Appendix

In this appendix section, we provide additional details that could not be included in the main paper due to space constraints:

- Additional details on the **Baselines** (extending Sec.4.1 and Fig. 1 of the main manuscript).
- **Implementation** details and **hyperparameter** settings (extending Sec.4.1).
- Further **Ablation** studies for **Hyperparameters** (extending Sec.4.3)
- Analysis on **Why PROGRESS Works**: Representativeness of Selected Subsets and Order of Skills
- Comparison with concurrent work ICONS (extending Sec.4.2 and Sec 2).
- **Intuition and Justification** for using relative improvement (extending Sec.3).
- In-depth analysis **Wall-clock Time, Time Breakdown Analysis, Annotation Time Analysis,** and **Scaling Performance** (extending Sec.4.3).
- **Word Cloud Visualization** for our multimodal clustering approach (extending Sec.3.1).
- Visualization of the **Diversity** of samples selected by our method (extending Sec. 4.3).
- Additional details on the setup and algorithms used for our analysis (extending Sec.4.4).
- Limitations of our method and LLM usage disclosure statement

6. Details of Experimental Setups

6.1. Baselines

We follow the standard experimental settings and implementation protocols for all baselines as established in recent prior work (see COINCIDE [19]), using official code. For completeness, clarity and reproducibility, we additionally provide detailed descriptions of each baseline here.

COINCIDE [19] is a strong coreset selection method that leverages concept-based clustering and mutual transferability between clusters to guide sample selection. COINCIDE performs coreset selection only once, before training, by clustering internal activations of a separately trained (additional) reference VLM (e.g., TinyLLaVA[50]). COINCIDE relies on static selection strategy that does not adapt to the model's learning progress, requires an additional pretrained VLM, ground-truth annotations for full dataset to extract activation maps, and manual intervention to select appropriate activation layers—making the method resource-intensive and difficult to scale.

CLIP-Score [13] It ranks image-instruction pairs based on visual-textual similarity computed by the CLIP model [37], selecting top-scoring samples for training. While this approach assumes that higher similarity indicates greater informativeness, it relies on static, precomputed metrics that do not adapt to the model's learning progress. Prior work [19] has shown that such metrics often fail to capture important data modes, resulting in reduced diversity and suboptimal generalization—limitations that PROGRESS overcomes through dynamic, progress-driven selection.

EL2N [36] ranks training samples based on the expected L2-norm of prediction error: $\mathbb{E}[\|p(x) - y\|_2]$ where $p(x)$ is the token distribution predicted by a reference VLM, x is the input, and y is the ground-truth label. This score reflects how confidently and accurately the reference model predicts each sample. However, it requires access to a fully trained additional VLM and ground-truth labels for the entire dataset, making it resource-intensive and static.

Perplexity [31] measures the uncertainty in the model's predictions and is defined as $\exp(-\mathbb{E}[\log p(x)])$, where $p(x)$ denotes the likelihood assigned to input x by a additional reference VLM model. Samples from the middle of the perplexity distribution are selected, following prior work [19]. However, it requires access to a fully trained additional VLM and, like other static metrics, often fails to capture important data modes—potentially limiting diversity and downstream generalization.

SemDeDup [1] aims to reduce redundancy by removing semantically duplicated samples. It clusters the output embeddings of the final token from a reference model's last layer and retains a diverse subset by eliminating near-duplicates and reducing redundancy. This method also requires an additional reference VLM to extract the final token features and ground-truth labels for the entire dataset.

D2-Pruning [29] constructs a graph over training data where nodes encode sample difficulty and edges capture pairwise similarity. It selects a diverse coreset by pruning this graph while preserving representative and challenging samples. Difficulty is measured using the AUM score, defined as $p_y(x) - \max_{i \neq y} p_i(x)$, where $p_y(x)$ is the model's confidence for the ground-truth label y. Similarity is computed using the L2 distance between average final-layer token embeddings from a additional reference VLM. This method requires access to an additional reference VLM for embedding extraction and scoring and ground-truth labels for the entire dataset.

Self-Sup [40] clusters data using averaged output embeddings from the final-layer tokens of a reference model. It assigns scores based on distance to cluster centroids, selecting samples that are closest—assumed to be the most prototypical representatives of the data distribution. This method also requires access to an additional reference VLM for embedding extraction.

Self-Filter [6] is a recent coreset selection method originally proposed for the LLaVA-158k dataset (containing three vision-language tasks). It jointly fine-tunes a scoring network alongside the target VLM on the *entire labeled*

dataset, using this as learned reference model to score and filter training samples—hence it requires an additional reference model trained on full data with full annotations. Following previous work [19], we adopt the stronger variant that also incorporates CLIP scores and features.

Random. We additionally report results for *Random*, which finetunes the model using a coreset selected via random sampling. Despite its simplicity, Random serves as a strong and competitive baseline—prior work [19]has shown that random sampling often preserves sample diversity and can outperform more complex selection methods in certain settings.

Note: We use standard setup for the baseline implementations as described in prior work (see COINCIDE Appendix [19]). For **COINCIDE**, **EL2N**, **SemDeDup**, **D2-Pruning**, and **Self-Sup**, we use image, question, and ground-truth answer for full dataset as inputs along with additional reference VLM (i.e., TinyLLaVA) to extract features following prior work [19]. **Self-Filter** requires full dataset to finetune additional reference network—the scorenet. As a result, these baselines require an additional reference vision-language model or full dataset annotations (100%) or both.

Active Learning Nature of the Compared Baselines. AL selects which unlabeled samples to label next using acquisition function a(x) [44]. Higher a(x) (e.g., uncertainty, error, diversity, representativeness) implies higher expected utility of sample. EL2N [36] and Perplexity [31] are uncertainty-based AL methods with error/uncertanity as acquisition functions. Self-Filter [6] uses deep network as acquisition function. It jointly trains a deep score network with the target model to later rank and select samples. COINCIDE [19] is hybrid sampling based AL which uses a hybrid of skill transferability + diversity for acquisition of samples to select a representative coreset. Self-Sup [40] aligns with representativeness-based AL coreset selection, aiming to pick prototypical samples that best cover the data distribution.

