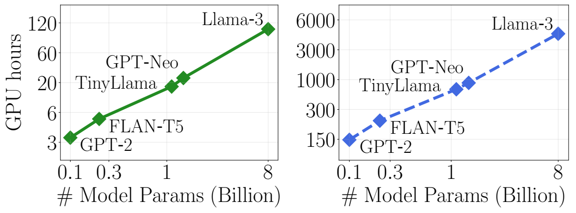
For *instruction tuning*, we use three datasets, including FLAN V2 with 1,691 tasks, Chain of Thoughts with 18 tasks, and Alpaca with 38 tasks. These datasets encompass over 150 task categories. See Appendix D for the statistics.

For *chain-of-thought fine-tuning*, we use CommonSenseQA (Talmor et al., 2019) and StrategyQA (Geva et al., 2021). The chain-of-thought explanations are generated with a GPT-3 175B model. We sample 10% of the data for evaluating $f(S)$.

We set LoRA as the base protocol. We adjust the rank parameter of LoRA between 16, 32, 64, and 128. For chain-of-thought fine-tuning, we partition

Table 3: We report the relative RSS between $\hat{f}(S)$ and $f(S)$, measured on Alpaca and StrategyQA. For measuring speedup, we report the ratio of the number of FLOPs required for computing $f(S)$ divided by GRADEX. The speedup remains the same across different models for the same dataset, since the speed-up stems from a reduced number of forward/backward passes.

Alpaca	**GPT-2**	**FLAN-T5-Base**	**TinyLlama-1.1B**	**GPT-Neo-1.3B**	**Llama-3-8B**	**Speedup**
GRADEX-FS	7.4×10^{-4}	3.2×10^{-4}	3.9×10^{-4}	3.5×10^{-4}	2.7×10^{-4}	17.6×
GRADEX-RE	8.7×10^{-4}	3.4×10^{-4}	4.2×10^{-4}	3.8×10^{-4}	2.9×10^{-4}	43.3×
StrategyQA	**GPT-2**	**FLAN-T5-Base**	**TinyLlama-1.1B**	**GPT-Neo-1.3B**	**Llama-3-8B**	**Speedup**
GRADEX-FS	7.4×10^{-3}	3.0×10^{-4}	3.2×10^{-4}	3.0×10^{-4}	2.4×10^{-4}	30.5×
GRADEX-RE	8.9×10^{-3}	3.6×10^{-4}	3.8×10^{-4}	3.4×10^{-4}	2.8×10^{-4}	44.8×



(a) Our estimation (GRADEX) (b) Full fine-tuning

Figure 3: Number of GPU hours as the number of model parameters between our estimation approach (left, 3a) and full fine-tuning (right, 3b). We estimate the full fine-tuning cost by fine-tuning on randomly sampled 100 subsets of tasks.

the data into 100 groups using spectral clustering on the gradient similarity matrix. For random ensembles, we sample 1000 subsets, each containing 75% of all tasks, to ensure the T scores (cf. equation (1)) have converged.

4.2 Approximating model fine-tuning losses

We now assess the accuracy of GRADEX. We measure the relative error between the estimated $\hat{f}(S)$ and the true $f(S)$:

$$\frac{1}{m}\sum_{i=1}^{m}\frac{(f(S_i)-\hat{f}(S_i))^2}{(f(S_i))^2}.$$

We obtain $f(S)$ by fine-tuning a pretrained LM on the samples of S (along with the target task). We measure computation cost through the total number of FLoating-point OPerations (FLOPs), and the number of GPU hours (measured on a desktop with three Nvidia RTX6000 GPUs).

We evaluate the relative distance on both Alpaca and StrategyQA using five different LMs listed in Table 3. Due to computation constraints, for models with more than 1 billion parameters, we sample 100 subsets to estimate approximation error for random ensemble.

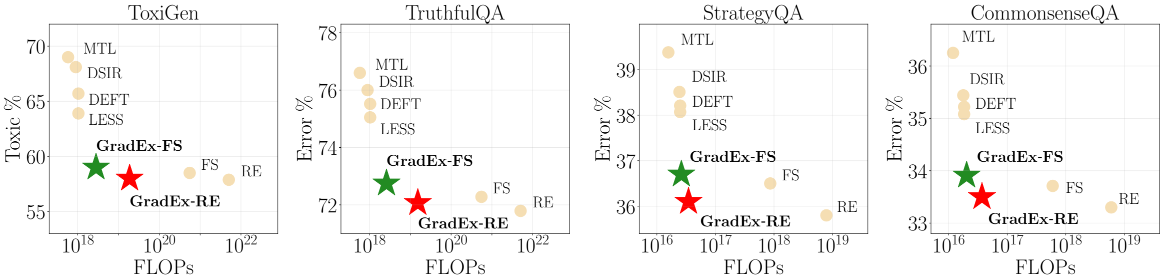
We find that our algorithm approximates the fine-tuned model losses within **1**% error across the five LMs. Furthermore, the approximation quality is generally better for larger models.

