SMART: Submodular Data Mixture Strategy for Instruction Tuning

H S V N S Kowndinya Renduchintala
Media and Data Science Research
Adobe Inc., India
rharisrikowndinya333@gmail.com

Sumit Bhatia
Media and Data Science Research
Adobe Inc., India
sumit.bhatia@adobe.com

Ganesh Ramakrishnan
Dept. of Computer Science & Engineering
Indian Institute of Technology Bombay
ganesh@cse.iitb.ac.in

Abstract

Instruction Tuning involves finetuning a language model on a collection of instruction-formatted datasets in order to enhance the generalizability of the model to unseen tasks. Studies have shown the importance of balancing different task proportions during finetuning, but finding the right balance remains challenging. Unfortunately, there's currently no systematic method beyond manual tuning or relying on practitioners' intuition. In this paper, we introduce SMART (**S**ubmodular data **M**ixture str**A**tegy for inst**R**uction **T**uning) — a novel data mixture strategy which makes use of a submodular function to assign importance scores to tasks which are then used to determine the mixture weights. Given a fine-tuning budget, SMART redistributes the budget among tasks and selects non-redundant samples from each task. Experimental results demonstrate that SMART significantly outperforms traditional methods such as examples proportional mixing and equal mixing. Furthermore, SMART facilitates the creation of data mixtures based on a few representative subsets of tasks alone and through task pruning analysis, we reveal that in a limited budget setting, allocating budget among a subset of representative tasks yields superior performance compared to distributing the budget among all tasks. The code for reproducing our results is open-sourced at https://github.com/kowndinya-renduchintala/SMART.

1 Introduction

"Your ability to juggle many tasks will take you far."

One of the main goals of artificial intelligence (AI) research is to build machines that can *communicate* (Turing, 1950), and an essential part of communication is to understand and follow *instructions*. Large Language Models (LLMs), which are pre-trained over massive text corpora on *next-token-prediction* objective, can perform a wide range of NLP tasks via *"prompting"* (Brown et al., 2020; Kojima et al., 2022; Almazrouei et al., 2023; Liu et al., 2023; Touvron et al., 2023).

Instruction Tuning (Wei et al., 2021; Sanh et al., 2021; Chung et al., 2022) is an approach that further enhances the instruction-following ability and generalizability of pre-trained LLMs to unseen tasks. It involves fine-tuning an LLM on a collection of instruction-formatted instances (encompassing multiple tasks) - each consisting of an instruction (or task description), an optional input, the corresponding output (the ground truth) and optionally a few demonstrations/examples. It is a special case of multitask learning where the LLM is finetuned on a collection of instruction-formatted multitask datasets (Chung et al., 2022). Finetuning on multiple tasks simultaneously, allows the model to share and transfer information across tasks, resulting in a better common internal representation that is preferred by all tasks while suppressing task-dependent noise (Caruana, 1997). Consequently, the model learns to generalize to unseen tasks by discerning helpful cues from both implicitly and explicitly related tasks that it has previously seen.

The performance enhancement from instruction tuning is heavily contingent on data quality, data quantity, and task composition (Wang et al., 2023b). Studies by Iyer et al. (2022) and Longpre et al. (2023) have shown that while scaling the number of tasks is important, the relative proportion of various tasks (mixture weighting) merits as much attention for optimal instruction tuning. Intuitively, we want the model to see enough data for a given task that it can perform well on it, but not to see so much data that it memorizes the training set (Raffel et al., 2020). Iyer et al. (2022) performed *manual tuning* of various benchmark proportions and decided on a final mixture, whereas Longpre et al. (2023) studied the impact of removing each benchmark from the finetuning mixture and relied on their *practioners' intuition* from there on, to decide

Findings of the Association for Computational Linguistics ACL 2024, pages 12916–12934

on the exact proportions of benchmarks. In this work, we would like to explore a more systematic approach to mixture weighting. Specifically, we are motivated by the fact that in a large multitask dataset like FLAN 2022 (Longpre et al., 2023), which has 1840 tasks, there will likely be many similar tasks leading to redundancies and not all of them may require sampling in equal proportions. For instance, there might be many tasks of the type Natural Language Inference (NLI), and it might be enough to sample relatively more instances from a few *representative* NLI tasks and less from the others. Furthermore, *which* samples we select from each task is also crucial because the samples should faithfully represent the task at hand. A random subset may fail to do this as it can miss out on essential corner cases.

With this context, we focus on the following two fundamental research questions (**RQ**s) that form the basis for our subsequent inquiry:

- **(RQ1)** Given a *huge* multitask instruction-tuning dataset and a limited fine-tuning budget which is defined by the total number of $(prompt, response)$ instances that can be used for fine-tuning, how do we divide this budget among thousands of tasks present in the dataset? i.e., *how many* instances to sample from each task? and *which* instances to sample from each task?
- **(RQ2)** Can we go a step further and strategically prune some tasks altogether and only fine-tune on a small subset of representative tasks without hurting the performance? If yes, what is the nature of this subset?

To the best of our knowledge, there's currently no principled approach to determining task compositions for instruction tuning, other than *manual tuning* and/or *practioners' intuition*.

As a first step towards addressing both of the above **RQ**s, we first define a common subset selection problem (more formally stated in Section 3) as follows - Given a *huge* collection of M instruction-formatted task datasets, a task budget $M' \leq M$ and a total budget (N') of $(prompt, response)$ pairs, which M' tasks to select? and how many instances to select from each of these M' tasks and which instances to select? Note that **RQ1** is an instance of this problem where $M' = M$.

Constrained Submodular Maximization (Section 2) proves to be a good model for discovering representative subsets (or *coresets*) of a massive training dataset (or *ground set)* that acts as surrogate (i.e., achieves similar performance) and are much better than uniformly-at-random subsets. Intuitively, this is because submodular functions model *information* in subsets, and hence maximizing a submodular function subject to a constraint yields non-redundant subsets of the ground set. An essential feature of this model is that it returns weighted subsets, i.e., each sample in the coreset comes with an associated score, which indicates how important the sample is.

Inspired by submodular functions, we propose our solution (Section 3) to the above subset selection problem for instruction tuning that works in two stages. In the first stage, we select a weighted subset of tasks from the full dataset where the weights will determine how many samples to select from each task. In the next stage, we select samples from each task based on the assigned task budgets. Note that the submodular functions used in each stage are not necessarily identical (Section 4.8).

The *main contributions* of our work can be summarized as follows:

- We introduce SMART — a novel data mixture strategy for instruction tuning that models the data mixture problem (Section 3) as a sequence of two cardinality-constrained submodular maximization problems and offer empirical evidence that it outperforms both examples proportional and equal mixing baselines (Section 4) as well as the mixture weights proposed by Longpre et al. (2023).
- Existing works like Longpre et al. (2023) have reported a continuous increase in performance upon increasing the number of tasks (though the gains themselves may be diminishing). However, we posit that this depends on the order in which new tasks are incorporated and show empirically that in the case of SMART mixtures, a performance peak is observed with an initial addition of few representative tasks and upon adding more and more tasks, the performance is not sustained (Section 4.6).
- We find that the nature of instances that should be selected in each task (i.e, whether a representative or diverse subset) also depends on the total task budget, M' (Section 4.8). For higher M's, each task on average gets the relatively low budget and selecting representative samples is more important; however for lower M's, when there is sufficient enough budget for each task, the need for diversity dominates that of representation.

2 Background: Submodularity

Notations. Let $f : 2^{\mathcal{V}} \rightarrow \mathbb{R}$ be a *set function* that assigns a value to every subset of the *ground set* $\mathcal{V}$. We use the notation $f(v|X)$ as a shorthand for $f(X \cup \{v\}) - f(X)$ i.e., the incremental value *gain* of v in the context of X.

Definition 1. *(Submodular Function) A given set function $f : 2^{\mathcal{V}} \rightarrow \mathbb{R}$ is submodular if for all $X, Y \subseteq \mathcal{V}$, where $X \subseteq Y$ and for all $v \notin Y$, the following inequality holds true:*

$$f(v|X) \geq f(v|Y) \quad \text{(diminishing gains property)}$$

Intuitively, the definition states that — Adding an element to a smaller set X yields more value gain than adding the same element to a superset Y of X. Table 1 contains examples of three submodular functions — Facility Location (models representation), Log Determinant (models diversity), and Graph Cut (models a trade-off between representation and diversity controlled by the parameter λ).

Submodular Function	$f(X)$
Facility Location	$\sum_{i \in \mathcal{V}} \max_{j \in X} s_{ij}$
Graph Cut	$\sum_{i \in \mathcal{V}, j \in X} s_{ij} - \lambda \sum_{i,j \in X} s_{ij}$
Log Determinant	$\log\det(\mathcal{S}_X)$

Table 1: Examples of Submodular Functions. $\mathcal{V}$ is the ground set and $X \subseteq \mathcal{V}$. s_{ij} is the similarity between two elements i and j of the ground set. $\mathcal{S}_X$ is the similarity matrix between the items in X. Facility Location models representation; Log Determinant models diversity and Graph Cut models a trade-off between representation and diversity (governed by the parameter λ).

