A Parameter Distance

We measure the L2 norm of model parameter difference between fine-tuned and pre-trained checkpoints, as a signal of how much the distribution has drifted during SFT ichi Amari (2016); Cover & Thomas (2006). We notice that training over well-matched distribution shifts the parameter less than training over those ill-matched.

	Mistral-7B-v0.3	Llama3.1-8B	Qwen2.5
GRAPE	8.006	8.196	8.426
Worst	8.029	8.202	8.467

Table 6: Performance comparison across different models

B Further Experiments

B.1 Comparing with Reward Based Selection

We compare GRAPE with purely reward-based selection. where for each instruction, we select the response with the highest scalar reward as determined by a reward model - Skywork-Reward-Llama3.1-8B-v0.2. Once the top-ranked response for each instruction is selected, we proceed with standard supervised fine-tuning on the resulting instruction-response pairs.The RFT setup provides a natural contrast to our proposed GRAPE method by emphasizing reward alignment over base-model alignment, thereby enabling us to disentangle the effects of distribution matching versus reward optimization in SFT data selection.

As shown in Table B.1, GRAPE outperforms reward-based selection across both models and all benchmarks. These results suggest that aligning supervision with the base model's own distributional preferences—rather than relying on external reward models—can yield better task performance.

Model	Method	AE WR	AE WR (LC)	LeetCode	MATH	MMLU	BBH	Avg
LLaMA3.1-8B	GRAPE	15.2	14.8	19.4	32.1	64.5	69.6	**35.9**
	Reward	14.0	14.5	17.2	31.3	63.3	69.0	34.9
Mistral-v0.3-7B	GRAPE	13.9	13.6	18.3	24.2	59.2	62.3	**31.9**
	Reward	12.5	13.9	13.3	22.4	58.5	62.2	30.5

Table 7: Performance comparison between GRAPE and reward-based selection across benchmarks for LLaMA3.1-8B and Mistral-v0.3-7B. Metrics are benchmark-specific scores (higher is better).

B.2 Experiment on OpenHermes

To test the generality of our findings beyond the UltraInteract and Tulu-Olmo settings, we conduct additional experiments on the OpenHermes-2.5 (Teknium, 2023) dataset—a large-scale, high-quality instruction-tuning corpus with approximately 1 million distinct instructions.

Following the setup from §5, we apply GRAPE to select from responses aggregated across sources, including Huang et al. (2024) and HuggingFace-H4 (2024). For preference-based datasets, we retain only the winning responses to ensure quality, mirroring our earlier selection protocol. This results in 575K unique instructions and 1.34M instruction-response pairs.

Item	Metric	Data	Llama 3.1-8B	Mistral -7B-v0.3	Qwen 2.5-7B
Alpaca -Eval2	LC	Subset	8.6	5.9	7.6
		Random	8.0	6.2	9.0
		GRAPE	**11.3**	**8.2**	**10.8**
	WR	Subset	6.2	3.9	5.2
		Random	6.4	4.8	7.2
		GRAPE	**9.4**	**7.5**	**9.6**
Truthful -QA	MC2	Subset	51.6	49.0	54.4
		Random	51.4	49.9	55.2
		GRAPE	**52.7**	**51.6**	**56.4**

Table 8: Results on OpenHermes-2.5. The Subset row refers to training exclusively on the SFT responses over the subset.

As shown in Table 8, GRAPE continues to outperform naive combination strategies. The consistent gains across diverse data sources and model families strengthen our central claim: GRAPE is a general-purpose, model-aligned response selection strategy that reliably improves SFT performance in real-world, large-scale instruction tuning.

B.3 Data Selection For Long Chain-of-Thoughts

O1-/R1-style long chain-of-thoughts have drawn increasing attention. This paradigm, exemplified by models like OpenAI's O1 and DeepSeek-R1, has shown remarkable success in challenging domains such as mathematics and coding. We further experiment with the use of GRAPE in long chain-of-thought distillation. We generate multiple candidate trajectories using R1-Distill-Qwen-1.5B DeepSeek-AI et al. (2025) for a subset of OpenR1-Math (Face, 2025) dataset — LUFFY (Yan et al., 2025), and verify the correctness of each, retaining the correct ones.