On Alpaca, we find that GRADEX-FS uses 117 GPU hours for Llama-3-8B; this is **17.6**× less computation compared to running forward selection with full fine-tuning. As for random ensembles, GRADEX-RE uses 120 GPU hours for Llama-3-8B and **43.3**× less computation compared to full fine-tuning. We also observe qualitatively similar results when measured on StrategyQA: GRADEX achieves **30.5**× and **44.8**× less computation compared to full fine-tuning in forward selection and random ensembles, respectively.

We note that the speedup remains consistent across different models when applied to the same dataset. Figure 3 illustrates the number of GPU hours used by GRADEX when running random ensembles on five LMs on Alpaca (left, 4a) vs. full fine-tuning (right, 4b).

Projection dimension: Recall that we project the dimension of the gradients down to a much smaller dimension during the second estimation stage. We vary the projection dimension d between 50, 100, 200, and 400 for testing GRADEX on FLAN-T5-Base. We observe that once d increases above 100, the error stabilizes around 0.03%. Hence, we set d as 100 for all the experiments. With $d = 100$, solving a logistic regression problem for one subset takes less than 2 seconds.

Reducing overfitting in meta-training: Recall that in the meta-training stage, we apply multitask training to the combined samples of all the tasks. We conduct a preliminary experiment, where we use a sharpness-reduced training algorithm (i.e., sharpness-aware minimization (Foret et al., 2021; Zhang et al., 2024)) for multitask training (in place of SGD). We find that this can reduce the approxi-



(a) Task selection for instruction tuning ($n = 1,729$)

(b) Data selection for chain-of-thought fine-tuning ($n = 100$)

Figure 4: Illustration of the tradeoff between the number of FLOPs (computation cost) and test error rate, measured on our methods and six baseline methods. *Our approach delivers comparable downstream performance to conventional subset selection methods*, while using less than **0.5**% of total computation. Furthermore, our approach achieves a **3.1**% performance improvement over existing data selection methods using comparable computation costs. The comparison of GPU hours is qualitatively similar and can be found in Appendix D.3.

mation error (relative to full fine-tuning results) by 23%. Further applying multitask and meta-learning techniques to improve GRADEX would be an interesting question for future work.

4.3 Downstream evaluation for fine-tuning

Baselines: We compare our algorithms to forward selection (FS) and random ensemble (RE) with full fine-tuning. Additionally, we compare with four baseline methods, including fine-tuning a single model on all tasks with LoRA (MTL), Data Selection with Importance Resampling (DSIR) (Xie et al., 2023), Data Efficient Fine-Tuning with cross-task nearest neighbors clustering based on feature similarity (DEFT) (Ivison et al., 2023), and Low-rank gradiEnt Similarity Search (LESS) (Xia et al., 2024). For each baseline, the performance is reported based on fine-tuning a pretrained LM with the same amount of trainable parameters on the selected subset of tasks.

Evaluation metrics: For instruction tuning, we follow the protocol of Wang et al. (2023a) to evaluate the toxicity and truthfulness of fine-tuned models in ToxiGen (Hartvigsen et al., 2022) and TruthfulQA (Lin et al., 2022), respectively. For ToxiGen, we measure the percentage of toxic outputs generated by the model. For TruthfulQA, we evaluate model accuracy for identifying truthful statements. For chain-of-thought fine-tuning, we measure the accuracy of two reasoning tasks.

4.3.1 Results for instruction tuning

We illustrate the results of applying GRADEX-FS and GRADEX-RE to select tasks in instruction tuning in Figure 4a, using the TinyLlama-1.1B model. For ToxiGen, we plot the percentage of toxic generations. For TruthfulQA, we plot the error rate as one minus the accuracy of identifying the correct answer. For each method, we vary the ratio of selected tasks between 5%, 10%, 15%, and 20%, and report the best result.

First, we compare GRADEX-FS with using the predictions from the pretrained LM directly, using the features from the LM in a regression problem, and fine-tuning on all tasks (MTL). We find that GRADEX-FS outperforms each of these three methods by 9%, 11%, and 8% on average, respectively.

Second, GRADEX-FS and GRADEX-RE deliver comparable performance (within a 1% performance gap) to the forward selection and random ensemble, using less than **0.5**% of the computation cost of full fine-tuning.

Lastly, we find that GRADEX-FS improves over DSIR, DEFT, and LESS by **3.8**%, all of which use a similar number of FLOPs, and GRADEX-RE outperforms these baseline methods by **4.7**%.

4.3.2 Results for chain-of-thought fine-tuning

Next, we report the results on chain-of-thought fine-tuning, illustrated in Figure 4b, tested on FLAN-T5-Base. In a nutshell, our findings remain consistent with those of Section 4.3.1.