Definition 2. *(Cardinality Constrained Submodular Maximization) Given a submodular function $f : 2^{\mathcal{V}} \rightarrow \mathbb{R}$ defined over the subsets of the ground set $\mathcal{V}$, the constrained submodular maximization problem involves finding $\mathcal{S}^*$ such that*

$$\mathcal{S}^* = \underset{\substack{X \subseteq \mathcal{V} \\ |X| \leq N'}}{\arg\max} f(X)$$

The above cardinality-constrained submodular maximization problem is NP-complete (Feige, 1998). However, if f is monotone submodular (i.e., $f(X) \leq f(Y)$ whenever $X \subseteq Y$), Nemhauser et al., 1978; Fisher et al., 1978 show that a simple greedy algorithm described in Algorithm 1 can be used to find an approximate solution $\mathcal{S}^{greedy}$, with a guarantee that $f(\mathcal{S}^{greedy}) \geq (1 - 1/e) f(\mathcal{S}^*)$.[1]

Algorithm 1 The Naïve Greedy

Input: Ground Set ($\mathcal{V}$), Budget (N')
$X_0 \leftarrow \emptyset$;
$\mathcal{S} \leftarrow [\,]$;
$Gains \leftarrow [\,]$;
for $i = 0$ **to** $(N' - 1)$ **do**
  $e^* = \underset{v \in \mathcal{V} \setminus X_i}{\arg\max} f(v|X_i)$;
  $g_{i+1} = f(e^*|X_i)$;
  $X_{i+1} = X_i \cup \{e^*\}$;
  $\mathcal{S}$.append(e^*);
  $Gains$.append(g_{i+1});
end for
return $\mathcal{S}, Gains$;

Remark. *The algorithm produces a weighted subset where the value gains themselves are the weights (i.e., $\max_{v \in \mathcal{V} \setminus X_i} f(v|X_i)$ is the weight of $\arg\max_{v \in \mathcal{V} \setminus X_i} f(v|X_i)$).*

Algorithm 1 (also known as Naïve Greedy) requires $\mathcal{O}(N'.|\mathcal{V}|)$ function evaluations which is costly in practice. Accelerated Greedy (Minoux, 2005), also known as Lazy Greedy, can be used instead, which leverages submodularity and offers a more efficient heap-based implementation of the same algorithm.

3 Approach

Inspired by submodularity (Section 2), in this section, we introduce a novel data mixture strategy, SMART, as a technique to solve the following subset selection problem introduced in Section 1:

Consider a collection $\mathcal{D} = \{\mathcal{T}_1, \ldots, \mathcal{T}_M\}$ of M instruction-formatted task datasets where each $\mathcal{T}_i = \{(prompt_{ij}, response_{ij})\}_{j=1}^{N_{\mathcal{T}_i}}$ consists of $N_{\mathcal{T}_i}$ $(prompt, response)$ pairs. Let $\sum_{i=1}^{M} N_{\mathcal{T}_i} = N$. Given an $M' \leq M$ and an $N' \leq N$, how do we select a subset of tasks $\mathcal{D}' = \{\mathcal{T}'_1, \ldots, \mathcal{T}'_{M'}\}$(where $\mathcal{D}' \subseteq \mathcal{D}$), and subsequently $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_{M'}\}$ (where $\mathcal{S}_j \subseteq \mathcal{T}'_j$) and $\sum_{j=1}^{M'} |\mathcal{S}_j| = N'$) such that efficiently fine-tuning on the subset $\mathcal{S}$ alone is (nearly) as effective as fine-tuning on the entire collection $\mathcal{D}$?

SMART models the above problem as a sequence of two cardinality-constrained submodular

[1] Assuming $P \neq NP$, this is the best approximation ratio that can be achieved by any polynomial time algorithm.

maximization problems. The first one is to select a weighted subset of M' tasks and the second one is to select instances - a total of N' instances from these tasks. The weights obtained in the first stage will be used to determine how many instances we sample from each task.

3.1 The Algorithm

We now give a detailed description of the two stages of SMART (summarized in Algorithm 2):

3.1.1 Stage-1: Weighted Task Subset Selection

In this first stage, given the instruction-tuning dataset $\mathcal{D} = \{\mathcal{T}_1, \ldots, \mathcal{T}_M\}$, our goal is to find $\mathcal{D}' = \{\mathcal{T}'_1, \ldots, \mathcal{T}'_{M'}\}$ where $\mathcal{D}' \subseteq \mathcal{D}$, along with the instance budgets, $\{N'_1, \ldots, N'_{M'}\}$, such that $\sum_{j=1}^{M'} |N'_j| = N'$.

If f_1 is the submodular function that we use in this stage, $\mathcal{D}'$ is given by:

$$\mathcal{D}' = \underset{\substack{X \subseteq \mathcal{D} \\ |X| \leq M'}}{\arg\max} f_1(X)$$

To find the instance budgets (N'_js), we use the second-order Taylor-softmax operation (De Brebisson and Vincent, 2015) on the value gains obtained from the greedy algorithm, to compute a probability distribution which determines the probability with which instances will be sampled from a given task i.e., if $\{g_1, \ldots g_{M'}\}$ are the value gains returned by the greedy algorithm, corresponding to the tasks $\{\mathcal{T}'_1, \ldots, \mathcal{T}'_{M'}\}$, the instance budgets are given by

$$N'_j = \frac{(1 + g_j + 0.5g_j^2)}{\sum_{k=1}^{M'}(1 + g_k + 0.5g_k^2)} \times N'$$

3.1.2 Stage-2: Instance Subset Selection

In this stage, given the subset of tasks, $\{\mathcal{T}'_1, \ldots, \mathcal{T}'_{M'}\}$, and the instance budgets $\{N'_1, \ldots N'_{M'}\}$ from the first stage, the goal is to actually select those many samples from each task. If f_2 is the submodular function used, the final subset $\mathcal{S}$ is given by

$$\mathcal{S} = \bigcup_{j=1}^{M'} \underset{\substack{X_j \subseteq \mathcal{T}'_j \\ |X_j| \leq N'_j}}{\arg\max} f_2(X_j)$$

Algorithm 2 The SMART Data Mixture Strategy

Input: Task datasets $\mathcal{D} = \{\mathcal{T}_1, \ldots, \mathcal{T}_M\}$, Task Budget ($M'$), Instance Budget ($N'$), f_1, f_2
Each $\mathcal{T}_i = \{(prompt_{ij}, response_{ij})\}_{j=1}^{N_{\mathcal{T}_i}}$
$encoder \leftarrow$ SentenceEncoder()
$prompt\_embeddings \leftarrow$ dict()
$task\_embeddings \leftarrow$ dict()
for $i = 1$ **to** M **do**
  $\mathbf{V} \leftarrow [\,]$
  for $j = 1$ **to** $N_{\mathcal{T}_i}$ **do**
    $\mathbf{v} \leftarrow encoder$.encode($prompt_{ij}$)
    $\mathbf{V}$.append($\mathbf{v}$)
  end for
  $prompt\_embeddings[\mathcal{T}_i] = \mathbf{V}$
  $task\_embeddings[\mathcal{T}_i] = \frac{1}{N_{\mathcal{T}_i}} \sum_{j=0}^{N_{\mathcal{T}_i}-1} \mathbf{V}[j]$
end for
$\mathcal{K}_{task} =$ cos_sim($task\_embeddings$)
$\mathcal{D}', gains =$ Greedy($f_1, \mathcal{D}, \mathcal{K}_{task}, M'$)
$\mathcal{D}' = \{\mathcal{T}'_1, \ldots \mathcal{T}'_{M'}\}$
$probs \leftarrow$ Taylor_Softmax($gains$)
$\mathcal{S}' \leftarrow [\,]$
for $j = 1$ **to** M' **do**
  $N'_j = probs[j] \times N'$
  $\mathcal{K}_{\mathcal{T}'_j} =$ cos_sim($prompt\_embeddings[\mathcal{T}'_j]$)
  $\mathcal{S}_{\mathcal{T}'_j}, gains =$ Greedy($f_2, \mathcal{T}'_j, \mathcal{K}_{\mathcal{T}'_j}, N'_j$)
  $\mathcal{S}'$.append($\mathcal{S}_{\mathcal{T}'_j}$)
end for
$\mathcal{S} = \bigcup_{j=1}^{M'} \mathcal{S}'[j]$
return $\mathcal{S}$

3.2 Obtaining Similarity Measures

The three submodular functions listed in Table 1 require computation of similarity measures (s_{ij}) between items in the ground set. So, all prompts in the dataset are first encoded using a sentence encoder. For instance subset selection (i.e., **Stage-2**), cosine similarity between prompt embeddings is used as the similarity measure and for weighted task subset selection (i.e., **Stage-1**), cosine similarity between task embeddings is computed, where the task embeddings are computed as the average prompt embeddings, following Vu et al. (2020). Although there are other sophisticated methods (Achille et al., 2019; Zhou et al., 2022; Xi et al., 2023; Vu et al., 2021) for obtaining task embeddings, most of them depend on the LLM at hand.

3.3 Choosing f_1 and f_2

Each submodular function in Table 1 captures a different property: facility-location emphasizes representation, graph-cut balances representation and diversity, and log-determinant prioritizes diversity. We treat f_1 and f_2 as hyperparameters in our grid search in Section 4.8, exploring three options for each function. Determining the optimal f_1 and f_2 is part of our research question (**RQ2**). We discuss the findings of grid search and their qualitative implications for instruction tuning in Section 4.10.

4 Experiments

4.1 Finetuning Data

In all the experiments, the underlying ground set ($\mathcal{D}$) is FLAN 2022 (Longpre et al., 2023; Chung et al., 2022). The collection consists of the following five sub mixtures adding up to a total of 1840 tasks and 17,591,640 (∼17.5M) instruction-formatted $(prompt, response)$ pairs:

- FLAN 2021 (Wei et al., 2021)
- T0 (Sanh et al., 2021)
- NIV2 (Wang et al., 2022)
- CoT (several chain-of-thought datasets)
- Dialog (a few dialog datasets)

Each of the tasks comes in a variety of templates - zeroshot with and without answer options, fewshot with and without answer options.