We compare the results on lowest versus highest perplexity instances below in table 9.

Model	Perplexity	Acc.
Qwen2.5-1.5B	Highest	0.330
Qwen2.5-1.5B	Lowest (GRAPE)	0.396
Qwen2.5-3B	Highest	0.524
Qwen2.5-3B	Lowest (GRAPE)	0.544

Table 9: Performance metrics on MATH dataset

B.4 Token-level GRAPE

To further investigate the alternative uses of our insight, we conduct experiments beyond data selection by incorporating token-level likelihoods directly into the training objective. Specifically, we modified the loss function to weigh each token proportionally to its likelihood. We trained this variant on the OpenThoughts-114k Team (2025) dataset, a curated collection of 114k high-quality reasoning samples spanning domains such as math, science, code etc. Evaluation followed the same benchmark suite as LUFFY Yan et al. (2025), including competition-level math datasets (AIME24/25, AMC, MATH-500, Minerva, OlympiadBench) and general reasoning tests (ARC-c, GPQA-Diamond, MMLU-Pro). Our result in Table 10 show that our token-level likelihood-weighted training yields consistent improvements across several benchmarks.

Benchmark	Baseline (%)	Token-level GRAPE (%)
MATH	55.60	60.20
OLYMPIADBENCH	21.04	28.30
MINERVA	15.07	19.85
AIME-24	3.12	4.90
AMC	23.76	29.74
AIME-25	4.38	3.23
ARC-C	60.84	76.62
GPQA-DIAMOND	7.07	19.70
MMLU-PRO	27.02	34.35

Table 10: Qwen2.5-3B performance (in %) on various benchmarks under Baseline vs. Token-level GRAPE.

C Further Details On Baselines

This section details the experimental setup for our data selection baselines: **S2L**,**LESS** and **NV-Embed**.

C.1 S2L

S2L, a state-of-the-art unsupervised data selection baseline, operates through two key steps: training a reference model to capture training dynamics and clustering the resulting trajectories to form a

diverse, balanced subset of training data. The reference models used in our setup are specifically selected to enhance S2L's performance, adhering to the theoretical underpinnings from the original paper that training dynamics remain consistent across models of varying sizes within the same family.

For our experiments, we train small reference models corresponding to the final target models. Specifically, we pair Llama-3.1-8B with Llama-3.2-1B, Qwen-2.5-7B with Qwen-2.5-0.5B, and Mistral-v0.3-7B with itself due to the lack of smaller models in the Mistral family. To minimize computational costs, LoRA is applied when training the Mistral reference model. **This choice of reference models are better compared to original S2L setup**, which employed a Pythia-70M proxy, thereby improving the fidelity of the selected subset.

Following S2L, the reference models are trained on a random 5% subset of the dataset over four epochs. This reduced training requirement is justified by prior work, which demonstrates that only partial data is sufficient for the proxy model to learn meaningful training dynamics. During trajectory collection, we record the training loss of all examples at intervals of 500 iterations. The batch size and learning rate schedules are set as batch size of 128 and a learning rate warmup of 3%, followed by a cosine decay to 2e-5.

We then perform K-means clustering using the Faiss library to efficiently partition the trajectory space into 100 clusters. The number of iterations is set to 20, and we use the Euclidean distance metric to ensure convergence to well-separated clusters. From each cluster, an equal number of examples are sampled to maintain a balanced subset distribution.

C.2 LESS

LESS is a state-of-the-art model-based and supervised data selection method that leverages gradient-based influence estimation. Given a small set of validation examples per task, LESS computes the influence of each training example by measuring the weighted cosine similarity between their LoRA gradients across multiple warmup checkpoints. It then aggregates these influence scores by averaging over validation examples within each task, followed by taking the maximum across tasks to obtain a scalar utility score per training example. Training examples are selected greedily based on these scores. In our experiments, we use the same base model for both selection and training, and follow the original LESS setup: 5% warmup training for 4 epochs and a gradient projection dimension of 8192.