First, GRADEX-FS outperforms directly using the pretrained model for making predictions, feature transfer from the pretrained model, and fine-tuning on all tasks by 7%, 16%, and 5% (on average), respectively. Second, even with the estimation we see similar downstream performance to forward selection and random ensemble (with full fine-tuning), while using only **3**% and **0.5**% of the total computation costs. Lastly, both GRADEX-FS and GradEx-RE can outperform the four baseline selection methods by **2.4**% and **3.5**% (on average), respectively.

As for data selection, we first cluster the samples into n groups. We vary n between 50, 100, 200, and 400 and we also evaluate our approach without the clustering step on StrategyQA. We find that using $n = 100$ groups yields the best performance, and indeed outperforms not using clustering by 1.4%. Therefore, we set $n = 100$ in all the data selection experiments.

5 Related Work

Parameter-efficient fine-tuning: One influential line of work has sought to design adapters, which are small modules injected into the intermediate layers of a deep neural network. With adapters, only a small fraction of the entire network has to change inside the adapters, and this approach has found applications in many settings such as text classification (Howard and Ruder, 2018), text transfer (Pfeiffer et al., 2021), and cross-lingual transfer such as named entity recognition and commonsense reasoning (Bapna and Firat, 2019). Another different approach is to use LoRA (Hu et al., 2021), which constrains the fine-tuning region inside a low-rank subspace, greatly improving the parameter efficiency of fine-tuning. Both LoRA and follow-up works such as QLoRA (Dettmers et al., 2023) and ReLoRA (Lialin et al., 2024) focus on fine-tuning a single task. Mahabadi et al. (2021) studies parameter-efficient multitask fine-tuning via shared hypernetworks. Our work can be viewed as expanding these methods to multitask learning (MTL). We believe this connection between MTL and fine-tuning would be worth further exploration in future work.

Multitask learning for NLP: There is also a line of work on building transfer learning approaches for tackling low-resource languages and tasks. For instance, adapting from a high-resource language to another low-resource language is a particularly effective strategy (Neubig and Hu, 2018). Vu et al. (2020) explore the transferability using a large collection of NLP tasks. This large-scale analysis underscores the intricacy of representational transfer in natural languages. Vu et al. (2022) examine feature transfer in the context of prompt tuning. There is another line of work for domain adaptation using a mixture of experts (Shazeer et al., 2017). Wu et al. (2020) and Yang et al. (2020) provide a theoretical analysis of the multi-headed architecture commonly used for conducting multitask learning. Their work highlights the issue of negative transfers in multitask learning. For further references, see recent surveys discussing the ongoing developments and challenges of multitask learning for NLP (Raffel et al., 2020; Zhang et al., 2023).

Data modeling: There is a growing line of work on understanding the role of individual samples in deep networks and large models. Wettig et al. (2024) select pre-training data by training a rater model to evaluate four data quality criteria that align with human intuitions. Thrush et al. (2024) select data based on the losses and perplexities of existing LLMs on pretraining texts. In contrast, this work focuses on task-relatedness in fine-tuning language models. This is a simpler problem compared to pre-training, as the number of tasks is relatively smaller. A potential adaptation could be splitting pre-training into multiple phases and applying our techniques to develop a curriculum for pre-training. Li et al. (2023b) introduce a surrogate modeling approach to identify subsets of source tasks that benefit a target task. They show that a linear surrogate model is particularly helpful for identifying negative transfers, including higher-order negative transfers from a subset of source tasks to the target task. This higher-order transfer corresponds to a boosting procedure that is helpful (Li et al., 2023a). Li et al. (2024a) introduce a tree-structured surrogate model for finding compositions of data augmentations. Our approach is also related to the work of Park et al. (2023) on using random sampling for data attribution. The difference is that we utilize the linearization property of LLMs for fine-tuning close to the initialization.

6 Conclusion

This paper introduced a novel method for estimating LM fine-tuning performances with a first-order approximation approach. The method can significantly accelerate conventional subset selection, thus unlocking their applications to task/data selection for fine-tuning. Evaluation across numerous datasets and LMs demonstrates the benefit of GradEx when applied to subset selection.

Acknowlegement: Thanks to the anonymous referees and the action editor for their constructive feedback. The work of D. Li, Z. Zhang, and H. Zhang is in part supported by NSF award IIS-2412008.

References

Ankur Bapna and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1538–1548. 9

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36. 3, 9

Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. 2021. Efficiently identifying task groupings for multi-task learning. *NeurIPS*. 3

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR. 2, 4

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. 2021. Sharpness-aware minimization for efficiently improving generalization. *ICLR*. 7

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *TACL*. 6

Thomas Hartvigsen, Saadia Gabriel, Hamid Palangi, Maarten Sap, Dipankar Ray, and Ece Kamar. 2022. Toxigen: A large-scale machine-generated dataset for adversarial and implicit hate speech detection. *ACL*. 8

Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. 2009. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer. 1, 3

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR. 1

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339. 9

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *ICLR*. 1, 3, 9

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. 2022. Datamodels: Predicting predictions from training data. *ICML*. 3