SMART Data Mixture Creation We encode prompts in the collection with GTE-large (Li et al., 2023b), a light-weight (340M parameters) BERT-based effective (Muennighoff et al., 2022) sentence encoder for semantic textual similarity. Task embeddings are obtained by averaging the corresponding prompt embeddings (Section 3.2). We use SUBMODLIB (Kaushal et al., 2022), which has the necessary algorithms implemented, to obtain the weighted task subsets (Section 3.1.1) and the instances (Section 3.1.2). We distribute the instance budgets equally among task templates based on findings by Longpre et al. (2023) which showed that an equal number of zero-shot and few-shot templates yield the best performance on held-out tasks.

4.2 Finetuning Procedure

We evaluate SMART on three 7B parameter LLMs: Llama-2 (Touvron et al., 2023), Falcon (Almazrouei et al., 2023), and Mistral (Jiang et al., 2023). We fine-tune each model for 1 epoch on a given data mixture with a learning rate of $2e-5$ for Llama-2 and Falcon, and $5e-6$ for Mistral. A batch size of 64, weight decay of 0.1, and cosine learning rate decay with a linear warmup for the initial 1% training steps are employed. The experiments all ran on 8 NVIDIA A100-SXM4-80GB GPUs, utilizing Flash Attention (Dao, 2023) for memory efficiency and speeding up finetuning. Our code is open-sourced here.

4.3 Baselines

We mainly compare SMART with two baseline mixture strategies: Examples Proportional Mixture, Equal Mixture (Raffel et al., 2020).

Examples Proportional Mixture (Baseline-1) Instances are sampled in proportion to the size of each task's dataset. This is equivalent to randomly sampling from the combined datasets.

Equal Mixture (Baseline-2) Instances are sampled from each task with equal probability *i.e.*, by dividing the total budget equally among tasks, and then uniformly sampling from each task.

We also compare SMART with mixture weights used by Longpre et al. (2023) but since their mixture weights apply to the five sub mixtures listed in Section 4.1 rather than individual tasks, we analyse this separately in Section 4.7.

4.4 Evaluation Protocol

We evaluate the fine-tuned models on two benchmark datasets: MMLU (Hendrycks et al., 2020) with 57 tasks assessing world knowledge and problem-solving, and BBH (Suzgun et al., 2022) with 23 challenging tasks from Big-Bench (Srivastava et al., 2022). MMLU covers STEM, Humanities, Social Sciences, and Other (business, medical, and misc.) categories, while BBH includes both NLP and Algorithmic tasks[2]. Evaluation involves prompting the LLM directly and using Exact Match as the scoring metric. Responses are generated using the greedy decoding approach and they undergo basic post-processing steps (removing punctuation and lower-casing) before calculating exact match. For baseline data mixtures, 3 mixtures with different random seeds are created and the mean exact match of the 3 fine-tuning runs is reported.

[2]The *Algorithmic* subcategory is so named because these tasks (e.g., 2-digit arithmetic) do not require an LLM to be solved.

N'	Data Mix.	MMLU-ZeroShot (Exact Match)					BBH-Zeroshot (Exact Match)			MMLU + BBH
		STEM	Humanities	Social Sciences	Other	MMLU FULL	NLP	Algorithmic	BBH FULL	(Weighted Avg.)
25000	EPM (Baseline-1)	30.82	46	46.71	43.05	40.63	**40.59**	**25.55**	**31.67**	38.05
	EM (Baseline-2)	30.33	45.01	43.81	40.55	39.03	40.08	20.29	29.46	36.27
	SMART (Ours)	**32.22**	**50.41**	**50.14**	**46.85**	**43.73**	38.85	24.16	30.05	**39.8**
50000	EPM (Baseline-1)	31.59	47.68	47.18	44.68	41.76	41.25	**26.49**	**32.64**	39.14
	EM (Baseline-2)	35.22	49.58	51.01	48.13	44.99	41.96	22.83	31.24	41.04
	SMART (Ours)	**36.51**	**53.06**	**54.49**	**50.79**	**47.58**	**46.73**	20.1	31.75	**43.03**
100000	EPM (Baseline-1)	32.66	50.6	51.37	47.18	44.25	43.38	**26.36**	33.48	41.16
	EM (Baseline-2)	36.03	50.7	52.53	47.1	45.57	44.4	25.18	33.55	42.11
	SMART (Ours)	**37.36**	**55.38**	**55.47**	**52.95**	**49.11**	**47.26**	24.22	**34.66**	**44.96**
200000	EPM (Baseline-1)	35.19	54.64	54.75	50.58	47.53	45.25	**26.57**	34.52	43.79
	EM (Baseline-2)	38.6	54.05	54.72	51.47	48.68	41.36	24.75	32.28	43.96
	SMART (Ours)	**39.2**	**57.29**	**58.71**	**55.01**	**51.32**	**47.99**	24.47	**35.27**	**46.7**
400000	EPM (Baseline-1)	38.16	56.53	56.99	52.56	49.85	48.72	26.04	36.49	46.01
	EM (Baseline-2)	39.43	55.97	57.59	53.65	50.52	47.37	26.08	35.8	46.29
	SMART (Ours)	**39.77**	**57.39**	**60.17**	**54.79**	**51.77**	**49.25**	**26.35**	**37.43**	**47.65**
17,591,640	Full FLAN 2022	42.44	59.1	61.82	55.1	53.43	50.7	27.6	38.11	49.03

Table 2: Comparison of SMART with baselines on MMLU-zeroshot and BBH-zeroshot for Llama-2-7b. EPM (Baseline-1) denotes Examples Proportional Mixture and EM (Baseline-2) denotes Equal Mixture. All the scores are Exact Matches. Weighted average on 57 MMLU tasks and 23 BBH tasks is reported in the last column. For baselines, exact matches are obtained by averaging across 3 fine-tuning runs.

4.5 Addressing RQ1 ($M' = M$)

RQ1 is an instance of the subset selection problem defined in Section 3, where $M' = M$, allowing all tasks for finetuning; however we still have a constraint (N') on total number of $(prompt, response)$ pairs. We use Graph-Cut and Facility Location functions (Table 1) in Stage-1 (Section 3.1.1) and Stage-2 (Section 3.1.2) of SMART respectively. This choice of f_1 and f_2 is determined via grid search discussed in Section 4.8. Table 2 contains comparison of SMART mixtures with baseline mixtures, on MMLU and BBH benchmarks, upon instruction fine-tuning Llama-2-7b on data mixtures generated by varying N' in $\{25000, 50000, 100000, 200000, 400000\}$. SMART data mixtures consistently perform better than both examples proportional mixtures and equal mixtures baseline.

4.6 Addressing RQ2 ($M' < M$)

RQ2 is an instance of the subset selection problem defined in Section 3, where $M' < M$ i.e., studying the effect of pruning some tasks altogether, on both SMART and the baseline data mixtures. We finetune Llama-2-7b on both the baseline and SMART data mixtures by varying the number of tasks (M') in $\{8, 16, 32, 64, 128, 256, 512, 1024, 1840\}$ and the total number of instances (N') in $\{25000, 50000, 100000, 200000, 400000\}$. Figure 1 depicts the task-scaling plots for each N'. Baseline data mixtures' performance steadily improves upon increasing the number of tasks, even though the gains are diminishing after a point. In contrast, SMART data mixtures yield optimal performance at an in-between point and this performance does not seem to sustain upon adding more and more tasks, suggesting that rather than increasing the tasks, focusing on representative tasks and sampling more instances from these might be more beneficial in low-budget scenarios. Even with an ample budget, scaling tasks should be done judiciously, as close to 97% of performance achievable by using the entire FLAN collection can be attained with just a subset of 16 representative tasks alone with $N' = 200000$.

4.7 Comparison with FLANv2 Mix

We now compare SMART with FLANv2-mix i.e., by using the mixture weights suggested by Longpre et al. (2023) where the weights 40%, 32%, 20%, 5% and 3% are assigned to the 5 sub mixtures of FLAN 2022 — FLANv1, T0, NIV2, CoT

N'	FLANv2 Mix	SMART
25000	**40.05**	39.8
50000	41.49	**43.03**
100000	43.18	**44.96**
200000	44.73	**46.7**
400000	46.26	**47.65**

Table 3: Comparison of SMART with mixture weights of Longpre et al. (2023) for Llama-2-7b. The scores are weighted average of exact match on MMLU and BBH.