Curriculum Learning Nature of Compared Baselines. Curriculum Learning (CL) methods can be partitioned into two types: (Type 1) External model decides difficulty and select samples; (Type 2) Models own feedback decides difficulty and select samples.

Self-Filter [6] is an instantiation of Type 1 CL methods where an external model (i.e deep score network) is jointly trained with the target model to quantify the learning difficulty of sample and select hard-examples for training. The auxiliary score network guides selection of samples expected to have significant value for model training based on difficulty.

We also compare with Type 2 curriculum policies in Table 3: 1) Easiest Selection (Table 3 row 3) - model self-evaluates and chooses easy-samples with highest absolute performance, 2) Medium difficulty Selection (Table 3, row 4)- model self-evaluates and chooses samples with medium absolute performance, 3) Hardest Selection (Table 3, row 5) - model self-evaluates and chooses hard-samples with lowest absolute performance 4) Relative-Improvement based selection (ours, Table 3, row 6-7).
Overall, our PROGRESS method (relative improvement based policy) performs much better than other CL baselines.

Table 5. Hyperparameter configurations. K represents the number of clusters.

Method	LLaVA-1.5	Vision-Flan
CLIP-Score	high score selected	high score selected
EL2N	medium score selected	medium score selected
Perplexity	medium score selected	medium score selected
SemDeDup	$\boldsymbol{K}$: 10,000	$\boldsymbol{K}$: 5,000
D2-Pruning	$\boldsymbol{k}$: 5, γ_r : 0.4, γ_f : 1.0	$\boldsymbol{k}$: 5, γ_r : 0.4, γ_f : 1.0
Self-Sup	$\boldsymbol{K}$: 10,000	$\boldsymbol{K}$: 5,000
Self-Filter	$\boldsymbol{k}$: 10, γ : 1	$\boldsymbol{k}$: 10, γ : 1
COINCIDE	$\boldsymbol{K}$: 10,000, τ : 0.1	$\boldsymbol{K}$: 5,000, τ : 0.1
PROGRESS		
Warmup Stage		
Number of Clusters	$\boldsymbol{K}$: 10,000	$\boldsymbol{K}$: 5,000
Warmup Ratio (w.r.t full data)	9%	8.4%
Prioritized Concept Learning		
Number of Clusters	$\boldsymbol{K}$: 1,000	$\boldsymbol{K}$: 200
Temperature of Softmax	τ : 1.0	τ : 1.0
BatchSize	128	128
Selection Gap	$\boldsymbol{\gamma} * BatchSize$: 7,500	$\boldsymbol{\gamma} * BatchSize$: 3,500
Random Exploration	$\delta\%$: 10 %	$\delta\%$: 10 %

6.2. Implementation Details

In this section, we provide elaborate details on implementation of our approach in continuation to the brief details we provide in Section 4.1 of main manuscript.

We first partition the unlabeled data pool $\mathbb{U}$ into K skill clusters using spherical k-means, following the fully *unsupervised* concept categorization procedure described in Section 3.1. Training begins with a brief warmup phase (see details in Appendix 6.3), which equips the model with basic instruction-following capability and ensures that skill-level performance estimates are reliable in the beginning of training.

Subsequently, we apply our Prioritized Concept Learning (PCL) strategy (see Section 3.2) to estimate the expected performance improvement for each skill cluster between iteration t and $t - \gamma$ (as defined in Eqn 1), using either accuracy or loss as the tracking metric (see Table 1, row 10 and 11). For the accuracy-based variant, we compute cluster-wise accuracy using an LLM judge as metric that compares the VLM output to ground-truth answers—though this is not required for our loss-based variant. Samples are then selected using a temperature-controlled softmax over the

Prompt for Accuracy Estimation

Given an input question and two answers: a candidate answer and a reference answer, determine if the candidate answer is correct or incorrect.

Rules:

- The candidate answer is correct if it is semantically equivalent to the reference answer, even if they are phrased differently.
- The candidate answer should be marked as incorrect if it:
  - Contains factual errors compared to the reference answer
  - Only partially answers the question
  - Includes hedging language (e.g., "probably", "likely", "I think", etc.)
  - Answers a different question than what was asked
- Give a reason for your prediction.

Output Format:

- Answer - correct or incorrect
- Reason -

improvement scores (see Eqn 2). This selection process is repeated every γ iterations, and in each round, we sample a total of $\gamma * BatchSize$ examples for annotation and training. We refer to this $\gamma * BatchSize$ as selection gap from here on.

Hyperparameters for Baselines and PROGRESS . To ensure fair comparison, we use the same hyperparameters as COINCIDE [19] for all baselines. The hyperparameters for both the baselines and PROGRESS are summarized in Table 5. For model training, we apply LoRA [14] to LLaVA-v1.5 and follow the official fine-tuning settings provided in the LLaVA-1.5 release. For Qwen2-VL, we perform full fine-tuning using the official hyperparameters specified by Qwen2-VL. For Qwen2.5-32B-Instruct, we apply LoRA and follow the official fine-tuning settings provided in LLaMA-Factory [49]. For accuracy estimation, we use LLMs such as InternLM2-Chat-20B [4] as the judge. We provide the question, ground-truth answer, and predicted answer (without the image) as input and ask the LLM to decide whether the prediction is correct. The full prompt is shown below. Ablation studies on all hyperparameters are provided in Section 4.3 and Appendix 7.

6.3. Warmup Phase

Warmup strategy details. Following prior work [45, 46], we begin with a brief warmup phase using a small subset—9% of the total pool size (we show in Fig. 10(a) that PROGRESS is robust to even smaller warmup ratios, remaining close to full-data performance even when the warmup percentage is further reduced.) — to equip the model with basic instruction-following capability and to obtain credible skill-level performance estimates at the start of training, when the model is still untrained. These initial estimates are essential for tracking relative learning progress across skills in subsequent phases. To make the warmup set effective, it should broadly cover diverse skill clusters.