Hamish Ivison, Noah A Smith, Hannaneh Hajishirzi, and Pradeep Dasigi. 2023. Data-efficient finetuning using cross-task nearest neighbors. In *ACL*. 1, 5, 6, 8

William B Johnson. 1984. Extensions of lipshitz mapping into hilbert space. In *Conference modern analysis and probability, 1984*, pages 189–206. 5

Dongyue Li, Kailai Chen, Predrag Radivojac, and Hongyang R Zhang. 2024a. Learning tree-structured composition of data augmentation. *Transactions on Machine Learning Research*. 9

Dongyue Li, Haotian Ju, Aneesh Sharma, and Hongyang R Zhang. 2023a. Boosting multitask learning on graphs through higher-order task affinities. *SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. 3, 9

Dongyue Li, Huy Nguyen, and Hongyang Ryan Zhang. 2023b. Identification of negative transfers in multitask learning using surrogate models. *Transactions on Machine Learning Research*. 3, 9

Dongyue Li, Aneesh Sharma, and Hongyang R Zhang. 2024b. Scalable multitask learning using gradient-based estimation of task affinity. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1542–1553. 3

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. 2024. Relora: High-rank training through low-rank updates. In *The Twelfth International Conference on Learning Representations*. 9

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Truthfulqa: Measuring how models mimic human falsehoods. *ACL*. 8

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 565–576. 9

Sadhika Malladi, Alexander Wettig, Dingli Yu, Danqi Chen, and Sanjeev Arora. 2023. A kernel-based view of language model fine-tuning. In *ICML*. PMLR. 4

Graham Neubig and Junjie Hu. 2018. Rapid adaptation of neural machine translation to new languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 1, 9

Sung Min Park, Kristian Georgiev, Andrew Ilyas, Guillaume Leclerc, and Aleksander Mądry. 2023. Trak: attributing model behavior at scale. In *Proceedings of the 40th International Conference on Machine Learning*, pages 27074–27113. 9

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503. 9

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673. 1

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*. 9

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*. 1

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*. 9

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *NAACL-HLT*. 6

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca. 3

Tristan Thrush, Christopher Potts, and Tatsunori Hashimoto. 2024. Improving pretraining data using perplexity correlations. *arXiv preprint arXiv:2409.05816*. 9

Ahmet Üstün, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord. 2020. Udapter: Language adaptation for truly universal dependency parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2302–2315. 1

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, and Daniel Cer. 2022. Spot: Better frozen model adaptation through soft prompt transfer. *ACL*. 1, 3, 9

Tu Vu, Tong Wang, Tsendsuren Munkhdalai, Alessandro Sordoni, Adam Trischler, Andrew Mattarella-Micke, Subhransu Maji, and Mohit Iyyer. 2020. Exploring and predicting transferability across nlp tasks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7882–7926. 1, 3, 9

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICML*. 3

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023a. How far can camels go? exploring the state of instruction tuning on open resources. *NeurIPS (Dataset and Benchmark Track)*. 8

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. Self-instruct: Aligning language models with self-generated instructions. *ACL*. 3

Alexander Wettig, Aatmik Gupta, Saumya Malik, and Danqi Chen. 2024. Qurating: Selecting high-quality data for training language models. *arXiv preprint arXiv:2402.09739*. 9

Sen Wu, Hongyang R Zhang, and Christopher Ré. 2020. Understanding and improving information transfer in multi-task learning. In *International Conference on Learning Representations*. 3, 9

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. 2024. Less: Selecting influential data for targeted instruction tuning. *ICML*. 1, 2, 5, 6, 8, 13

Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy S Liang. 2023. Data selection for language models via importance resampling. *Advances in Neural Information Processing Systems*, 36:34201–34227. 1, 2, 5, 6, 8, 13

Fan Yang, Hongyang R Zhang, Sen Wu, Christopher Ré, and Weijie J Su. 2020. Precise high-dimensional asymptotics for quantifying heterogeneous transfers. *arXiv preprint arXiv:2010.11750*. 9

Hongyang R. Zhang, Dongyue Li, and Haotian Ju. 2024. Noise stability optimization for finding flat minima: A hessian-based regularization approach. *Transactions on Machine Learning Research*. 4, 7

Zhihan Zhang, Wenhao Yu, Mengxia Yu, Zhichun Guo, and Meng Jiang. 2023. A survey of multi-task learning in natural language processing: Regarding task relatedness and training methods. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 943–956. 1, 3, 9

Barret Zoph, Deniz Yuret, Jonathan May, and Kevin Knight. 2016. Transfer learning for low-resource neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 3

Limitations

One limitation of our algorithm is that it requires estimating the fine-tuned model performance based on all the model weights. Designing techniques to estimate fine-tuning performance in a limited set of model weights or for closed-source models such as GPT-4 is an interesting open question. Our study is also restricted to fine-tuning, while there may well be other scenarios where one would like to adopt a language model, such as in-context learning or alignment with RLHF. Further exploring functional approximation techniques for these settings would be another promising avenue for future work.

