
Supplemental Material: Computational Budget Should Be Considered in Data Selection

In this appendix, we provide comprehensive additional materials to supplement the main text. The contents include:

- **Broader impacts (Section A):** A discussion on the broader implications of our research.
- **Experimental setting and implementation details (Section B):** Detailed information on the experimental settings and implementation details.
- **Log-linear surrogate of compute-constrained training loss (Section C):** Explanation of approximating $\mathcal{L}_{\text{trn}}(\mathbf{u}, \mathbf{m})$ by the scale-dependent surrogate $l(|\mathbf{m}|)$, its log-space interpolation, and fitting procedure.
- **Visualizations on truncated Gaussian distribution (Section D):** Detailed visualization on the truncated Gaussian distribution.
- **Additional experimental results (Section E):** MNIST experiments varying the total compute budget to compare random, PBCS, CADS-E, and Bilevel-CADS. The results show that our method consistently achieves the highest accuracy (Table 9).
- **Time efficiency analysis (Section F):** Benchmarking of average training and sampling runtimes on representative hardware.
- **Scalability to other bilevel optimization problems (Section G):** Discussion on the applicability conditions, loss estimation accuracy, and a validation framework for applying CADS to new domains.
- **Limitations and future work (Section H):** Analysis of the limitations in our current framework, and plans for future improvements.
- **Licenses for existing assets (Section I):** Acknowledgment and respect for the licenses and terms of use of datasets and code libraries utilized in our research.

A Broader impacts

This work introduces CADS, a compute-aware data selection algorithm that dynamically adapts the training data budget according to available computational resources. By optimizing data efficiency relative to compute constraints, CADS significantly reduces the computational overhead of training deep learning models without sacrificing performance. This advancement enables more accessible and environmentally sustainable machine learning research and deployment, particularly in scenarios constrained by limited hardware resources. The method’s potential to lower energy consumption during model training aligns with broader community efforts towards greener AI, contributing to reductions in carbon footprint and operational costs. Additionally, CADS facilitates democratization of deep learning by empowering researchers and practitioners with modest resources to train competitive models. However, as with any automation that accelerates model training, there exist potential risks, including the possibility of speeding up the development of models with harmful biases, privacy vulnerabilities, or malicious intent if not applied responsibly. We stress the importance of ethical use and compliance with community norms and regulatory standards when deploying such techniques. Overall, CADS represents a step forward in efficient and responsible machine learning practices, promoting sustainability and equitable access within the deep learning community.

B Experimental setting and implementation details

In this section, we detail the protocols and implementation specifics underlying all our evaluations. All experiments were run on a single machine equipped with an NVIDIA A100 80 GB GPU under CUDA 12.6 and NVIDIA driver 470.199.02, using PyTorch 2.5.1.

B.1 Exploratory experiment on spectrum bias

To isolate the effect of spectral content on generalization, we generate two synthetic datasets with identical noise level $\sigma = 0.1$ but different frequency composition.

Data generation. We sample $N = 50,000$ inputs for each group; for the low-frequency dataset (Group A) we set $y = x + \mathcal{N}(0, \sigma^2)$, and for the high-frequency dataset (Group B) we set $y = x + \frac{\sin(\pi x)}{\pi x} + \mathcal{N}(0, \sigma^2)$. A fixed validation set of 10,000 points is used to track generalization error.

Model and training. Each model is a three-layer MLP (100–100 hidden units, ReLU). We train for 160 steps using Adam (learning rate 3×10^{-4}) and batch size 1000. After every parameter update, we compute the MSE on the validation set and log the result. The different positions on the x-axis in Figure 1 correspond to these intermediate results recorded during the training process.

B.2 Small-scale validation

We validate CADS on a reduced-scale MNIST classification task using a simple convolutional network. All experiments use a fixed training set of 1,000 examples and batch size 1,000. We optimize under an epoch-based budget of 20 epochs. The selection parameters $\mathbf{s} \in \mathbb{R}^{|D|}$ are initialized as $\mathbf{s} = v \cdot \mathbf{1}$, $v \in \{0.2, 0.4, 0.6, 0.8\}$, $\mathbf{1} \in \mathbb{R}^{|D|}$ is the all-ones vector. We solve the bilevel problem with Adam for both the network parameters θ (learning rate 5×10^{-3}) and the selection weights \mathbf{s} (learning rate 5×10^{-2}). The outer loop runs for 300 iterations with variance reduction and gradient clipping enabled.

Network architecture. We employ a simple convolutional model: two convolutional layers with 5×5 kernels and channel counts $\{32, 64\}$, each followed by ReLU and optional 2×2 max-pooling; the feature map is flattened ($4 \times 4 \times 64 = 1024$ units) and passed through a fully connected layer of 128 units with ReLU; a final linear layer outputs class logits. Input normalization by fixed mean and standard deviation is applied when enabled. No dropout is used.

B.3 Heterogeneous data sources: small-scale experiments on CIFAR-10

We evaluate CADS-S on a grouped CIFAR-10 variant with five demographic groups and up to 90% label noise. All experiments use the full training set (no limit) with batch size 256. We allocate an epoch-based compute budget of 2 to 5 epochs. The selection parameter \mathbf{s} is initialized to $0.5 \cdot \mathbf{1}^{|D|}$. We solve the bilevel problem with Adam for both network parameters θ (learning rate 5×10^{-3}) and selection weights \mathbf{s} (learning rate 5×10^{-2}), over 100 outer iterations with variance reduction and gradient clipping. We adopt the standard ResNet-18 backbone for all runs.