To this end, we use a simple cluster-based sampler that prioritizes clusters which are both diverse and likely to generalize well. Concretely, we sample from each unsupervised concept cluster with probability $P_i \propto \exp(S_i/\tau D_i)$ where S_i and D_i denote the cluster's transferability and density, respectively, and τ is temperature following [19]. Importantly, unlike [19], our clusters and scores are computed directly from concatenated self-supervised DINO and BERT features (Sec. 3.1), and do *not* require any additional auxiliary reference VLM (i.e TinyLLaVA) features or ground-truth answers.

Table 4 (main manuscript) shows that the warmup phase alone provides only modest performance (94.6% Rel.), while PROGRESS yields large gains over this baseline (+4% relative, rows 2 vs. 1). This demonstrates that warmup merely initializes reasonable per-cluster estimates, whereas the major improvements stem from PROGRESS 's progress-driven sample selection. Moreover, row 3 shows that PROGRESS remains robust to different warmup strategies, achieving near–full-data performance even with a weak (random sampling) warmup.

6.4. Comparison with ICONS

ICONS [45] is a concurrent unpublished work that differs significantly from our approach. It requires **(1)** high memory and compute resources—reportedly over 100 GPU hours—to compute and store gradient-based influence scores for selection, and **(2)** access to explicit knowledge of the target task or its distribution in the form of labeled samples from validation set of target benchmarks. This assumption is impractical in general-purpose VLM training, where target tasks may be unknown at training time and usage of such high-compute refutes the goal of efficient learning. As such, ICONS is **not directly comparable** and falls outside the scope of our setting, which avoids both gradient-based selection and downstream task knowledge from target benchmarks prior to training the VLM model.

Nevertheless, we strive to compare with them in good faith by reproducing ICONS using their official codebase for fair comparison. Although ICONS has released its codebase, the validation data (for each target benchmark) it uses to simulate target task knowledge is not publicly available. The paper reports the number of validation samples used per benchmark (see Table 6), however the specific validation

Table 6. Statistics of ICONS target validation sets.

Dataset	MME	POPE	SQA-I	MMB-en	MMB-cn	VQAv2	GQA	VizWiz	TextVQA	LLaVA-W
$\lvert\mathcal{D}_{\text{val}}\rvert$	986	500	424	1,164	1,164	1,000	398	8,000	84	84
$\lvert\mathcal{D}_{\text{test}}\rvert$	2,374	8,910	4,241	1,784	1,784	36,807	12,578	8,000	5,000	84

Table 7. Comparison between PROGRESS and ICONS. Repro. means reproductions of ICONS.

Method	VQAv2	GQA	VizWiz	SQA-I	TextVQA	POPE	MME	MMBench en	MMBench cn	LLaVA-Wild	SEED	AI2D	ChartQA	CMMMU	Rel. (%)
0 Full-Finetune	79.1	63.0	47.8	68.4	58.2	86.4	1476.9	66.1	58.9	67.9	67.0	56.4	16.4	22.1	100
1 ICONS	76.3	60.7	50.1	70.8	55.6	87.5	1485.7	63.1	55.8	66.1	-	-	-	-	-
2 ICONS (Repro.)	75.0	57.7	45.9	63.7	55.1	86.0	1434.0	47.1	37.3	68.4	57.3	45.3	17.2	24.3	91.6
PROGRESS															
3 Loss as Obj.	75.7	58.6	49.6	70.1	55.1	86.3	1498.4	62.5	55.5	65.5	63.4	53.3	17.3	23.7	98.4
4 Accuracy as Obj.	75.2	58.8	53.4	69.9	55.1	85.9	1483.2	61.1	54.4	65.5	63.0	52.8	17.3	24.6	98.8

samples remain unspecified and are not publicly released. To approximate their setup, we randomly select an equal number of samples from the publicly available validation sets of target benchmarks and reproduce their performance.

Table 7 presents results across three settings: Row 1 shows the original ICONS results as reported; Row 2 presents our reproduction using their codebase and randomly selected validation samples; Rows 3 and 4 report results for PROGRESS . We observe that PROGRESS outperforms our ICONS reproduction in relative performance even though our method does not rely on compute-intensive gradient-based selection and does not assume any knowledge from target benchmarks, reinforcing the practicality and effectiveness of our method under realistic constraints.

6.5. Intuition, justification, and grounding for using relative improvement and softmax-based sampling

Empirical Validation. We show ablation results and discussion in Table 3 for different selection policy including Our Relative improvement strategy (Table 3, row 6,7) vs Easy, Medium and Hard selection based on hard-thresholding on absolute performance. Overall, our relative improvement policy performs much better than other variants

Intuition, justification for using relative improvement. Inspired by prior curriculum literature [3, 34, 38] we use relative improvement over absolute gains because it scales progress by a skill's baseline performance i.e normalizes across tasks of different difficulty and scale: tasks that are too easy (high baseline) or too hard (low baseline) tend to have small relative improvements, whereas tasks at a "moderate" difficulty yield the largest relative improvements and therefore get prioritized. This aligns with the zone-of-proximal-development [17, 42], problems that are neither too easy nor too hard produce the largest advantage, focusing on learning where progress is most productive.

In contrast, using absolute improvement would bias selection towards tasks with high raw score over-favoring raw score jumps and fail to account for the fact that a one-point increase on a hard skill is more meaningful than the same gain on an easy one, fixed thresholds likewise cannot adapt to varying skill difficulties. Overall, relative improvement avoids over-favoring weak skills with noisy raw swings and instead emphasizes areas where progress per unit of remaining error is highest, providing a stable, self-paced curriculum [34, 38].

Intuition for Softmax Sampling. We map Δ_k^t to sampling weights using a temperature-controlled softmax, $p_k \propto \exp(\Delta_k^t/\tau)$, which provides a smooth, entropy-regularized trade-off between exploiting high-utility skills and exploring others. A small floor probability safeguards rare skills. This results in a stable, noise-robust scheduler that outperforms absolute gains and hard thresholds. The temperature parameter τ explicitly balances informativeness and diversity which is crucial for effective learning and avoid mode collapse (see Fig 5).