Discussion about Potential Risks

This paper examines the problem of fine-tuning language models given multiple data sources. While the use of language models may have potential societal consequences in the future, there are no specific ethical concerns arising from our work. Due to the technical nature of this paper, there are no direct implications of negative societal impacts.

A Mathematical Notations

We provide a list of symbols and their meaning used in the paper for reference below:

- S: A subset of $\{1, 2, \dots, n\}$.
- $f(S)$: The performance of an LM fine-tuned on tasks in S, evaluated on the target set.
- $\theta^\star$: The meta-initialization, i.e., vectorized LM weights fine-tuned on all tasks.
- $\hat{\theta}_S$: The vectorized LM weights fine-tuned on a subset of tasks S.
- $h_{\theta^\star}(s, y)$: Model loss given an input pair s, y.
- $\nabla h_{\theta^\star}(s, y)$: Vectorized gradients of model output with respect to model weights X.
- T_i: The average performance of $f(S)$ over multiple subsets S that include task i, for every $i = 1, 2, \dots, n$.

B Omitted Materials from Section 2

$f(S)$ is not monotone: We find that $f(S)$ is not monotone, in the sense that if we add one helper task into S, this does not necessarily improve the outcome. Concretely, we start with an initial set S. It contains a target task t corresponding to the instructions of "arrange" in Alpaca. Then, we keep adding a "helping" task (i) to S, if $f(\{i, t\}) < f(\{t\})$ (reducing the loss). According to our experiment, once more than two tasks are added, $f(S)$ increases gradually, indicating that adding more tasks can worsen the performance of the target task, and these are all "helpful" tasks (in the sense of pairwise transfer).

$f(S)$ is not submodular: We also find that $f(S)$ is not submodular. A function $f(\cdot)$ is submodular if for any two subsets $S \subseteq S' \subseteq \{1, 2, \dots, T\}$ and any single task x, $f(\{x\} \cup S') - f(S') \leq f(\{x\} \cup S) - f(S)$. To test this, we start with a set S that includes the target task and a task that negatively transfers, i.e., increases the loss of the target task. Then, we also keep adding a "helping" task (i) to S. According to our experiment, we observe that the benefit of adding tasks gradually decreases. Initially, adding the first two tasks to the original pair improves performance. However, this improvement diminishes once three or more tasks are added.

C Ommited Materials for Section 3

Meta-training improves approximation quality: In Section 3, we observe that first-order expansion near a meta-initialization fine-tuned on all tasks provides an accurate approximation. Next, we evaluate whether this approximation can be achieved without the meta-training step. We measure the first-order Taylor expansion on the pretrained GPT-Neo-1.3B and TinyLlama-1.1B model. We observe that the approximation from the pretrained initialization incurs within 7% error. In contrast, using meta-initialization is significantly better. The approximation achieves less than 1% error. We report the results in Table 4.

While using the second-order approximation can further reduce estimation error, it requires computing Hessian-gradient products, which is computationally expensive. Thus, we focus on the first-order approximation in this paper.

Dimension reduction: Recall that the dimension of $\nabla h_{\theta^\star}(s, y)$ is the same as the number of trainable parameters in a neural network. Thus, we project the gradients to a much lower dimension using random projection. Let P be a p by d Gaussian random matrix, whose entries are independently sampled from a Gaussian distribution $N(0, d^{-1})$.

Table 4: We compare the error of first-order approximation from the pretrained initialization and the meta-initialization fine-tuned on all data sources. The results are averaged over 50 random task subsets. We sample subsets of subsets of size 19 on Alpaca.

Distance\RRSS	**GPT-Neo-1.3B**		**TinyLlama-1.1B**	
Initialization $\theta^\star$	Pretrained LM	Fine-tuned on all tasks	Pretrained LM	Fine-tuned on all tasks
2%	$6_{\pm 0.6} \times 10^{-4}$	$6_{\pm 0.5} \times 10^{-5}$	$8_{\pm 1.1} \times 10^{-4}$	$3_{\pm 0.4} \times 10^{-5}$
4%	$1_{\pm 0.2} \times 10^{-3}$	$3_{\pm 0.3} \times 10^{-4}$	$2_{\pm 0.4} \times 10^{-3}$	$1_{\pm 0.3} \times 10^{-4}$
6%	$2_{\pm 0.1} \times 10^{-3}$	$6_{\pm 0.6} \times 10^{-4}$	$3_{\pm 0.4} \times 10^{-4}$	$3_{\pm 0.7} \times 10^{-4}$
8%	$3_{\pm 0.3} \times 10^{-3}$	$1_{\pm 0.2} \times 10^{-3}$	$3_{\pm 0.7} \times 10^{-4}$	$4_{\pm 0.9} \times 10^{-4}$
10%	$4_{\pm 0.2} \times 10^{-2}$	$5_{\pm 0.8} \times 10^{-3}$	$7_{\pm 0.6} \times 10^{-2}$	$5_{\pm 0.8} \times 10^{-3}$

We project the gradients from dimension p to dimension d as $\tilde{g}_i = P^\top \nabla h_{\theta^\star}(s, y)$. Then, we solve the logistic regression in dimension d. Denote the solution as $\hat{\theta}_d$. We set $\hat{\theta}_S \leftarrow P\hat{\theta}_d + \theta^\star$ to map the projected solution back to p-dimensions.