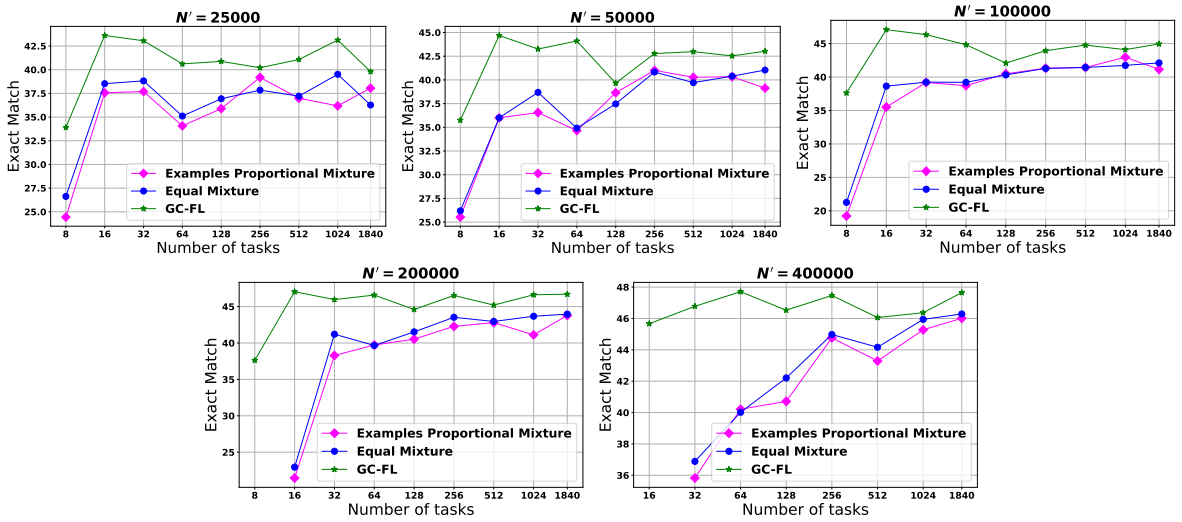

Figure 1: Task-Scaling Curves in the case of Llama-2-7b for different N's, where X-axis is the number of tasks (M') and Y-axis is the weighted average of Exact Matches on MMLU and BBH.

and Dialog respectively and instances are randomly sampled from each sub mixture. Since this method only prescribes weights on sub mixtures and not on individual tasks, we only consider the case where $M' = M$. Table 3 contains comparison of SMART vs FLANv2-mix for Llama-2-7b. FLANv2-mix seems to perform better than EPM (Baseline-1) and EM (Baseline-2) in some cases although SMART almost always outperforms FLANv2-mix.

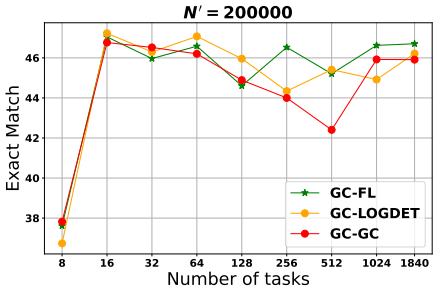

Figure 2: Varying f_2 when f_1 is Graph Cut

4.8 Ablation Study: Submodular Function

Both Stage-1 and Stage-2 of SMART require us to choose submodular functions f_1 and f_2 - which are best treated as hyperparameters because of uncertainty with respect to which functions are best suited for instruction tuning (Section 3.3). We conduct a grid search on f_1 and f_2 using three functions from Table 1, by setting $M' = M$ and varying N' in $\{25000, 50000, 100000, 200000\}$. In this work, we use $\lambda = 0.4$ for Graph-Cut. The grid search (summarized in Table 4) suggests that Graph Cut is optimal for task subset selection, while Facility Location is best for instance selection when $M' = M$. The task scaling curves for each combination of f_1 and f_2 (for different N's) are present in Appendix B. However, in this section, we highlight the case where f_1 is Graph Cut and f_2 is varied. We find that optimal f_2 in this case also depends on M'. For instance, in Figure 2 where $N' = 200000$, Facility Location performs the best at higher M', highlighting the importance of representation, while Graph Cut and Log Determinant show better performance at lower M', highlighting the importance of diversity. We hypothesize this is because — for higher M's, each task on average gets a relatively low budget and the need for representation dominates the need for diversity; however when there is sufficient enough budget for each task, i.e., for lower M's, the need for diversity takes over.

$N' = 25000$

f_1 \ f_2	FL	GC	LOGDET
FL	28.81	27.02	28.05
GC	39.8	40.84	40.61
LOGDET	38.83	35.52	**41.38**

$N' = 50000$

f_1 \ f_2	FL	GC	LOGDET
FL	25.31	25.95	28.35
GC	**43.03**	40.98	42.28
LOGDET	41.97	40.67	41.84

$N' = 10000$

f_1 \ f_2	FL	GC	LOGDET
FL	25.42	25.2	26.89
GC	**44.96**	43.92	43.61
LOGDET	43.63	42.25	42.81

$N' = 200000$

f_1 \ f_2	FL	GC	LOGDET
FL	43.58	41.67	42.72
GC	**46.7**	45.91	46.21
LOGDET	43.75	43.86	43.82

Table 4: Grid Search on submodular functions f_1 and f_2 for Llama-2-7b where weighted average of exact match on MMLU and BBH are compared for different choices of f_1 and f_2. FL denotes Facility Location, GC denotes Graph Cut and LOGDET denotes Log-Determinant.

4.9 Ablation Study: LLM

We also test the effectiveness of SMART strategy on two other LLMs - Falcon-7B and Mistral-7B. Table 5 presents a comparison of SMART mixture with the baseline mixtures for Falcon and Mistral when $M' = M$. For Falcon, we only report MMLU-zero-shot since Falcon gets an exact match of 0.0 for BBH-zero-shot even after fine-tuning. The task-scaling curves for these models are present in Appendix C.

N'	Data Mix.	Mistral	Falcon*
25000	EPM(Baseline-1)	52.69	23.09
	EM(Baseline-2)	51.65	21.1
	FLANv2-mix	53.24	**26.16**
	SMART	**53.33**	22.73
50000	EPM(Baseline-1)	53.75	19.86
	EM(Baseline-2)	53.16	22.96
	FLANv2-mix	**54.15**	25.66
	SMART	53.8	**26.31**
100000	EPM(Baseline-1)	54.24	20.88
	EM(Baseline-2)	**54.57**	24.85
	FLANv2-mix	54.14	**27.47**
	SMART	53.75	27.22
200000	EPM(Baseline-1)	55.25	24.7
	EM(Baseline-2)	54.71	24.39
	FLANv2-mix	55.27	27.63
	SMART	**55.4**	**30.96**

Table 5: Comparison of SMART with baselines for Mistral and Falcon. The scores correspond to weighted averages of exact match on MMLU and BBH.

4.10 Discussion

In Section 4.8, we saw that Graph Cut proves to be the most effective for selecting the weighted task subsets. Figures 6, 7, 8 of Appendix D contain t-SNE visualizations for task subsets selected by the three submodular functions listed in Table 1. Facility Location being a sum-max formulation, a single point is sufficient to represent a cluster and hence it gives more weightage to cluster centers and very less weightage to others. As a result, the submodular gains for first few selected points are very high and then the gains quickly become very small for other points in case of facility location. Log Determinant on the other hand predominantly selects diverse points not taking representation into account. Graph Cut, which models both, hence performs better than both Facility Location and Log Determinant for selecting tasks.

Further, in Figure 1 of Section 4.6 we saw that, at around 16 tasks, there is a peak in performance, after which there is a slight decline. To facilitate more investigation into what tasks are assigned more weightage, Table 6 and Table 7 list down the top-128 and last-128 tasks in the submodular ordering obtained using graph cut. While the former set contains more traditional NLP tasks like Natural Language Inference, Next Sentence Prediction, Question Answering, Summarization, *etc* ., the latter set mostly consists of tasks like Program Execution.

5 Related Work

5.1 Data for Instruction Tuning

Following Wang et al. (2023b), we summarize the related work in three categories — data quality, data quantity, and task composition.

Data Quantity Research diverges on scaling instruction data quantity, with some advocating for limited data (Zhou et al., 2023; Chen et al., 2023a) to expose pretraining knowledge, while others argue for scaling up (Wei et al., 2021; Sanh et al., 2021). According to Ji et al. (2023); Dong et al. (2023); Yuan et al. (2023); Song et al. (2023), the impact of scaling varies across tasks and model abilities. AlShikh et al. (2023) also introduce a metric for instruction following ability, and suggest an early stopping criterion for instruction-tuning.

Data Quality High-quality data is crucial in instruction tuning (Chia et al., 2023; Ding et al., 2023; Zhou et al., 2023). Wang et al. (2023a) use perplexity to select suitable instructions generated by models, while Li et al. (2023a) employ the language model itself to augment and curate high-quality training examples to improve its own performance. Cao et al. (2023) propose InstructionMining, utilizing natural language indicators to predict inference loss as a proxy for data quality without human intervention and select the best subset based on this. Chen et al. (2023b) introduce AlpaGasus, which uses a strong LLM to select high-quality subsets. Lu et al. (2023) and Madaan et al. (2024) also leverage the power of fine-tuned LLM itself to evaluate the quality of instructions. Attendu and Corbeil (2023) employs dynamic data subset selection by filtering out unimportant samples *during* finetuning, based on an extended EL2N metric (Paul et al., 2021; Fayyaz et al., 2022). Taori et al. (2023) propose #InsTag to assess instruction diversity in SFT datasets using ChatGPT. Additionally, Wan et al. (2023) propose Explore-Instruct, utilizing LLMs to

actively explore domain-specific spaces and gather diverse instruction-tuning data. Wu et al. (2023) select new data points that are distinct from existing ones in the model embedding space, augmenting the training dataset iteratively to enhance diversity within subsets.

Task Composition Many previous works show the benefit of scaling the number of tasks (Wei et al., 2021; Chung et al., 2022; Wang et al., 2022; Sanh et al., 2021). However, works like Iyer et al. (2022); Longpre et al. (2023) have also acknowledged that task balancing is also very important for effective instruction tuning. Dong et al. (2023) explore data composition across GSM8k, Code Alpaca, and ShareGPT datasets, finding differential impacts of data scaling on performance across abilities and propose a Dual-stage mixed fine-tuning strategy as a promising solution to activate multiple abilities efficiently. Ivison et al. (2022) identifies relevant multitask subsets based on the similarity between the pre-trained model's representations, using a small amount of target task data. Yin et al. (2023) uses instruction representations for task selection, acting as a replay strategy to mitigate catastrophic forgetting and improve generalization in continual learning. Yue et al. (2023) construct math generalist models via instruction tuning on hybrid of chain-of-thought and program-of-thought rationales in math. Lou et al. (2023); Zhang et al. (2023) provide a survey of instruction tuning in general and Wang et al. (2023b) provide a detailed survey of data management for instruction tuning.