7. Further Ablation Studies and Analysis

7.1. Scalability and Generalization to Larger VLMs

Here we additionally present results for Qwen2.5-VL-32B-Instruct in Table 8, which we instruction-tuned using our method on the LLaVA-665K dataset, using a 20% sampling ratio following standard protocol. PROGRESS (trained with just 20% data) performs better than full- data fine-tuning, achieving 100.2% relative performance compared to full-data fine-tuning, demonstrating our method's scalability potential and its generalization to new large scale architectures.

7.2. Scalability and Transferability to Larger Datasets

To evaluate scalability beyond the LLaVA-665k corpus, we report results on an expanded $\sim$1M-sample dataset (Table 9) comprising the LLaVA-Next-Dataset ($\sim$800k) and Vision FLAN ($\sim$200k). On this scale, PROGRESS recovers 99.5% of the full-data performance using only 20% of the

Table 8. **Scalability and Fine-tuning.** We report results for Qwen2.5-VL-32B on the LLaVA-665K dataset using 20% sampling ratio.

Method	VQAv2	GQA	VizWiz	SQA-I	TextVQA	POPE	MME	MMBench en	MMBench cn	LLaVA-Wild	SEED	Rel. (%)
Full-Finetune	83.8	64.5	58.8	93.1	82.3	88.5	1678.5	85.3	85.3	78.9	75.8	100.0
PROGRESS	83.7	63.7	59.0	93.1	82.0	88.1	1684.8	86.7	87.1	77.8	76.4	**100.2**

Table 9. Performance for QwenVL-2-7B on 1M sample dataset.

Method	VQAv2	GQA	VizWiz	SQA-I	TextVQA	POPE	MMB-en	MMB-cn	AI2D	SEED	Rel. (%)
Full-Finetune (100% data)	76.1	61.4	45.6	80.5	55.5	85.0	78.9	77.9	79.9	67.2	100.0
COINCIDE (20% data)	75.5	59.8	40.2	78.7	58.4	83.7	76.7	73.5	78.0	67.9	97.7
PROGRESS (20% data)	74.9	59.3	46.6	79.8	56.1	85.0	78.4	76.4	78.5	68.3	**99.5**
PROGRESS-ReAssign (20% data)	75.1	59.6	46.6	80.4	53.8	83.2	76.9	76.8	78.7	68.6	98.9

training data. Furthermore, to investigate the transferability of the clusters across different data distributions, we introduce PROGRESS-REASSIGN. This variant trains on the ∼1M dataset by reusing the clusters previously derived from LLaVA-665k, thereby enabling skill assignment without the need for re-clustering. PROGRESS achieves 98.9% at 20% data density, significantly outperforming the COINCIDE and demonstrating that our learned skill space generalizes effectively to new, larger datasets.

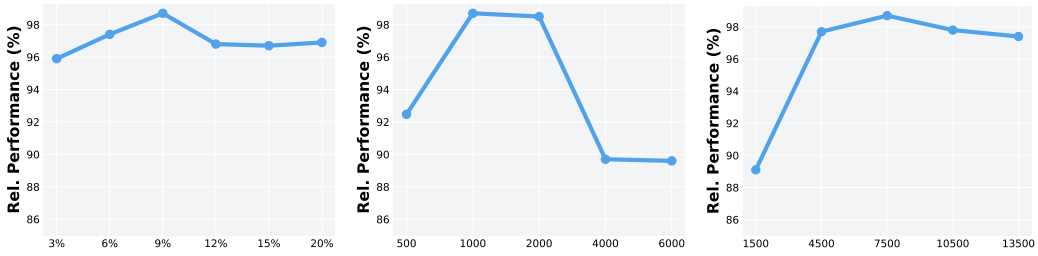
7.3. Ablation Studies

Ablations on Hyperparameter. We conduct further ablations studies in Fig. 10 on effect of hyperparameters. All experiments use LLaVA-v1.5-7B on the LLaVA-665K dataset with 20% sampling ratio and accuracy as the objective. Fig. 10(a) shows the effect of different warm-up ratios relative to the total training data pool size. Our results show that a 9% warm-up ratio achieves the best performance, as it strikes a balance between preparing the model adequately and leaving enough room for our iterative Prioritized concept learning strategy to select informative samples. The 20% warm-up ratio (i.e using only warmup selected samples to entirely train the model eleminating our Prioritized concept learning strategy completely), results in significantly reduced performance in overall relative score. Next, Fig. 10(b) shows the effect of varying the number of clusters K used for our concept categorization module (Section 3.1). Using too few skill-clusters reduces skill diversity and leads to lower purity (in terms of skill types) within given cluster, while too many clusters result in redundant clusters of the same skill category and insufficient samples per cluster to yield credible accuracy estimates. We find that using approximately 1,000-2000 clusters strikes the best balance and yields optimal performance. Finally, Fig. 10(c) shows the influence of the selection gap i.e $\gamma * BatchSize$ (see definition in Appendix 6.2). We find that the model is particularly sensitive to small gaps; for instance, a gap size of 1,500 leads to a rapid performance decrease. Smaller gaps cause the model to switch too soon, not allowing it to learn the selected concepts sufficiently.

Why PROGRESS Works: Representativeness of Selected Subsets and Order of Skills

Representativeness of the Selected Subset. We evaluate how well the subsets selected by PROGRESS and COINCIDE reflect the full dataset using PCA-based analyses. We first concatenate the DINO and BERT features and fit a single Principal Component Analysis (PCA) model on the entire dataset, reducing the dimensionality to 50 components. This global PCA transform is then applied consistently to the full dataset and to the samples selected by both methods. In this 50-dimensional PCA space, we compute two measures of representativeness:

- **Mean Nearest-Neighbor (NN) Distance.** To estimate how densely each subset covers the data manifold, we build a FAISS `IndexFlatL2` index containing all PCA-transformed points from the full dataset. For every point in a subset, we query this index to find its closest neighbor *within the full dataset* and measure that Euclidean distance. Averaging these distances yields a single value that reflects how close the subset stays to the dense, frequently occurring regions of the distribution. Lower values indicate that the subset lies in typical, well-populated regions rather than isolated or sparsely sampled areas. PROGRESS achieves a mean NN distance of $|0.083|$, substantially lower than COINCIDE's $|0.198|$, indicating that PROGRESS selects samples closer to the core of the dataset.
  While mean NN distance is intuitive and meaningful, it can be minimized by degenerate subsets that concentrate in a few high-density regions and fail to preserve the global structure of the distribution. For this reason, we also report a complementary *Variance Coverage* metric.
- **Variance Coverage.** PCA orders directions in the feature space by how much variation they capture: the leading components encode the dominant structural patterns of the dataset, while later components reflect finer-grained variations. To assess how well each subset preserves this structure, we compare its variance along every principal component to the corresponding variance in the full dataset. For each component, we compute ratio $= \frac{\text{variance (subset)}}{\text{variance (full)}}$, which measures how much of the dataset's variation along that axis the subset retains. We summarize this comparison in two ways. First, we compute a *Weighted Variance Retained* score by weighting each variance ratio by the component's explained variance in the full dataset and summing across all 50 components.



(a) Warmup Ratio (b) Cluster Number K (c) Selection Gap γ*$BatchSize$

Figure 10. **Ablation Studies.** (a) Effect of the warm-up ratio. (b) Effect of the number of clusters. (c) Effect of the selection gap.

This yields a single measure of how much of the dataset's overall variability the subset preserves. PROGRESS retains more variance than COINCIDE (38.7% vs. 34.2%). Second, we directly examine the variance ratios for the top principal components, which capture the major structural axes of the dataset. High coverage on these components indicates that a subset spans the most important modes of variation rather than collapsing onto a narrower region. PROGRESS covers a substantially larger fraction of these leading components compared with COINCIDE (77–88% vs. 48–72%).

These results show that PROGRESS spans the overall data distribution more effectively, while COINCIDE selects a more concentrated and less representative subset.

How important is the order of skill acquisition? Unlike prior methods that focus solely on selecting which samples to use [19, 45], PROGRESS also controls when to introduce them during training (Section 3.2)—effectively guiding both skill selection and the order of acquisition. To assess the importance of learning order, we ablate this component by randomly shuffling the data selected by PROGRESS and training the model without respecting the intended sequence. Even with the same data, training in a random order leads to a noticeable performance drop—from 98.8% to 94.6%—highlighting that when to introduce concepts is just as important as what to learn.

7.4. Wall-clock Time Comparison

We compare average relative performance (Rel.) against wall-clock time on the Vision-Flan dataset in Fig. 11. PROGRESS (Accuracy as Objective) achieves relative performances of 89.4%, 93.3%, 99.0%, 100.2% and 101.8% with wall-clock times of 0.69, 0.89, 1.21, 1.65 and 2.52 hours. Full fine-tuning on entire dataset takes about 2.56 hours. Remarkably, our method exceeds 100% relative performance (i.e surpasses vanilla full data fine-tuning) in just 1.65 hours—including both data selection and model training time—needing only 64% of the time required for full dataset fine-tuning.

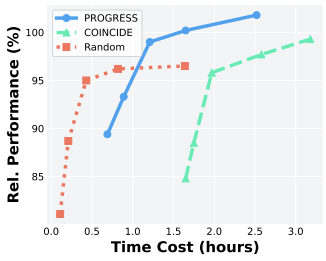


Figure 11. **Wall-clock time** comparison on Vision-Flan dataset.

7.5. Stage-Wise Time Breakdown Analysis

Table 10 presents a detailed time breakdown comparing our PROGRESS method with the closest competitor COINCIDE. We use same GPU compute for both COINCIDE and our method for fair comparison. COINCIDE requires over 8 hours of total computation time (plus additional unknown manual inspection time) and needs an additional pre-trained TinyLLaVA VLM to extract and store features from multiple MSA layers (4th, 8th, 12th, 16th, 20th). For each sample, they need 4096×5 = 20,480 dimensional features, which demand substantial memory resources and significantly increases clustering time. In contrast, our PROGRESS method completes in only 5.67 hours without requiring any additional VLM models. By using lightweight uni-modal self-supervised feature extractors—combining DINO-v2 (1,024 dimensions) and Sentence-BERT (384 dimensions)—we achieve efficient clustering with only 1,408-dimensional features per sample. This represents a 14.5× reduction in feature dimensionality compared to COINCIDE, while still achieving better relative performance w.r.t COINCIDE using only 20% of the training data.

7.6. Annotation Time Analysis: Significance of Annotation Cost as the Primary Bottleneck

In this section, we estimate the overall annotation cost and show that it is the dominant bottleneck in scaling VLM training to larger datasets—a cost that our method drastically reduces by 80% (as it requirs only 20% data for training).

The LLaVA-665K dataset has 0.67 million samples comprising Human-curated data (OKVQA, A-OKVQA, OCRVQA, TextCaps) and Synthetic QA which still need human-annotated bounding box & object names (from

Table 10. **Time breakdown comparison:** COINCIDE vs PROGRESS vs Full Data Training. We use official code to produce this time analysis

Stage	Time	Note
Full Data Training	**540 min (9h)**	Training on 100% data on 4×A100
COINCIDE		
Manual Inspection	Unknown	Manual MSA layer selection to extract features, needs ground-truth annotations
Feature Extraction	280 mins	TinyLLaVA feature extraction on 2×A100
Clustering	50 mins	With faiss-gpu KNN on single A100. COINCIDE requires large memory to store 20,480 dim. features from multiple MSA layers.
Training	150 mins	Training on 20% data on 4×A100
Total	**480 min (8h)**	
PROGRESS		
Feature Extraction	30 mins	Sentence-BERT (all-MiniLM-L6-v2) + DINO-v2-base on single A100
Clustering	30 mins	With faiss-gpu KNN on single A100
Annotation Decision	130 mins	Model self-evaluates and decides what to annotate next. 13 mins × 10 rounds
Training	150 mins	Training on 20% data on 4×A100
Total	**340 min (5.67h)**	

COCO etc, see details in [23, 25]) which are then provided to an LLM to generate Q/A.