Application to multi-classification tasks: To accommodate multi-classification tasks, we can view the cross-entropy loss in the same form as a logistic loss by transforming the model output function. Specifically, for a training example (s, y) where y is a multi-classification label, we can define the model output function as $h_\theta(s, y) := \log\left(\frac{p(y|s;\theta)}{1-p(y|s;\theta)}\right)$ where $p(y|s;\theta)$ is the softmax probability assigned to the correct class. Then, the cross-entropy can be rewritten as the logistic loss as $\ell(s, y; \theta) = -\log p(y|s;\theta)$.

Application to generative tasks: To accommodate generative tasks, we can view the loss at each output position as a multi-classification loss and average the gradient over every output position. Specifically, for each training example (s, y), suppose that y is a sequence of length L denoted as $y = (y_1, y_2, \ldots, y_L)$. The loss can be written as the sum over L conditional probabilities. We can view this as L multi-class classification losses and apply the above transformation to each position $i = 1, 2, \ldots, L$. Then, we can compute the averaged gradient over the L output positions.

Runtime complexity. As outlined in Table 2, we compare the runtime complexity of our algorithm with that of the subset selection baselines. Below, we detail the exact number of forward passes required by each method. We denote the number of tasks as n. Note that the following number of forward passes is multiplied by the average number of forward passes to train on one task.

Forward selection: This algorithm performs greedy subset selection. It starts with an empty set and iterates up to n times to select the optimal task to combine with the current subset. At the i-th iteration, it fine-tunes one model on the current subset combined with each of the remaining $n - i + 1$ unselected tasks. Consequently, the total number of forward passes is at most $\sum_{i=1}^{n}(n - i + 1) \cdot i = \frac{1}{6}n(n + 1)(n + 2)$.

Random ensemble: This algorithm fine-tunes models on randomly sampled subsets of tasks and averages their performance on each task to generate a score. It requires $O(n \log n)$ subsets for these scores to converge. In practice, we observe that sampling approximately $10n$ subsets is usually sufficient. If we denote the size of each subset as α, the total number of forward passes required is $\alpha n \log n$.

DSIR (Xie et al., 2023): This algorithm assesses the n-gram features of every data sample without requiring model forward passes. It trains a model on a selected subset of tasks. The total number of forward passes is at most n.

DEFT (Xia et al., 2024): These two algorithms follow a similar procedure. First, train a model on all given tasks. Then, compute the similarity score between feature representations (or gradients) of training and test samples. Lastly, select tasks based on the scores and train a model on the selected subset of tasks. This process requires a total of $3n$ forward passes.

GRADEX: Our algorithm requires a comparable number of forward passes as DEFT. First, we train a model on all task data and project the gradients of each data sample. Then, instead of computing gradient similarities, we estimate model fine-tuning performances by solving logistic regression on the projected gradients. Finally, we select a subset of tasks based on these estimated performances and train a model on the selected subset. This also requires a total of $3n$ forward passes.

Table 5: Detailed statistics about all the datasets used in the experiments.

Dataset	# Tasks	Avg size	Category	Source
GLUE	9	106,416	3 task categories	gluebenchmark.com/
FLAN v2	1,691	59	150 task categories	huggingface.co/datasets/philschmid/flanv2
COT	18	8,302	Chain-of-thought reasoning	huggingface.co/datasets/QingyiSi/Alpaca-CoT
Alpaca	38	1,073	Text generation	huggingface.co/datasets/tatsu-lab/alpaca
ToxiGen	1	7,000	Text generation	huggingface.co/datasets/toxigen/toxigen-data
TruthfulQA	1	818	Open-domain QA	huggingface.co/datasets/truthfulqa/truthful_qa
CQA	100	97	Commonsense reasoning	huggingface.co/datasets/tau/commonsense_qa
StrategyQA	100	128	Commonsense reasoning	github.com/eladsegal/strategyqa

Table 6: List of hyper-parameters used in the experiments.