C.3 NV-Embed

Embedding-based data selection as detailed in (Ivison et al., 2025) is a supervised data selection method that ranks training examples by computing cosine similarity between their embeddings and those of validation examples. Unlike model-aware methods like LESS, embedding-based data selection is model-agnostic: it relies on fixed, pretrained embedding models (in our case, we used NV-embed-v2, the state-of-the-art embedding model) rather than the target model. Instead of aggregating similarity scores into a single utility value per training example, embedding-based data selection uses a round-robin strategy that iteratively selects the highest-scoring example for each validation instance, ensuring diverse coverage across tasks. We follow the original setup from Ivison et al. (2025) in our experiments.

D Further Training Details

We train our models on a 4-GPU Nvidia-GH200 node, with batch size 256 and micro batch size 2.

E Further Ablations on UltraInteract.

See Tables 11 and 4.

F Additional Related Works On Model Dependent Data Selection Approaches

Model-dependent data selection methods leverage internal signals from a target model—such as gradients, embeddings, or log-probabilities—to identify training examples that are most useful for

Data		Full UI	Closest-1	Random-1
Num. Instances		**280K**	**80K**	**80K**
HumanEval		46.3	42.1	41.5 (-)
LeetCode		15.6	13.9	11.1 (-)
MBPP		50.1	52.1	49.1 (-)
MATH	CoT	21.6	19.2	15.5 (-)
	PoT	32.6	24.9	15.1 (-)
GSMPlus	CoT	45.9	44.1	35.3 (-)
	CoT	45.3	43.2	45.2 (-)
TheoremQA	CoT	16.8	15.8	15.8
	PoT	20.1	12.9	15.3
Avg.		***32.7***	***29.8***	***27.1(-)***

Table 11: Ablations on data selection with MISTRAL-7B-V0.3 by selecting within UltraInteract-SFT (since it contains varying numbers of responses per-instruction). Closest-1 denotes the one closest to the base model's initial distribution. Random-1 is sampled from the entire enlarged dataset formed by both original and generated responses. We use (-) to denote **Random-1** underperforming **Closest-1**.

Model	Data	HE	LC	MBPP	MATH		GSMPlus		TheoremQA		Avg.
					CoT	PoT	CoT	PoT	CoT	PoT	
Mistral-7B	Self-Distill	46.3	13.3	49.6	17.3	18.5	43.3	33.2	16.8	17.4	28.4
	Original-UI	46.3	15.6	50.1	21.6	32.6	45.9	45.3	16.8	20.1	32.7
	Ours	52.4	15.6	53.4	28.9	34.6	50.5	52.8	17.8	20.6	36.3
Llama3.1-8B	Self-Distilled	47.6	6.7	51.7	22.9	12.7	47.2	35.3	18.8	21.5	29.4
	Original-UI	54.3	11.1	58.9	29.7	31.0	53.7	51.6	20.0	20.8	36.8
	Ours	57.3	19.4	63.8	34.8	39.2	56.6	56.1	22.5	23.9	41.5
Llama3.2-3B	Self-Distilled	32.3	5.6	41.9	8.8	7.0	12.1	12.1	5.9	10.5	15.1
	Original-UI	32.9	3.9	41.6	12.8	16.1	30.8	19.5	14.6	10.5	20.3
	Ours	42.6	13.3	44.6	16.4	17.6	34.9	20.6	15.1	11.4	24.1

Table 12: The detailed comparison across benchmarks for self-distillation discussed in Section 5.5

fine-tuning. These approaches have led to strong empirical results across various settings. However, many of them involve substantial computational costs, such as repeated gradient computations or auxiliary model training, which can limit their scalability. We discuss these approaches and the costs they incur in this section.