Prior studies [8] estimate it typically takes 10.3 sec for a human (Mturk worker) to annotated 1 sample (which consists Image & 4-5 Q/A pairs). So, to annotate 0.67 million samples in LLaVA dataset it will take 1902 hours (0.67 M * 10.3 / 3600).

Estimated annotation time:

• Full-data training (100% of 0.67M samples): $\sim$1902 hr

• PROGRESS (20% of 0.67M samples): $\sim$380 hr

Overall time (annotation + training):

• Full-data finetuning: 1902 hr (annotation) + 9 hr (training)

• PROGRESS : 380 hr (annotation) + 5.7 hr (training)

Conclusion. Annotation is by far the dominant cost and major bottleneck in VLM training, especially when we scale to even larger datasets. By reducing annotation to only 20%, PROGRESS drastically cuts the primary bottleneck and offers a scalable path for training on even larger datasets.

7.7. Scaling data improves PROGRESS even more than vanilla Full-Finetuning

We point to results shown in Table 11 and Fig 7 where scaling data size to higher percentages 32%, 64%, our method outperforms full-finetuning (which uses 100% data) by larger and larger margin (see Rel. Score) as data % is scaled up.**Reason for better scaling performance**- Our method removes redundancy & focuses on most informative samples that the model should learn next - naturally shifts attention toward skills that show strong learning potential, instead of spending excessive effort on skills it already performs well on or are too hard to learn at this instant of time.

7.8. Details for Analysis in main manuscript- How does the benchmark difficulty and data frequency impact performance?

In this section, we elaborate on details regarding the analysis in Section 4.4, where we analyze the impact of benchmark difficulty and data frequency.

Table 11. Scaling performance of PROGRESS .

Data Used	Rel. Score (%)
Full Finetune	
100%	100
PROGRESS	
4.2%	89.4
8.3%	93.3
16.7%	99.0
32.0%	100.2
64.5%	101.8

7.8.1. Details for Benchmark Difficulty Analysis

Here, we provide details regarding the analysis shown in Fig. 8 (a) of main manuscript.

Quantifying Benchmark Difficulty. Prior work has shown that human intuition about task difficulty may not align with a model's difficulty as defined in its feature or hypothesis space [38]. Therefore, we use the model's own performance as a proxy for determining benchmark difficulty. Specifically, we use the performance of full-dataset fine-tuned LLaVA-v1.5-7B (i.e., Row 0 in Table 1) as reference to determine difficulty of benchmark—benchmarks with higher performance are considered easier. We define **benchmark difficulty** for a given benchmark as $(100 -$ Performance of full fine-tuned LLaVA-1.5 on Benchmark$)/100$. This gives us a difficulty measure for each benchmark nor-

Algorithm 1 Sample Rarity Estimation via Gaussian Modeling

Require: Training dataset $\mathbb{U} = \{x_1, \ldots, x_N\}$ where $x_i = (I_i, Q_i)$; set of benchmarks $\mathcal{B} = \{B_1, \ldots, B_M\}$ with M different benchmarks.

Feature Extraction: Extract DINO features from images and BERT features from questions; concatenate to form joint feature vectors, for all training and benchmark samples.
We denote DINO-BERT embedding of x_i as $\phi(x_i)$

Fit Gaussian Models: Fit multivariate Gaussian $\mathcal{N}(\mu_j, \Sigma_j)$ for each benchmark B_j using its feature vectors

$n_j \leftarrow 0,\ \forall j \in \{1, \ldots, M\}$ {Initialize match counts for each benchmark}

for each training sample $x_i \in \mathbb{U}$ **do**

  $\ell_j = \log \mathcal{N}(\phi(x_i) \mid \mu_j, \Sigma_j),\ \forall j \in \{1, \ldots, M\}$ {Log-likelihoods under each benchmark's Gaussian}

  $k = \arg\max_j \ell_j$ {Assign to benchmark with highest likelihood}

  $n_k \leftarrow n_k + 1$ {Increment matched sample count for B_k}

end for

for each benchmark $B_j \in \mathcal{B}$ **do**

  $f_j = n_j / N$ {Compute frequency}

  $r_j = \log(1/f_j)$ {Compute rarity}

end for

return $\{r_1, \ldots, r_M\}$ {Rarity scores for all benchmarks}

malized between $[0, 1]$ [3].

Quantifying Performance Improvement. To isolate the impact of our core contribution—Prioritized Concept Learning (PCL) described in Section 3.2—we measure the performance improvement brought solely by our dynamic sample selection strategy. Specifically, we compute the difference in performance between the full PROGRESS framework (Table 3, Row 7) and the warm-up only model trained prior to applying PCL (Table 4, Row 1). This comparison quantifies the gain attributable to dynamically selecting the most informative samples using our PCL strategy during training.

7.8.2. Details for Data frequency Analysis

Here, we provide details regarding the analysis shown in Figure 8 (b) of main manuscript.

[3]For MME, where the full score is not out of 100, we normalize the score by dividing it by the maximum score (2800), the difficulty is computed as $(1 - \text{MME Score}/2800)$

Sample Rarity Estimation. Our goal is to identify, for each sample in the training dataset, the benchmark it most closely aligns with in terms of skill distribution. Each training sample is assigned to exactly one benchmark—whichever it is closest to—based on similarity in distribution over skills. This allows us to estimate the frequency of training samples aligned with each benchmark, enabling us to quantify how commonly each benchmark's skills are represented in the training data.

Assignment Procedure. To quantify how training samples align with various benchmarks, we use a Gaussian modeling approach. Specifically, we first extract visual and textual features using DINO (for images) and BERT (for questions) and form joint multimodal embeddings as described in Section 3.1—for all samples in training data and each benchmark.

Next, we fit a multivariate Gaussian distribution to each benchmark's embeddings, capturing its mean and covariance to model the underlying skill distribution. Then, for every training sample, we compute the log-likelihood under each benchmark's Gaussian model, reflecting how well the sample fits that benchmark's distribution. Each training sample is then assigned to the benchmark with the highest log-likelihood (refer to Algorithm 1 for full details).