Dataset	Model	Step size	Batch size	Epochs	LoRA rank	Results
GLUE	BERT-Base, RoBERTa-Base	$5e^{-5}$	16	5	Full model	Table 1
Alpaca	Pythia-70M, GPT-2	$5e^{-5}$	16	10	Full model	Table 1 and 3
	FLAN-T5-Base, BloomZ-560M TinyLlama-1.1B, GPT-Neo-1.3B, OPT-1.3B, Gemma-2-2B Mistral-7B, Llama-3-8B	$5e^{-5}$	16	10	16	
StrategyQA	GPT-2	$3e^{-4}$	8	20	Full model	Table 3
	FLAN-T5-Base, TinyLlama-1.1B, GPT-Neo-1.3B, Llama-3-8B	$3e^{-4}$	8	20	16	
ToxiGen	TinyLlama-1.1B	$2e^{-5}$	128	10	128	Figure 4a
TruthfulQA	TinyLlama-1.1B	$2e^{-5}$	128	10	128	
StrategyQA	FLAN-T5-Base	$3e^{-4}$	8	20	16	Figure 4b
CQA	FLAN-T5-Base	$3e^{-4}$	8	20	16	

D Additional Experiments

D.1 Setup

We describe each dataset in Table 5. Among them, FLAN v2 includes a variety of tasks, as it combines four prior instruction tuning datasets. We refer the reader to the paper for their task categories. We used a sampled version of FLAN v2. We partition Alpaca by their instruction types into tasks.

We experiment with the following models: BERT-Base, RoBERTa-Base, FLAN-T5-Base, Pythia-70M, GPT-2, BloomZ-560M, TinyLlama-1.1B, OPT-1.3B, GPT-Neo-1.3B, Gemma-2-2B, Mistral-7B, and Llama-3-8B.

In our experiments, we fine-tune the models using the AdamW optimizer. We fine-tune the entire model for smaller models, including BERT-Base, RoBERTa-Base, Pythia-70M, and GPT-2. For other models, we use LoRA for fine-tuning. The training hyper-parameters used for each experiment are described in Table 6.

For ToxiGen, we measure the percentage of toxic generations of the model. First, the model is prompted to produce toxic languages using human-designed hateful prompts; then, the generations are classified by a toxic content detection model. For TruthfulQA, we evaluate the model's accuracy in detecting truthful statements. Given a question and five answer choices, we measure the accuracy of the model in assigning the highest probability to the correct answer.

D.2 Omitted results

In Table 7, we report the complete results corresponding to Figure 4.

D.3 Comparison of GPU hours

We also compare the GPU hours of each method on StrategyQA. We observe that GRADEX-FS uses 10.6 GPU hours and GRADEX-RE uses 14.7 GPU hours. Corroborating with the FLOPs comparison results, our algorithm reduces the GPU hours of classic selection methods by 10×. Moreover, our algorithm uses comparable GPU hours to existing selection baselines, with LESS and DEFT taking 10.2 GPU hours. The comparison has a similar trend in instruction fine-tuning datasets.

Table 7: Performance scores and computation cost (FLOPs) on four datasets. ToxiGen and TruthfulQA are used to evaluate instruction tuning. CommonsenseQA and StartegyQA are used to evaluate chain-of-thought fine-tuning. We report the averaged results and standard deviations over three random seeds for each method.

Dataset	ToxiGen	TruthfulQA	CommonsenseQA	StrategyQA
# all training samples	272,770	272,770	9,741	12,824
# test samples	7,000	818	1,221	687
Model	TinyLlama-1.1B	TinyLlama-1.1B	FLAN-T5-Base	FLAN-T5-Base
Metrics (%)	Toxic generations (↓)	Accuracy (↑)	Accuracy (↑)	Accuracy (↑)
Pretrained model	70.14 ± 0.00	22.03 ± 0.00	72.23 ± 0.00	51.23 ± 0.00
Feature transfer	73.20 ± 0.44	21.03 ± 0.28	51.33 ± 0.44	44.68 ± 0.45
MTL	69.44 ± 0.43	23.86 ± 0.78	59.05 ± 0.69	60.84 ± 0.80
DSIR	68.10 ± 0.32	24.02 ± 0.43	64.56 ± 0.28	61.49 ± 0.19
DEFT	65.71 ± 0.32	24.38 ± 0.61	64.78 ± 0.93	61.79 ± 0.32
LESS	63.90 ± 0.27	24.53 ± 0.48	64.92 ± 0.32	61.93 ± 0.21
Forward selection	58.50 ± 0.79	27.72 ± 0.21	67.27 ± 0.68	63.54 ± 0.13
Random ensemble	57.88 ± 0.28	28.21 ± 0.60	67.78 ± 0.68	64.28 ± 0.24
GradEx-FS	59.36 ± 0.53	27.25 ± 0.42	67.09 ± 0.53	63.30 ± 0.50
GradEx-RE	58.29 ± 0.66	27.93 ± 0.42	67.50 ± 0.94	63.90 ± 0.43
# FLOPs (↓)				
MTL	5.80×10^{17}	5.80×10^{17}	1.19×10^{16}	1.57×10^{16}
DSIR	8.98×10^{17}	8.98×10^{17}	1.78×10^{16}	2.43×10^{16}
DEFT	1.04×10^{18}	1.04×10^{18}	1.84×10^{16}	2.50×10^{16}
LESS	1.04×10^{18}	1.04×10^{18}	1.84×10^{16}	2.50×10^{16}
Forward selection	5.54×10^{20}	5.54×10^{20}	5.98×10^{17}	8.62×10^{17}
Random ensemble	5.04×10^{21}	5.04×10^{21}	5.98×10^{18}	7.84×10^{18}
GradEx-FS	2.81×10^{18}	2.61×10^{18}	2.03×10^{16}	2.62×10^{16}
GradEx-RE	1.87×10^{19}	1.53×10^{19}	3.71×10^{16}	3.47×10^{16}