5.2 Submodularity for Subset Selection

Submodularity (Fujishige, 2005) has a long history in combinatorial optimization, game theory, economics etc (Edmonds, 1970; Lovász, 1983; Carter, 2001; Topkis, 1998). It has recently gained traction in machine learning where it has been used for data subset selection for machine translation (Kirchhoff and Bilmes, 2014), speech recognition (Wei et al., 2014; Mittal et al., 2022; Kothawade et al., 2023), efficient pre-training of language models (Renduchintala et al., 2023), active learning (Wei et al., 2015; Kothawade et al., 2021), hyperparameter tuning (Killamsetty et al., 2022), domain adaptation (Karanam et al., 2022), computer vision (Kaushal et al., 2019), continual learning (Tiwari et al., 2022) etc. For a more detailed review of submodularity and its applications, please refer to the survey by Bilmes (2022).

6 Conclusion & Future Work

In this paper, we introduced SMART — a novel data mixture strategy for instruction tuning that utilizes a submodular function to assign importance scores to tasks, determine the mixture weights, and also select non-redundant samples from each task. Further, we also reveal that in a low-budget setting, splitting the budget among a small subset of representative tasks yields superior performance when compared to dividing it among all tasks, which suggests that task scaling should be done more judiciously. Future work could explore making this method more model-specific (*e.g.*, modify task embedding computation) and also possibly modify the approach for targeted instruction-data selection for creating expert LLMs that specialize in specific skills such as math, code etc.

7 Limitations

While the approach has its own advantages of computing the data mixture only once for a given dataset and using it for as many LLMs as one may wish, the approach might benefit from taking inputs from the language model as well and perform model specific instruction tuning. Secondly, SMART shows that there is an optimal number of tasks at which there is peak in performance observed. However, it doesn't say anything on how to find the optimal point as it may depend on language model, the total budget and most importantly the underlying dataset.

8 Ethical Considerations

While instruction tuning leverages pretrained language models and encompasses similar considerations, we anticipate that this approach will predominantly yield positive outcomes. It offers an enhanced methodology for task balancing, potentially allowing for more cost-effective fine-tuning of large language models compared to conventional data mixture strategies.

Acknowledgements

The authors acknowledge the use of ChatGPT[3] for solely paraphrasing and summarizing some parts of the paper and declare that *none* of the generated content is presented in the paper without rigorous manual checking.

[3]https://chat.openai.com/

References

Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C Fowlkes, Stefano Soatto, and Pietro Perona. 2019. Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6430–6439.

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. 2023. The falcon series of open language models. *arXiv preprint arXiv:2311.16867*.

Waseem AlShikh, Manhal Daaboul, Kirk Goddard, Brock Imel, Kiran Kamble, Parikshith Kulkarni, and Melisa Russak. 2023. Becoming self-instruct: introducing early stopping criteria for minimal instruct tuning. *arXiv preprint arXiv:2307.03692*.

Jean-Michel Attendu and Jean-Philippe Corbeil. 2023. Nlu on data diets: Dynamic data subset selection for nlp classification tasks. *arXiv preprint arXiv:2306.03208*.

Jeff Bilmes. 2022. Submodularity in machine learning and artificial intelligence. *arXiv preprint arXiv:2202.00132*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Yihan Cao, Yanbin Kang, and Lichao Sun. 2023. Instruction mining: High-quality instruction data selection for large language models. *arXiv preprint arXiv:2307.06290*.

Michael Carter. 2001. *Foundations of mathematical economics*. MIT press.

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28:41–75.

Hao Chen, Yiming Zhang, Qi Zhang, Hantao Yang, Xiaomeng Hu, Xuetao Ma, Yifan Yanggong, and Junbo Zhao. 2023a. Maybe only 0.5% data is needed: A preliminary exploration of low training data instruction tuning. *arXiv preprint arXiv:2305.09246*.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, et al. 2023b. Alpagasus: Training a better alpaca with fewer data. *arXiv preprint arXiv:2307.08701*.

Yew Ken Chia, Pengfei Hong, Lidong Bing, and Soujanya Poria. 2023. Instructeval: Towards holistic evaluation of instruction-tuned large language models. *arXiv preprint arXiv:2306.04757*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Tri Dao. 2023. FlashAttention-2: Faster attention with better parallelism and work partitioning.

Alexandre De Brebisson and Pascal Vincent. 2015. An exploration of softmax alternatives belonging to the spherical loss family. *arXiv preprint arXiv:1511.05042*.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*.

Guanting Dong, Hongyi Yuan, Keming Lu, Chengpeng Li, Mingfeng Xue, Dayiheng Liu, Wei Wang, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2023. How abilities in large language models are affected by supervised fine-tuning data composition. *arXiv preprint arXiv:2310.05492*.

Jack Edmonds. 1970. Matroids, submodular functions and certain polyhedra. *Combinatorial Structures and Their Applications*, pages 69–87.

Mohsen Fayyaz, Ehsan Aghazadeh, Ali Modarressi, Mohammad Taher Pilehvar, Yadollah Yaghoobzadeh, and Samira Ebrahimi Kahou. 2022. Bert on a data diet: Finding important examples by gradient-based pruning. *arXiv preprint arXiv:2211.05610*.

Uriel Feige. 1998. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652.

Marshall L Fisher, George L Nemhauser, and Laurence A Wolsey. 1978. *An analysis of approximations for maximizing submodular set functions—II*. Springer.

Satoru Fujishige. 2005. *Submodular functions and optimization*. Elsevier.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Hamish Ivison, Noah A Smith, Hannaneh Hajishirzi, and Pradeep Dasigi. 2022. Data-efficient finetuning using cross-task nearest neighbors. *arXiv preprint arXiv:2212.00196*.

Srinivasan Iyer, Xi Victoria Lin, Ramakanth Pasunuru, Todor Mihaylov, Daniel Simig, Ping Yu, Kurt Shuster, Tianlu Wang, Qing Liu, Punit Singh Koura, et al. 2022. Opt-iml: Scaling language model instruction meta learning through the lens of generalization. *arXiv preprint arXiv:2212.12017*.

Yunjie Ji, Yong Deng, Yan Gong, Yiping Peng, Qiang Niu, Lei Zhang, Baochang Ma, and Xiangang Li. 2023. Exploring the impact of instruction data scaling on large language models: An empirical study on real-world use cases. *arXiv preprint arXiv:2303.14742*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Athresh Karanam, Krishnateja Killamsetty, Harsha Kokel, and Rishabh Iyer. 2022. Orient: Submodular mutual information measures for data subset selection under distribution shift. *Advances in neural information processing systems*, 35:31796–31808.

Vishal Kaushal, Rishabh Iyer, Suraj Kothawade, Rohan Mahadev, Khoshrav Doctor, and Ganesh Ramakrishnan. 2019. Learning from less data: A unified data subset selection and active learning framework for computer vision. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1289–1299. IEEE.

Vishal Kaushal, Ganesh Ramakrishnan, and Rishabh Iyer. 2022. Submodlib: A submodular optimization library. *arXiv preprint arXiv:2202.10680*.

Krishnateja Killamsetty, Guttu Sai Abhishek, Aakriti Lnu, Ganesh Ramakrishnan, Alexandre Evfimievski, Lucian Popa, and Rishabh Iyer. 2022. Automata: Gradient based data subset selection for compute-efficient hyper-parameter tuning. *Advances in Neural Information Processing Systems*, 35:28721–28733.

Katrin Kirchhoff and Jeff Bilmes. 2014. Submodularity for data selection in machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 131–141.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Suraj Kothawade, Nathan Beck, Krishnateja Killamsetty, and Rishabh Iyer. 2021. Similar: Submodular information measures based active learning in realistic scenarios. *Advances in Neural Information Processing Systems*, 34:18685–18697.

Suraj Kothawade, Anmol Mekala, D Chandra Sekhara Hetha Havya, Mayank Kothyari, Rishabh Iyer, Ganesh Ramakrishnan, and Preethi Jyothi. 2023. Ditto: Data-efficient and fair targeted subset selection for asr accent adaptation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5810–5822.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. 2023a. Self-alignment with instruction backtranslation. *arXiv preprint arXiv:2308.06259*.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023b. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*.

Renze Lou, Kai Zhang, and Wenpeng Yin. 2023. Is prompt all you need? no. a comprehensive and broader view of instruction learning. *arXiv preprint arXiv:2303.10475*.

László Lovász. 1983. Submodular functions and convexity. *Mathematical Programming The State of the Art: Bonn 1982*, pages 235–257.

Jianqiao Lu, Wanjun Zhong, Wenyong Huang, Yufei Wang, Fei Mi, Baojun Wang, Weichao Wang, Lifeng Shang, and Qun Liu. 2023. Self: Language-driven self-evolution for large language model. *arXiv preprint arXiv:2310.00533*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Michel Minoux. 2005. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques: Proceedings of the 8th IFIP Conference on Optimization Techniques Würzburg, September 5–9, 1977*, pages 234–243. Springer.

Ashish Mittal, Durga Sivasubramanian, Rishabh Iyer, Preethi Jyothi, and Ganesh Ramakrishnan. 2022. Partitioned gradient matching-based data subset selection for compute-efficient robust asr training. *arXiv preprint arXiv:2210.16892*.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14:265–294.

Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. 2021. Deep learning on a data diet: Finding important examples early in training. *Advances in Neural Information Processing Systems*, 34:20596–20607.

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. *The Journal of Machine Learning Research*, 21(1):5485–5551.

H. S. V. N. S. Kowndinya Renduchintala, Krishnateja Killamsetty, Sumit Bhatia, Milan Aggarwal, Ganesh Ramakrishnan, Rishabh K. Iyer, and Balaji Krishnamurthy. 2023. INGENIOUS: using informative data subsets for efficient pre-training of language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 6690–6705. Association for Computational Linguistics.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. 2021. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*.

Chiyu Song, Zhanchao Zhou, Jianhao Yan, Yuejiao Fei, Zhenzhong Lan, and Yue Zhang. 2023. Dynamics of instruction tuning: Each ability of large language models has its own growth pace. *arXiv preprint arXiv:2310.19651*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

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.

Rishabh Tiwari, Krishnateja Killamsetty, Rishabh Iyer, and Pradeep Shenoy. 2022. Gcr: Gradient coreset based replay buffer selection for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 99–108.

Donald M Topkis. 1998. *Supermodularity and complementarity*. Princeton university press.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Alan M. Turing. 1950. Computing Machinery and Intelligence. *Mind*, 59(October):433–60. Publisher: Oxford University Press.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, and Daniel Cer. 2021. Spot: Better frozen model adaptation through soft prompt transfer. *arXiv preprint arXiv:2110.07904*.

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. *arXiv preprint arXiv:2005.00770*.

Fanqi Wan, Xinting Huang, Tao Yang, Xiaojun Quan, Wei Bi, and Shuming Shi. 2023. Explore-instruct: Enhancing domain-specific instruction coverage through active exploration. *arXiv preprint arXiv:2310.09168*.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. 2022. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705*.

Yue Wang, Xinrui Wang, Juntao Li, Jinxiong Chang, Qishen Zhang, Zhongyi Liu, Guannan Zhang, and Min Zhang. 2023a. Harnessing the power of david against goliath: Exploring instruction data generation without using closed-source models. *arXiv preprint arXiv:2308.12711*.

Zige Wang, Wanjun Zhong, Yufei Wang, Qi Zhu, Fei Mi, Baojun Wang, Lifeng Shang, Xin Jiang, and Qun Liu. 2023b. Data management for large language models: A survey. *arXiv preprint arXiv:2312.01700*.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Kai Wei, Rishabh Iyer, and Jeff Bilmes. 2015. Submodularity in data subset selection and active learning. In *International conference on machine learning*, pages 1954–1963. PMLR.

Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. 2014. Unsupervised submodular subset selection for speech data. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4107–4111. IEEE.

Shengguang Wu, Keming Lu, Benfeng Xu, Junyang Lin, Qi Su, and Chang Zhou. 2023. Self-evolved diverse data sampling for efficient instruction tuning. *arXiv preprint arXiv:2311.08182*.

Zhiheng Xi, Rui Zheng, Yuansen Zhang, Xuan-Jing Huang, Zhongyu Wei, Minlong Peng, Mingming Sun, Qi Zhang, and Tao Gui. 2023. Connectivity patterns are task embeddings. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 11993–12013.

Da Yin, Xiao Liu, Fan Yin, Ming Zhong, Hritik Bansal, Jiawei Han, and Kai-Wei Chang. 2023. Dynosaur: A dynamic growth paradigm for instruction-tuning data curation. *arXiv preprint arXiv:2305.14327*.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. 2023. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2023. Lima: Less is more for alignment. *arXiv preprint arXiv:2305.11206*.

Wangchunshu Zhou, Canwen Xu, and Julian McAuley. 2022. Efficiently tuned parameters are task embeddings. *arXiv preprint arXiv:2210.11705*.

APPENDIX

A Code and Data

Our code we used for instruction tuning and for creating data mixtures is open-sourced at https://github.com/kowndinya-renduchintala/SMART. The code development utilized open-source tools, primarily relying on the HuggingFace library for model training with PyTorch as the underlying framework. Both PyTorch and HuggingFace are licensed under permissive licenses, with PyTorch under the BSD license and HuggingFace under the Apache 2.0 license. Additionally, submodular optimization was performed using SUBMODLIB, which is an openly accessible library on GitHub at https://github.com/decile-team/submodlib under the MIT license.

B Task Scaling Curves: Varying f_1 and f_2

Figure 3 consists of task scaling curves for the 9 possible combinations (Section 4.8) of f_1 and f_2.

C Task Scaling Curves: Mistral, Falcon

Figure 4 and Figure 5 consist of task scaling curves for Mistral-7B and Falcon-7B respectively (Section 4.9).

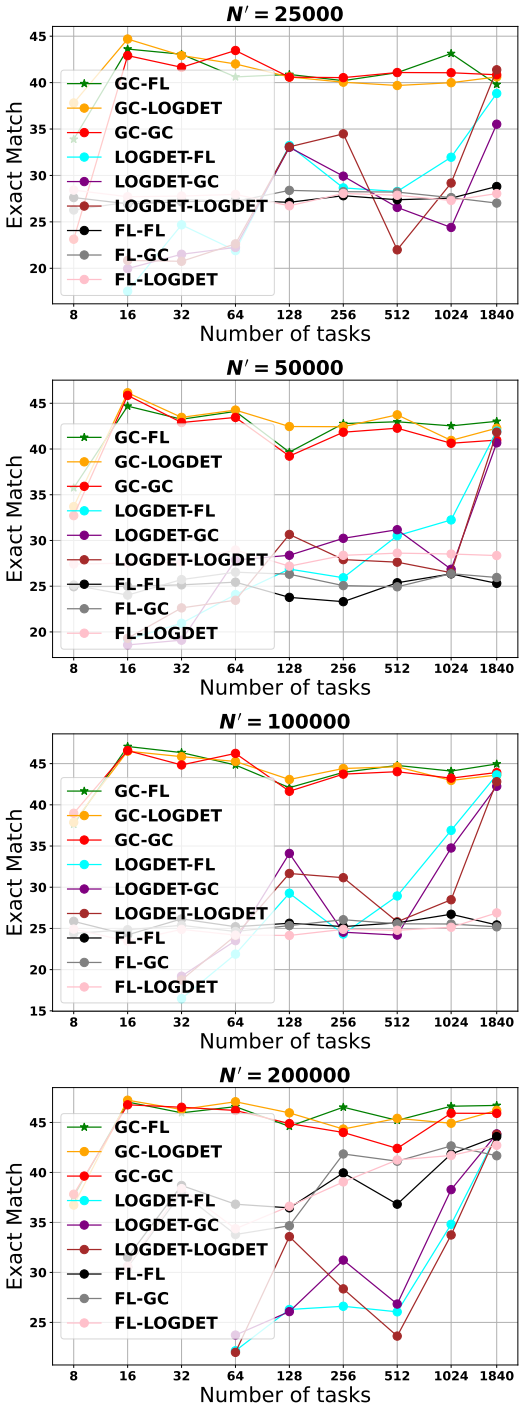

Figure 3: Task Scaling Curves for various f_1 and f_2 combinations for Llama-2-7b