We compute the frequency of training data samples aligned with each benchmark as the proportion of training samples assigned to it:

$$\text{frequency} = \frac{\text{\# matched samples}}{\text{total training samples}}.$$

Finally, we define the rarity as:

$$\text{rarity} = \log(1/\text{frequency}).$$

This formulation enables us to assess how frequently the skills associated with each benchmark appear in the training set (see rarity calculation Algorithim 1).

7.9. Details for Analysis - What Skills Does the Model Prioritize and When?

In this section, we elaborate on the analysis from Section 4.4 (specifically Fig. 9 in main manuscript), where we investigate which skills the model prioritizes and when during training.

Our goal is to identify the specific ability each skill-cluster—obtained through our concept categorization module described in Section 3.1)—represents and track both the number of selected samples and the performance of that skill over time. To do this, we assign each skill cluster in our framework to one of the standardized ability categories defined by the MME benchmark (i.e *count*, *position*, *OCR* etc) which offer interpretable and fine-grained labels covering both perception and cognitive tasks. To determine the

Algorithm 2 Ability Assignment for Clusters

Require: Training dataset $\mathbb{U} = \{x_1, \ldots, x_N\}$ where $x_i = (I_i, Q_i)$; Clusters $\mathcal{C} = \{C_1, \ldots, C_K\}$; samples from MME benchmark $\mathcal{B} = \{b_1, \ldots, b_M\}$ with ability labels; similarity threshold $\alpha = 0.9$; top-k nearest neighbors ($k = 5$)

Feature Extraction: Extract DINO features for images and BERT features for questions; concatenate to form joint feature vectors, for all training and benchmark samples.
We denote DINO-BERT embedding of x_i as $\phi(x_i)$.
for each cluster $C_k \in \mathcal{C}$ **do**
  for each sample $x_i \in C_k$ **do**
    $\mathcal{N}_i = \{\text{TopK}(\text{sim}(\phi(x_i), \phi(b_j))_{j=1}^{M})\}$ where $\text{sim}(\phi(x_i), \phi(b_j)) = \cos(\phi(x_i), \phi(b_j))$
    $\mathcal{N}_i' = \{b_j \in \mathcal{N}_i \mid \text{sim}(\phi(x_i), \phi(b_j)) \geq \alpha \cdot \max_j \text{sim}(\phi(x_i), \phi(b_j))\}$
    $\mathcal{A}_i = \{\text{ability}(b_j) \mid b_j \in \mathcal{N}_i'\}$
  end for
  $\mathcal{A}_k = \bigcup_{x_i \in C_k} \mathcal{A}_i$ {Aggregate ability labels from all samples in C_k}
  $\text{Ability}(C_k) = \text{Mode}(\mathcal{A}_k)$ {Assign the most frequent ability via majority vote}
end for
return $\{\text{Ability}(C_1), \ldots, \text{Ability}(C_K)\}$

dominant ability for each skill-cluster , we use a similarity-based assignment procedure (see Algorithm 2 for details).

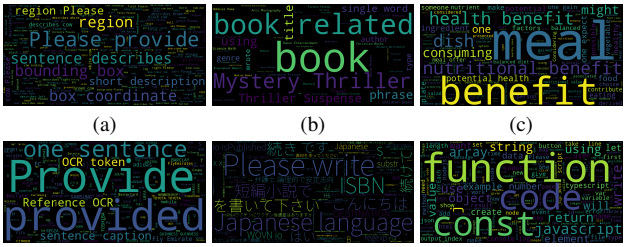
We first extract visual and textual features using DINO (for images) and BERT (for questions) and form joint multimodal embeddings as described in Section 3.1—for all samples in training data and MME benchmark dataset.

For each training sample in a given skill-cluster generated by our concept categorization module, we compute its cosine similarity with all samples in the MME benchmark. We identify its top-K nearest neighbors in MME benchmark and retain only those with similarity above a 90% threshold, ensuring that we capture the most aligned MME samples for each training example. MME abilities associated with these filtered neighbors are aggregated for samples in the cluster, and majority voting is applied to assign the most frequent ability to the entire cluster. This process offers a principled way to characterize each skill-cluster's dominant visual-linguistic ability, ensuring robustness through both similarity filtering and voting (refer to Algorithm 2 for more details).

8. Analysis

8.1. Word Cloud Visualization of Skill Clusters

To qualitatively assess the semantic coherence and purity of discovered skill clusters obtained through our concept categorization module (Section 3.1), we generate word clouds by aggregating all questions from all samples assigned to a given cluster. For each cluster, we concatenate all the corresponding questions into a single string and visualize the most frequent words using wordclouds. Note that we remove standard stopwords while plotting the wordclouds.



(a) (b) (c) (d) (e) (f)

Figure 12. **Word Cloud Visualization of Skill Clusters.** Each subfigure shows the word cloud generated by concatenating all questions within a single skill-cluster discovered by our unsupervised concept categorization module. The clusters exhibit clear semantic themes: (a) object localization and region descriptions, (b) book metadata and genres, (c) food and nutritional benefits, (d) OCR, (e) multilingual Japanese text and language prompts, and (f) programming and function-related instructions. These word clouds highlight the semantic coherence and fine-grained granularity of the automatically discovered clusters, validating their utility for skill-level progress tracking.

Figure 12 shows representative word clouds for six clusters. Each cluster exhibits a distinct semantic theme, validating the purity and fine-grained granularity of the automatically discovered clusters and demonstrating the effectiveness of our multimodal concept categorization. For example, cluster (a) object localization and region descriptions, (b) book metadata and genres, (c) pertains to food and nutritional benefits, (d) corresponds to OCR and reference tokens, (e) involves multilingual Japanese text and language prompts, and (f) highlights programming and function-related tasks.

These visualizations demonstrate that our clustering method forms fine-grained, interpretable concept groupings while being fully unsupervised (see Section 3.1)—essential for skill-level tracking and prioritized learning in PROGRESS.