F.1 Notations

1. A training dataset $D = \{x_i\}_{i=1}^{N}$ of size N; the final language model to be trained on the selected data θ.
2. We denote **the average cost of one forward pass** of model θ on a training example as F_θ. As one backward pass is approximately the cost of two forward passes, **the average cost of one "gradient pass"** (i.e., one forward + one backward) is thus $3F_\theta$.
3. Another important source of computational cost in data selection comes from the training of additional models. We use $C(\theta, D, T)$ to denote the cost of training model θ on dataset D for T epochs (i.e., $N \cdot T$ examples are seen in total).
4. Therefore, we unify the computational cost of most data selection approaches into two parts:
   (a) **The training of additional models**. For example, gradient-based influence requires training an additional model on part of the training dataset for T epochs to obtain the checkpoints for gradient computation.
   (b) **The computation of per-sample features**. For example, for each training example, gradient-based influence requires computing its gradient for each saved checkpoint, which means T gradient passes are needed.
5. Note that some algorithms may have additional computational costs other than the two parts above, such as clustering or a greedy algorithm for the final data selection. Since the two parts above constitute the majority of computation for almost all the data selection approaches, **we omit the other cost and only focus on these two.**

F.2 TLDR: The Final Table

For GRAPE, we assume that in the training dataset D, various responses to the same instruction are already available, thus no additional cost is incurred in the *Response Collection* step of GRAPE. So the computational cost analysis of GRAPE under our framework is:

- **Additional Training**: 0, as GRAPE directly evaluates data using the base model.
- **Per-sample conditional probability**: NF_θ, as for a given target model θ, we only need to compute conditional probability for each response (example) once.

The table below shows that our method, GRAPE, achieves superior performance with minimal computational cost compared with other model-based data selection approaches.

	Additional Training	**Per-Sample Feature Computation**
GRAPE (ours)	0	NF_θ
Gradient-based influence (LESS)	$C(\theta_{\text{lora}}, D_{\text{warmup}}, T)$	$3T \cdot NF_\theta$
In-run gradient-based influence	$C(\theta, D, 1)$	0
Gradient matching	$C(\theta_{\text{lora}}, D_{\text{warmup}}, T)$	$3T \cdot NF_\theta$
Gradient norm	$m \cdot C(\theta, D, 1)$	$3m \cdot NF_\theta$
Embedding-based	0	NF_θ
Simple uncertainty indicators	0	NF_θ
Perplexity	$C(\theta_{\text{ref}}, D_{\text{ref}}, 1)$	$NF_{\theta_{\text{ref}}}$
Learnability	$C(\theta, D, 1)$	$2 \cdot NF_\theta$
Loss trajectory (S2L)	$C(\theta_{\text{ref}}, D, T)$	$T \cdot NF_{\theta_{\text{ref}}}$

Table 13: Computational cost comparison of data selection methods.

F.3 Gradient-based Methods

Gradients have long been an important source of information for training data selection, as they directly affect the whole optimization process of language models. Three kinds of model-based gradient-based data selection approaches have been proposed:

1. Gradient-based influence
2. Gradient matching
3. Gradient norm

F.3.1 Gradient-based Influence

Gradient-based influence computes the pairwise influence scores between each pair of training and validation examples. Training data with the highest influence are selected, as training on them leads to the theoretically largest decrease in model loss on validation data. LESS Xia et al. (2024) formulates the pairwise influence scores as the cosine similarity between the gradients of training and validation data, and computes these gradient features using the following two steps:

1. LoRA-train the final model on part of the whole training dataset, denoted as D_{warmup}, for T epochs, and save the T model checkpoints.
2. For each data point, compute its LoRA gradient with each of the T checkpoints, and later aggregate these T gradients together in the cosine similarity expression.

Therefore, the computational cost of gradient-based influence is:

- **Additional training**: $C(\theta_{\text{lora}}, D_{\text{warmup}}, T)$.
- **Per-sample gradient for each checkpoint**: $NT \cdot 3F_\theta = 3T \cdot NF_\theta$.