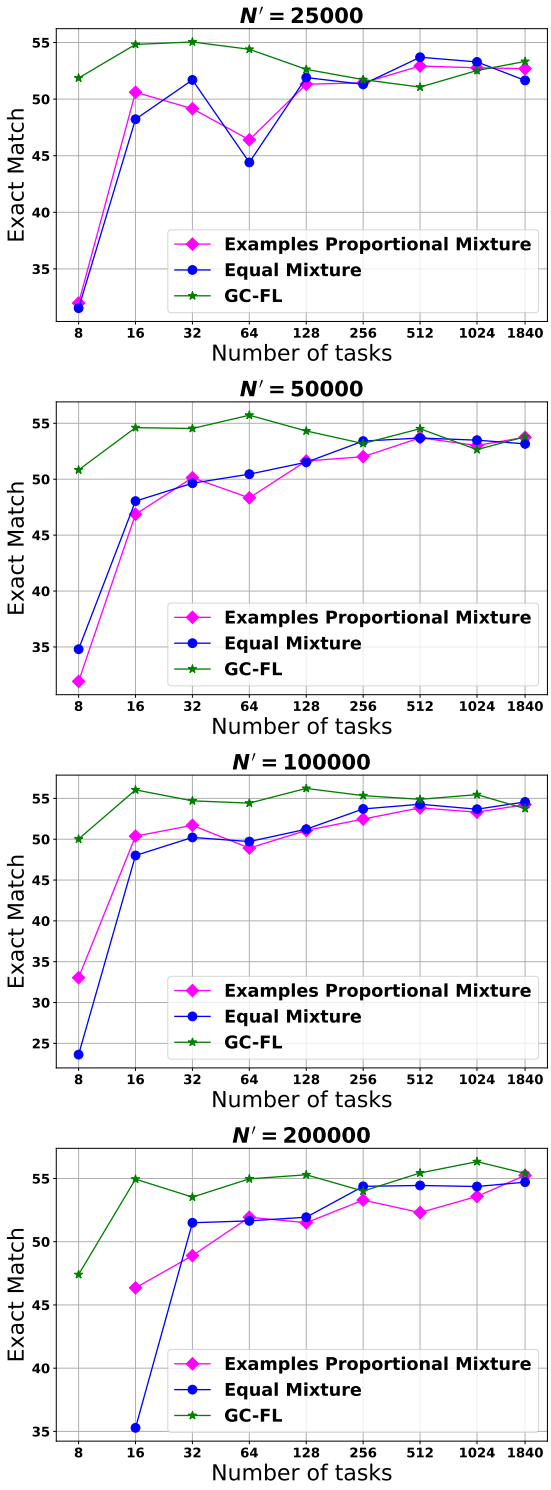

Figure 4: Task Scaling Curves for Mistral-7B

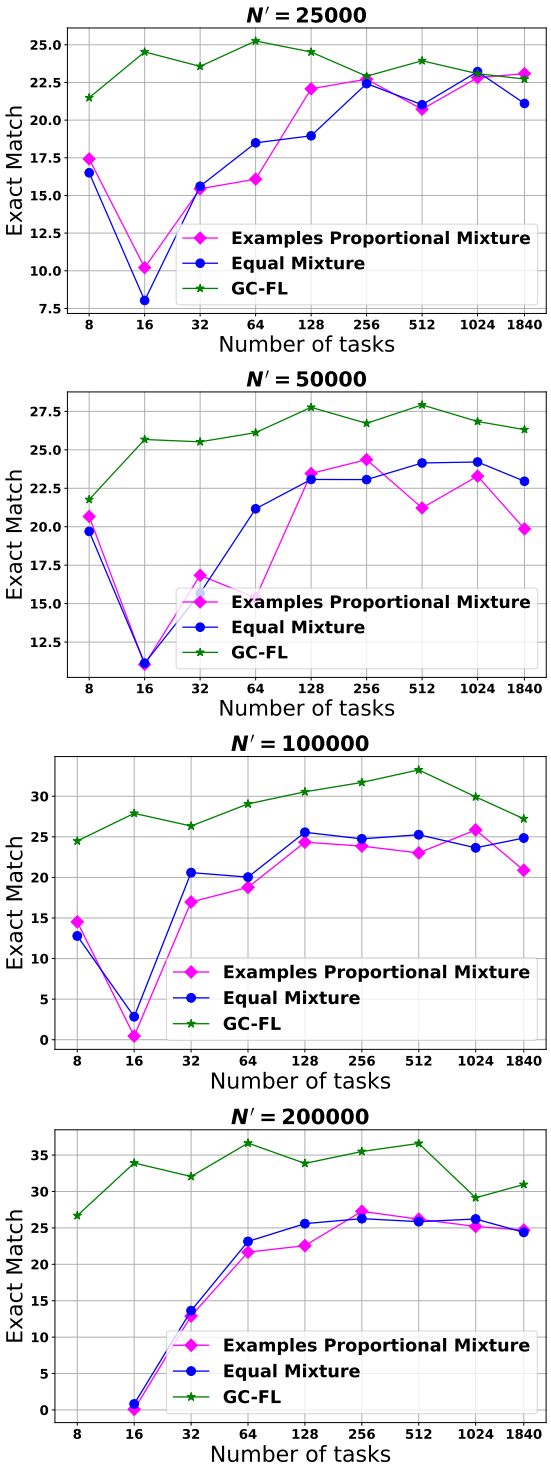

Figure 5: Task Scaling Curves for falcon-7B

D Visualization of Task Subsets

As pointed out in Section 2, the three submodular functions in Table 1 are different. Facility Location predominantly models representation; Graph Cut models a trade-off between representation and diversity; Log Determinant predominantly models diversity. To better visualize this, we present t-SNE plots of 1840 tasks present in the FLAN 2022 collection (Longpre et al., 2023) and highlight the tasks selected by these functions for different values of M'. Figure 6, Figure 7 and Figure 8 respectively contains the visualizations for Facility Location, Graph Cut and Log Determinant.

Rank	Task
1	anli/r3:0.1.0
2	hellaswag:1.1.0
3	task1392_superglue_multirc_answer_verification
4	task955_wiki_auto_style_transfer
5	task381_boolq_question_generation
6	task1291_multi_news_summarization
7	task1295_adversarial_qa_question_answering
8	task519_aquamuse_question_generation
9	anli/r2:0.1.0
10	anli/r1:0.1.0
11	race_high_Select_the_best_answer_no_instructions_
12	cot_ecqa_ii
13	super_glue/rte:1.0.2
14	task1660_super_glue_question_generation
15	task339_record_answer_generation
16	task302_record_classification
17	stream_qed_ii
18	task870_msmarco_answer_generation
19	paws_wiki:1.1.0
20	super_glue/multirc:1.0.2
21	task603_wikitext-103_fill_in_the_blank
22	task887_quail_answer_generation
23	stream_qed
24	task380_boolq_yes_no_question
25	coqa:1.0.0
26	task1412_web_questions_question_answering
27	super_glue/cb:1.0.2
28	task1293_kilt_tasks_hotpotqa_question_answering
29	quail_context_question_description_answer_text
30	fix_punct
31	true_case
32	winogrande:1.1.0
33	glue/wnli:2.0.0
34	super_glue/record:1.0.2
35	cot_ecqa
36	quail_context_question_description_answer_id
37	task919_coqa_incorrect_answer_generation
38	task520_aquamuse_answer_given_in_passage
39	task1290_xsum_summarization
40	task1609_xquad_en_question_generation
41	squad/v1.1:3.0.0
42	task231_iirc_link_classification
43	task349_squad2.0_answerable_unanswerable_question_classification
44	wiki_dialog_ii
45	task1661_super_glue_classification
46	quail_context_description_question_text
47	adversarial_qa_droberta_answer_the_following_q
48	task644_refresd_translation
49	bool_q:1.0.0
50	task470_mrqa_question_generation
51	race_middle_Select_the_best_answer_no_instructions_
52	task770_pawsx_english_text_modification
53	adversarial_qa_dbidaf_answer_the_following_q
54	glue/qnli:2.0.0
55	task1558_jfleg_incorrect_answer_generation
56	task344_hybridqa_answer_generation
57	quoref_Guess_Title_For_Context
58	super_glue/copa:1.0.2
59	word_segment
60	task1340_msr_text_compression_compression
61	wiki_dialog
62	squad/v2.0:3.0.0
63	task1530_scitail1.1_sentence_generation
64	task225_english_language_answer_generation
65	task303_record_incorrect_answer_generation
66	gem/web_nlg_en:1.1.0
67	task768_qed_text_span_selection
68	adversarial_qa_dbert_answer_the_following_q
69	task1389_hellaswag_completion
70	task1294_wiki_qa_answer_verification
71	wiki_qa_Topic_Prediction_Answer_Only
72	task596_mocha_question_generation
73	task871_msmarco_question_generation
74	task1564_triviaqa_answer_generation
75	task1344_glue_entailment_classification
76	cosmos_qa:1.0.0
77	task1555_scitail_answer_generation
78	task1557_jfleg_answer_generation
79	gem/common_gen:1.1.0
80	task238_iirc_answer_from_passage_answer_generation
81	task233_iirc_link_exists_classification
82	super_glue/wic:1.0.2
83	task1345_glue_qqp_question_paraphrashing
84	glue/qqp:2.0.0
85	glue/stsb:2.0.0
86	task595_mocha_answer_generation
87	task460_qasper_answer_generation
88	glue/mnli:2.0.0
89	task051_multirc_correct_answer_single_sentence
90	adversarial_qa_droberta_tell_what_it_is
91	task547_alt_translation_entk_en
92	quail_no_prompt_id
93	task311_race_question_generation
94	cot_sensemaking_ii
95	gem/dart:1.1.0
96	wiki_qa_Jeopardy_style
97	adversarial_qa_dbidaf_tell_what_it_is
98	task1593_yahoo_answers_topics_classification
99	quail_context_question_answer_description_text
100	cot_creak_ii
101	definite_pronoun_resolution:1.1.0
102	gigaword:1.2.0
103	super_glue/wsc.fixed:1.0.2
104	task234_iirc_passage_line_answer_generation
105	task556_alt_translation_en_ja
106	task604_flores_translation_entosn
107	adversarial_qa_dbert_tell_what_it_is
108	task310_race_classification
109	task1594_yahoo_answers_topics_question_generation
110	story_cloze/2016:1.0.0
111	task933_wiki_auto_style_transfer
112	gem/wiki_lingua_english_en:1.1.0
113	task1382_quarel_write_correct_answer
114	task1296_wiki_hop_question_answering
115	quail_context_description_question_answer_text
116	glue/cola:2.0.0
117	cot_strategyqa_ii
118	task1553_cnn_dailymail_summarization
119	openbookqa:0.1.0
120	task054_multirc_write_correct_answer
121	task1218_ted_translation_en_ja
122	quail_no_prompt_text
123	quail_context_question_description_text
124	task550_discofuse_sentence_generation
125	quail_context_question_answer_description_id
126	task1608_xquad_en_answer_generation
127	task1520_qa_srl_answer_generation
128	cos_e_v1.11_generate_explanation_given_text

Table 6: List of 128 most representative tasks in FLAN-2022 collection as ordered by the Graph Cut