8.2. Skill-level Diversity in Selected Sample Distribution

To better understand the selection behavior across methods, we follow protocol in previous work [19] and analyze the number of selected samples from each task in the Vision-Flan-191K dataset. Figure 13 shows the task-wise sample distribution for PROGRESS and several baseline approaches.

We observe that methods relying on single static scoring functions—such as CLIP-Score, EL2N, Perplexity, and

Self-Sup—tend to exhibit strong sampling bias, disproportionately selecting from a small subset of tasks while neglecting others. This narrow focus often overlooks important data modes, leading to poor generalization—a limitation also noted in prior work [19].

In contrast, PROGRESS maintains a more balanced and diverse sampling profile across tasks, ensuring that a broader range of skills and task types are represented during training. This diversity stems from our skill-driven selection strategy, which tracks learning progress across clusters and samples proportionally using a temperature-controlled distribution.

Overall, by avoiding the pitfalls of static scoring and overfitting to specific high-scoring skills or frequent tasks, our method instead promotes broader and more effective skill acquisition.

9. Limitations

While PROGRESS effectively orders and prioritizes more informative skills, it randomly samples within each selected skill cluster without ranking samples by usefulness. Additionally, the accuracy-based variant incurs extra inference time to compute skill-level progress (see Appendix 6), though our loss-based variant avoids this issue. However, overall, PROGRESS outperforms prior approaches while requiring no additional reference VLM and significantly less supervised data.

10. LLM Usage Disclosure

The LLM was primarily used for language refinement rather than any content generation—all experimental designs, results, analyses, and scientific contributions are original work by the authors. The LLM assistance was limited to editorial improvements such as fixing grammatical errors, suggesting clearer phrasing for complex technical concepts, and ensuring consistency in terminology throughout the manuscript. No experimental results, mathematical derivations, or scientific claims were generated by the LLM. All factual statements, citations, and technical content were independently verified by the authors.

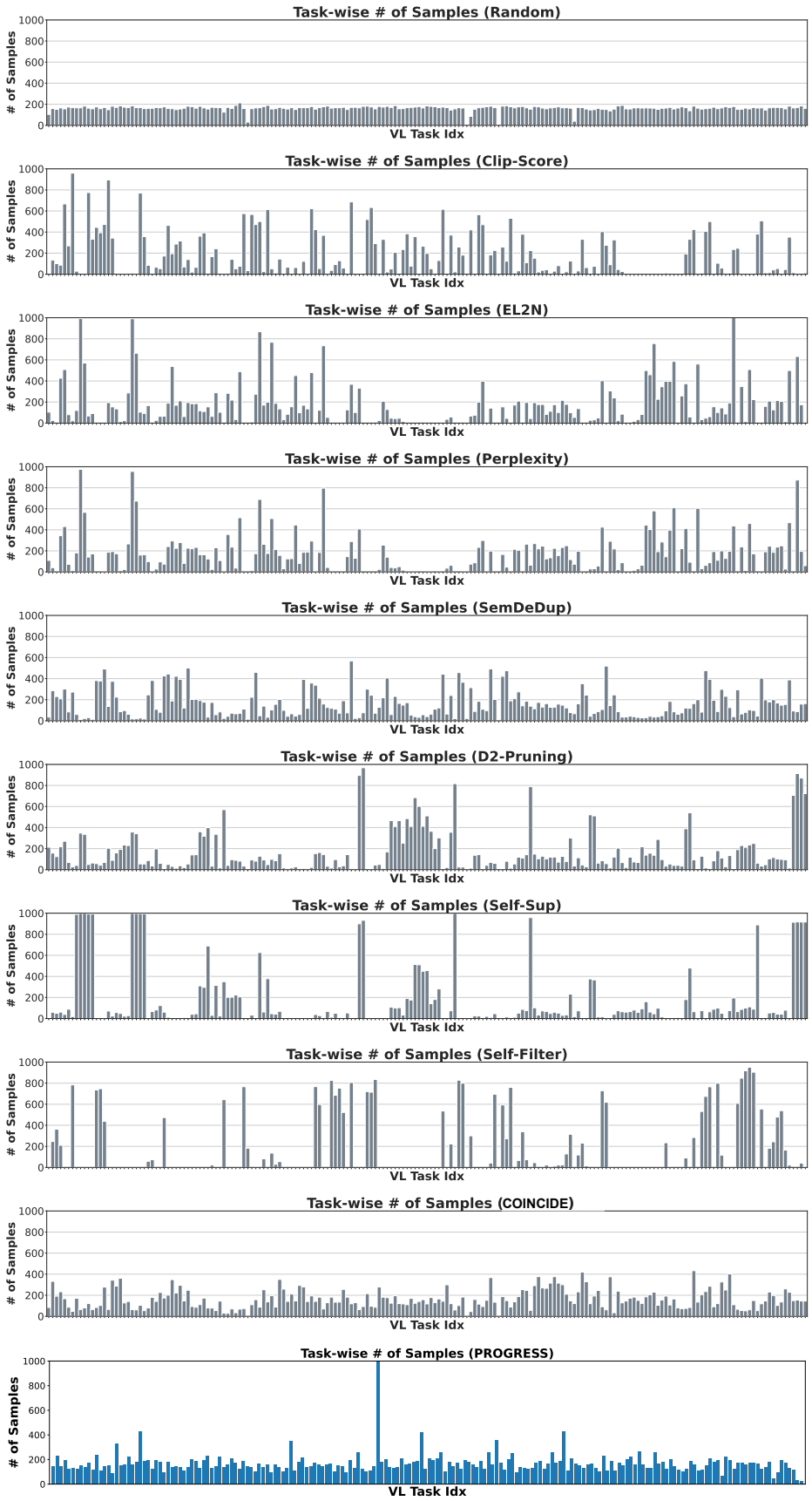


Figure 13. **Task-wise Distribution of Selected Samples.** Number of samples selected (y-axis) from each Vision-Flan-191K task (x-axis) across different methods. While baselines tend to concentrate heavily on a few high-scoring tasks, PROGRESS achieves a more balanced sampling pattern across the task spectrum—highlighting its ability to maintain skill diversity.