In order to reduce the cost incurred by per-sample gradient computation, recent work has developed *in-run gradient-based influence* that directly computes the dot product between gradients without

the need for separate gradient computations. However, this approach incorporates the dot product computations into the standard training process, which means in order to obtain pairwise influence scores for **the whole training set**, **a full training run** has to be done on all the training data. This incurs inefficiency when we do not actually need full dataset training. Moreover, the pairwise scores here only show the model's **"dynamic preference"**: scores computed at the t-th iteration only reflect the model's preference at this specific iteration. It is not theoretically guaranteed that these scores reflect the model's preference from the beginning of training. Thus, the cost of in-run gradient-based influence is:

- **Additional Training**: $C(\theta, D, 1)$.
- **Per-sample gradient**: 0.

F.3.2 Gradient Matching

Gradient matching also requires per-sample gradients, but utilizes their information in a different way. It performs clustering based on these gradient features to group similar data, and then applies an iterative greedy selection algorithm. **In order to scale to LLM-level gradient computation and clustering, TAGCOS Zhang et al. (2024b) completely follows the warmup training and gradient computation pipeline of LESS ?.** As the computational bottleneck here is still the gradient computation instead of clustering or iterative selection, Zhang et al. (2024b) also shares the same computational cost as Xia et al. (2024):

- **Additional training**: $C(\theta_{\text{lora}}, D_{\text{warmup}}, T)$.
- **Per-sample gradient for each checkpoint**: $NT \cdot 3F_\theta = 3T \cdot NF_\theta$.

F.3.3 Gradient Norm

The L_2-norms of gradient vectors can also serve as effective indicators for data selection. Paul et al. (2023) proposes GraNd, which obtains a utility score for each training point based on its gradient norm early in the training. More specifically, it starts from m different model weight initializations, trains each model on the whole dataset to obtain per-sample gradient norms, and finally averages the m gradient norms for each training point to obtain the final GraNd score. Therefore, the computational cost of GraNd is shown below:

- **Additional training**: $m \cdot C(\theta, D, 1)$.
- **Per-sample gradient for each weight initialization**: $Nm \cdot 3F_\theta = 3m \cdot NF_\theta$.

F.4 Embedding-based Methods

Embedding-based methods project the whole training set into an embedding space to quantify the information of each data point and their interactions. For model-based embedding-based selection methods, the embeddings are usually computed by the final model θ to align with its preference.

Under a supervised data selection setup where validation data representing target task distributions are available, **Representation-based Data Selection** (RDS; Rubin et al. (2022); Hanawa et al. (2021)) computes the embedding similarity between training and validation data, and selects training points that are most similar to the target distribution in the embedding space.

For an unsupervised setup where only the embeddings of training data are accessible, **geometry-based coreset sampling** methods are widely used Qin et al. (2024). Grounded on the intuition that close samples in the embedding space often share similar properties, a **diverse** subset can be obtained by controlling the minimum distance between any two selected data points. Among them, using K-center greedy sampling to select embedding-based **facility locations** has been proven especially effective for instruction fine-tuning of LLMs Bhatt et al. (2024a).

These embedding-based approaches share similar computational costs: they do not need any additional model training and can directly extract useful per-sample embeddings using the last-layer hidden states of the pretrained final model θ. Thus, their computational cost is shown below:

- **Additional training**: 0.
- **Per-sample embedding computation**: NF_θ.

F.5 LogProb-based Methods

LogProb-based methods also directly utilize the target LLM to evaluate the utility of each training data point.

4.1 Simple Uncertainty-based Indicators

Some simple model-based indicators inspired by the notion of uncertainty have been shown effective for a long time and recently extended to data selection for LLM instruction tuning Marion et al. (2023b); Bhatt et al. (2024b). Bhatt et al. (2024b) demonstrates the effectiveness of various indicators including **mean entropy, least confidence, mean margin**, etc. These simple indicators do not require additional training and can also be directly obtained with the pretrained final model θ. Their computational cost is shown below:

- **Additional training**: 0.
- **Per-sample per-token logits computation**: NF_θ.