Rank	Task
1713	task261_spl_translation_es_en
1714	task1384_deal_or_no_dialog_classification
1715	task854_hippocorpus_classification
1716	task110_logic2text_sentence_generation
1717	task360_spolin_yesand_response_generation
1718	task148_afs_argument_quality_gay_marriage
1719	task499_extract_and_add_all_numbers_from_list
1720	task176_break_decompose_questions
1721	task085_unnatural_addsub_arithmetic
1722	task108_contextualabusedetection_classification
1723	task472_haspart_classification
1724	task856_conv_ai_2_classification
1725	task600_find_the_longest_common_substring_in_two_strings
1726	task150_afs_argument_quality_gun_control
1727	task1508_wordnet_antonyms
1728	task183_rhyme_generation
1729	task488_extract_all_alphabetical_elements_from_list_in_order
1730	task682_online_privacy_policy_text_classification
1731	task1425_country_iso_numeric
1732	task756_find_longert_substring_and_return_all_unique_alphabets_in_it
1733	task1585_root09_hypernym_generation
1734	task958_e2e_nlg_text_generation_parse
1735	task584_udeps_eng_fine_pos_tagging
1736	task1319_country_by_barcode_prefix
1737	task1507_boolean_temporal_reasoning
1738	task509_collate_of_all_alphabetical_and_numerical_elements_in_list_separately
1739	task064_all_elements_except_first_i
1740	task130_scan_structured_text_generation_command_action_long
1741	task365_synthetic_remove_vowels
1742	task149_afs_argument_quality_death_penalty
1743	task210_logic2text_structured_text_generation
1744	task1495_adverse_drug_event_classification
1745	task684_online_privacy_policy_text_information_type_generation
1746	task1426_country_independence_year
1747	task126_scan_structured_text_generation_command_action_all
1748	task605_find_the_longest_common_subsequence_in_two_lists
1749	task128_scan_structured_text_generation_command_action_short
1750	task960_ancora-ca-ner_named_entity_recognition
1751	task078_all_elements_except_last_i
1752	task1427_country_region_in_world
1753	task063_first_i_elements
1754	task956_leetcode_420_strong_password_check
1755	task683_online_privacy_policy_text_purpose_answer_generation
1756	task091_all_elements_from_index_i_to_j
1757	task1542_every_ith_element_from_starting
1758	task1506_celebrity_minimal_dob_span
1759	task245_check_presence_in_set_intersection
1760	task497_extract_all_numbers_from_list_in_order
1761	task1428_country_surface_area
1762	task092_check_prime_classification
1763	task1088_array_of_products
1764	task1332_check_leap_year
1765	task127_scan_long_text_generation_action_command_all
1766	task129_scan_long_text_generation_action_command_short
1767	task1322_country_government_type
1768	task1331_reverse_array
1769	task131_scan_long_text_generation_action_command_long
1770	task371_synthetic_product_of_list
1771	task1189_check_char_in_string
1772	task208_combinations_of_list
1773	task211_logic2text_classification
1774	task1551_every_ith_element_from_kth_element
1775	task1194_kth_largest_element
1776	task101_reverse_and_concatenate_all_elements_from_index_i_to_j

Rank	Task
1777	task1404_date_conversion
1778	task504_count_all_alphabetical_elements_in_list
1779	task087_new_operator_addsub_arithmetic
1780	task850_synthetic_longest_palindrome
1781	task099_reverse_elements_between_index_i_and_j
1782	task206_collatz_conjecture
1783	task505_count_all_numerical_elements_in_list
1784	task1405_find_median
1785	task267_concatenate_and_reverse_all_elements_from_index_i_to_j
1786	task207_max_element_lists
1787	task1443_string_to_number
1788	task1188_count_max_freq_char
1789	task212_logic2text_classification
1790	task374_synthetic_pos_or_neg_calculation
1791	task1190_add_integer_to_list
1792	task243_count_elements_in_set_intersection
1793	task636_extract_and_sort_unique_alphabets_in_a_list
1794	task124_conala_pair_averages
1795	task1150_delete_max_min
1796	task755_find_longest_substring_and_replace_its_sorted_lowercase_version_in_both_lists
1797	task100_concatenate_all_elements_from_index_i_to_j
1798	task372_synthetic_palindrome_numbers
1799	task1148_maximum_ascii_value
1800	task506_position_of_all_alphabetical_elements_in_list
1801	task373_synthetic_round_tens_place
1802	task367_synthetic_remove_floats
1803	task1406_kth_smallest_element
1804	task1333_check_validity_date_ddmmyyyy
1805	task244_count_elements_in_set_union
1806	task205_remove_even_elements
1807	task1320_country_domain_tld
1808	task123_conala_sort_dictionary
1809	task122_conala_list_index_addition
1810	task076_splash_correcting_sql_mistake
1811	task094_conala_calculate_mean
1812	task507_position_of_all_numerical_elements_in_list
1813	task1403_check_validity_date_mmddyyyy
1814	task1315_find_range_array
1815	task098_conala_list_intersection
1816	task1087_two_number_sum
1817	task095_conala_max_absolute_value
1818	task1089_check_monotonic_array
1819	task077_splash_explanation_to_sql
1820	task097_conala_remove_duplicates
1821	task125_conala_pair_differences
1822	task368_synthetic_even_or_odd_calculation
1823	task1151_swap_max_min
1824	task852_synthetic_multiply_odds
1825	task606_sum_of_all_numbers_in_list_between_positions_i_and_j
1826	task090_equation_learner_algebra
1827	task1446_farthest_integers
1828	task096_conala_list_index_subtraction
1829	task868_cfq_mcd1_explanation_to_sql
1830	task369_synthetic_remove_odds
1831	task093_conala_normalize_lists
1832	task370_synthetic_remove_divisible_by_3
1833	task851_synthetic_multiply_evens
1834	task637_extract_and_sort_unique_digits_in_a_list
1835	task1498_24hour_to_12hour_clock
1836	task869_cfq_mcd1_sql_to_explanation
1837	task1445_closest_integers
1838	task1444_round_power_of_two
1839	task366_synthetic_return_primes
1840	task107_splash_question_to_sql

Table 7: List of 128 least representative tasks in FLAN-2022 collection as ordered by the Graph Cut

Figure 6: t-SNE visualizations for task subsets of various sizes (M') selected by Facility Location

Figure 7: t-SNE visualizations for task subsets of various sizes (M') selected by Graph Cut

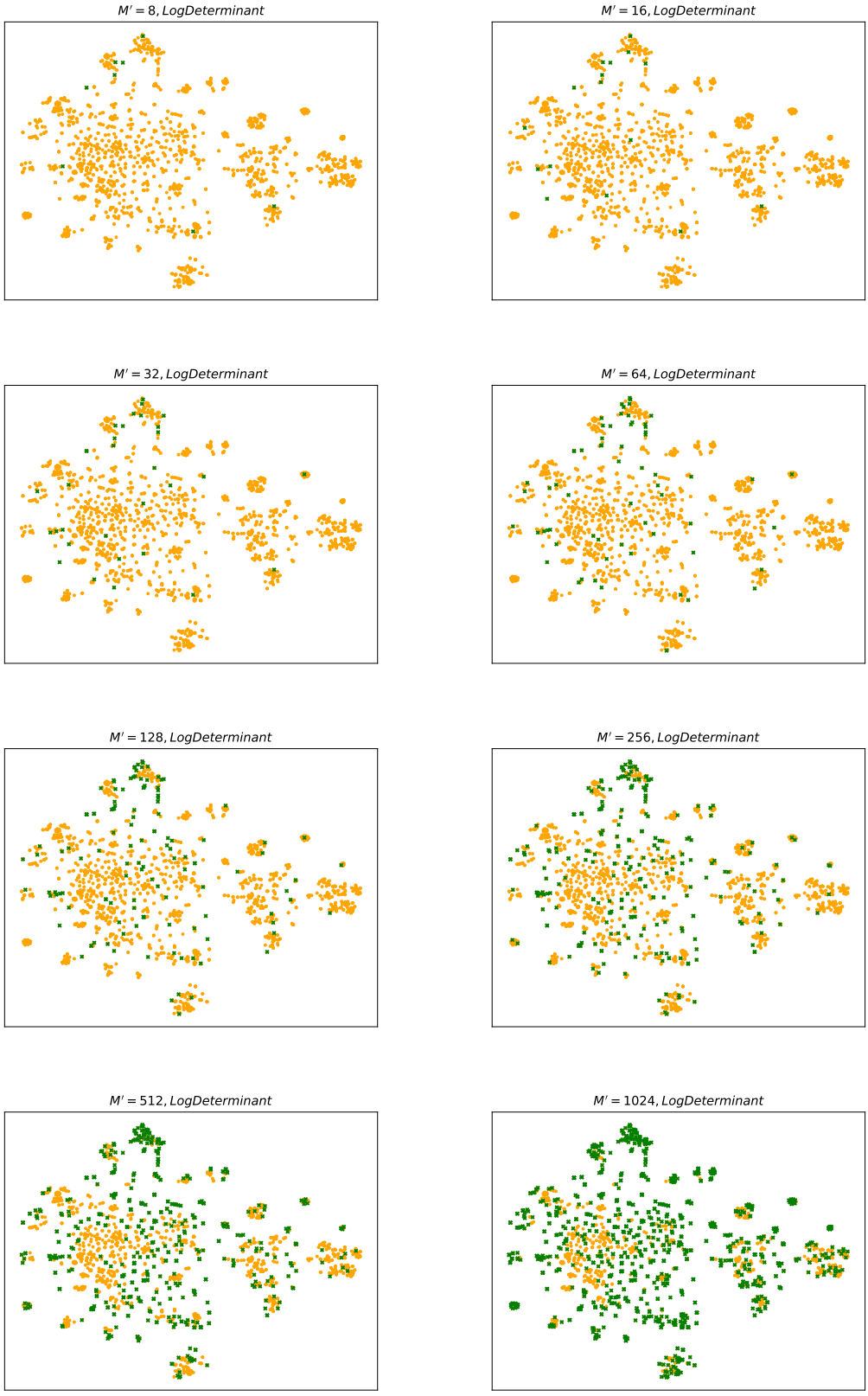

Figure 8: t-SNE visualizations for task subsets of various sizes (M') selected by Log Determinant