D.4 Ablation studies

Projection dimension: Recall that in our algorithm, we project gradients to a lower dimension before solving the logistic regression in the estimation stage. We note that a small value of the projection dimension is sufficient to achieve the approximation results. We vary the projection dimension d between 50, 100, 200, and 400 for running GradEx on FLAN-T5-Base. The results are shown in Table 8. Once d increases above 100, the error stabilizes around 0.03%, so we set d to 100 by default. d is approximately $7\log(p)$ where $p = 2654208$ is the number of trainable parameters in FLAN-T5-Base.

Table 8: Distance vs. speedup for varied d, computed using Alpaca and StrategyQA.

Alpaca	GradEx-FS		GradEx-RE	
d	**Distance**	**Speedup**	**Distance**	**Speedup**
50	4.8×10^{-4}	17.6×	5.2×10^{-4}	43.3×
100	3.2×10^{-4}	17.6×	3.4×10^{-4}	43.3×
200	2.9×10^{-4}	17.5×	3.0×10^{-4}	43.2×
400	2.8×10^{-4}	17.5×	2.9×10^{-4}	43.2×
StrategyQA	GradEx-FS		GradEx-RE	
d	**Distance**	**Speedup**	**Distance**	**Speedup**
50	4.6×10^{-4}	30.5×	5.6×10^{-4}	44.9×
100	3.0×10^{-4}	30.5×	3.8×10^{-4}	44.9×
200	2.8×10^{-4}	30.4×	3.6×10^{-4}	44.8×
400	2.7×10^{-4}	30.4×	3.4×10^{-4}	44.8×

Number of subsets m: Recall that in applying GradEx-RE, the score in the random ensemble for each task is estimated by averaging the value functions of m subsets that contain the task. For the strategy QA data with $n = 100$ datasets, we observe that the estimated higher-order task affinity converges using $m = 1000$ subsets. We computed the distance between the estimated scores T using m subsets and the final scores T^*, observing that the distance sharply decreases as m increases, eventually converging to zero. This analysis was conducted for different subset sizes, with $\alpha = 0.25n$, $\alpha = 0.5n$, and $\alpha = 0.75n$, respectively. The distance stabilizes as m approaches 1000, indicating that the estimated task affinity scores converge.

One might hypothesize that using 1000 or more subsets achieves comparable performance. We conduct the selection algorithm using $m = 500, 1000, 1500, 2000$. We notice no further gain in the downstream accuracy when using more subsets than 1000. Thus, we set $m = 1000$.

Subset size: We vary subset size α between $0.25n$, $0.5n$, and $0.75n$. We find the T_is all converge at a similar rate. In addition, using a subset size of $0.75n$ achieves better performance in downstream selection. The reason is that this size is closer to the size of the selected subset (i.e., $0.7n$).

D.5 Complete pseudocode for subset selection procedures

Algorithm 2 GradEx-FS

```
Input: Training datasets of $n$ data sources; a multitask fine-tuning algorithm $f$
Output: A subset of selected tasks $S^\star$
Initiate an empty set $S_{\text{cur}} = \{\}$
for Step $i = 1, 2, \ldots, n$ do
    for Each dataset not selected in current subset $j \in [n]/S_{\text{cur}}$ do
        Add dataset $j$ to current subset: $S_j \leftarrow S_{\text{cur}} \bigcup \{j\}$
        Evaluate $\hat{f}(S_j)$ using GradEx
    end for
    Choose the dataset with the highest value function $j^* \leftarrow \arg\max_{j \in [n]/S_{\text{cur}}} \hat{f}(S < j)$
    Add the dataset to the current subset: $S_{\text{cur}} \leftarrow S_{\text{cur}} + j^*$, if $\hat{f}(S_j) > \hat{f}(S_{\text{cur}})$. Otherwise, stop searching
end for
Return the selected subset $S^\star \leftarrow S_{\text{cur}}$
```

Algorithm 3 GradEx-RE

```
for $k = 1, \ldots, m$ do
    Sample a random subset $S_k$ from $\{1, 2, \ldots, n\}$ with size $\alpha$
    Evaluate $\hat{f}(S_k)$ using GradEx
end for
Estimate the random ensemble score for each dataset $i$ as the value function averaged over subsets that include $i$:
    $$T_i = \frac{1}{n_i} \sum_{1 \le k \le m:\ i \in S_k} \hat{f}\big(S_k\big), \text{ for } 1 \le i \le n$$
Select a subset of datasets by thresholding the scores: $S^\star = \{i \mid \hat{T}_i > \gamma,\ \forall\, i = 1, 2, \ldots, n\}$
```