F.5.1 Perplexity (PPL)

PPL is also a long-standing data selector and has been shown effective for LLM-scale data selection. Typically, a split of the training dataset, D_{ref}, is needed to train θ_{ref}, a reference model that will be used to compute PPL for the whole training set.

A common approach is to use the final model θ as the reference model θ_{ref} to ensure the alignment in PPL patterns Marion et al. (2023a), but prior work Ankner et al. (2024) also shows that a reference model much smaller than the final model can also be an effective PPL-based data selector. The computational cost for PPL-based selection is shown below:

- **Additional training**: $C(\theta_{\text{ref}}, D_{\text{ref}}, 1)$.
- **Per-sample PPL computation**: $NF_{\theta_{\text{ref}}}$.

F.5.2 Learnability

In addition, **learnability (Mindermann et al., 2022; Zhou et al., 2024a;b)** is a more effective metric than pure uncertainty or PPL, as it excludes uncertain but unlearnable points (e.g., noisy or less task-relevant) by considering the decrease in per-sample loss before and after the model is fully trained. More specifically, it trains the final model θ on the full training dataset to obtain a strong reference model θ_{ref}, and then computes the difference of loss on each training example between θ and θ_{ref}. In this way, it requires two forward passes for per-sample computation:

- **Additional training**: $C(\theta, D, 1)$.
- **Per-sample learnability computation**: $2 \cdot NF_\theta$.

F.5.3 Loss Trajectory

Moreover, logprob-based methods can also obtain finer-grained information from the **training dynamics** of LLMs. S2L Yang et al. (2024b) obtains a feature vector for **each** training point by collecting their **training loss trajectories** over T-epoch training on a small reference model θ_{ref}, and then applies K-means clustering to equally sample data points from each trajectory cluster. Prior work shows its superiority over other logprob-based indicators, but it also comes with significant computational cost:

- **Additional training**: $C(\theta_{\text{ref}}, D, T)$. Here the choice of θ_{ref} is especially important, as prior work Xia et al. (2023) shows that reference models that come from the same model family as the final model tend to have similar loss trajectories of training data, so they can preserve more fidelity in their loss trajectory patterns.
- **Per-sample loss trajectory computation**: $T \cdot NF_{\theta_{\text{ref}}}$. Note that T here is typically much larger than that in gradient-based influence computation, so the computational cost of this gradient-free approach can be even higher than gradient-based methods.

G Multi-Round GRAPE

To evaluate whether GRAPE continues to provide benefits after an initial round of fine-tuning, we conducted a second-round experiment using LLAMA3.1-8B. In the first round, we selected the best responses per instruction using GRAPE and fine-tuned the model accordingly. Then, we re-applied GRAPE on the newly fine-tuned model to select a fresh set of responses and conducted another round of fine-tuning. This second iteration led to further performance improvements across multiple benchmarks, including AlpacaEval, WizardEval (both original and LeetCode), and real-world tasks such as MATH, MMLU, and BBH. Notably, GRAPE Round 2 improved average benchmark scores from 35.9% to 37.3%, demonstrating that the method remains effective and even compounding when iteratively applied.

Method	AE	WR	WR (LC)	MATH	MMLU	BBH
GRAPE Round 1	15.2	14.8	19.4	32.1	64.5	69.6
GRAPE Round 2	**17.6**	**19.6**	18.9	**33.2**	64.5	**70.0**

Table 14: Performance of LLAMA3.1-8B after two rounds of GRAPE fine-tuning.

H Details Of Correlation Analysis

To analyze how well different models align with the training distribution, we conducted a perplexity-based study on the Tulu-v3 training set. Specifically, we randomly sampled 1,000 instances from the training data and used a wide array of generator models to produce responses for each instruction. For each response, we computed its perplexity under each base model and ranked the responses by perplexity, with lower perplexity indicating better matching of distribution. We then identified the top-1 response per instance for each base model and visualized the overall rankings using a heatmap (Figure 4).