B.4 Heterogeneous data sources: scaling to larger datasets

We evaluate CADS-S on DomainNet with 6 groups using the full training set and batch size 1024. We allocate an epoch-based compute budget of 10 epochs. The selection parameters $\mathbf{s} \in \mathbb{R}^{|D|}$ are initialized as $\mathbf{s} = v \cdot \mathbf{1}$, $v \in \{0.2, 0.4, 0.6, 0.8\}$, $\mathbf{1} \in \mathbb{R}^{|D|}$ is the all-ones vector. We solve the bilevel problem with Adam for both network parameters θ (learning rate 5×10^{-3}) and selection weights \mathbf{s} (learning rate 5×10^{-2}), over 100 outer iterations with variance reduction and gradient clipping. We adopt the standard ResNet-18 backbone for all runs.

DomainNet dataset. DomainNet comprises over 0.5 million images across six visual domains: Real, Sketch, Clipart, Painting, Infograph, and Quickdraw (As shown in Fig. 6). In our setup, we treat the Real domain as a high-quality source but include only 10,000 samples from it to simulate limited access; the other five domains are used in full.

B.5 Instruction tuning tasks: small-scale experiments

We evaluated our CADS-S approach on instruction fine-tuning tasks using the GPT-2 model [48]. The dataset consists of two heterogeneous data sources: (i) **Alpaca-GPT4** [46], a high-quality dataset from GPT-4-generated instructions, from which we sample 1,000 examples to simulate the rarity

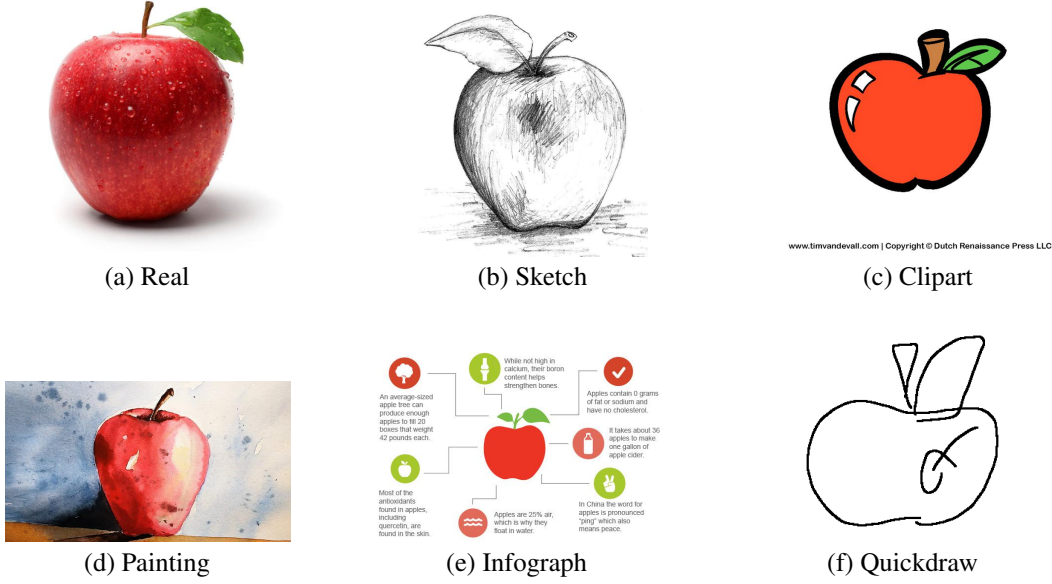


Figure 6: Representative examples from the DomainNet domains: Real, Sketch, Clipart (top row); Painting, Infograph, Quickdraw (bottom row).

of premium data sources in practical scenarios; (ii) **Alpaca** [57], a larger dataset containing 9,000 examples, representing standard-quality data. The total compute budget varies within the range $[1 \times 10^4, 5 \times 10^4]$ sample usages. The selection weight s is initialized to $0.5 \cdot \mathbf{1}^{|D|}$. We solve the bilevel problem with AdamW for both model parameters θ (learning rate 1×10^{-5}) and selection weights s (learning rate 5×10^{-2}). We adopt the GPT-2 backbone with maximum sequence length 1024 for all runs.

Data preprocessing. Raw instruction–response samples are loaded from JSON or JSONL files and consolidated into a flat record list. A pretrained tokenizer is initialized and extended to include five special markers: a padding token, an end-of-sequence token, and three role-demarcation tokens (`<system>`, `<user>`, `<assistant>`). Each record is then serialized into a single text sequence by:

1. Emitting the system token, followed by the system-level directive, and two line breaks.
2. Emitting the user token, followed by the instruction text (and any optional input), and two line breaks.
3. Emitting the assistant token, followed by the target response, and terminating with the end-of-sequence token.

The concatenated text is first tokenized without truncation to measure its length; any example that exceeds the maximum allowed token count is discarded. The remaining examples are split into training, validation, and test subsets according to either explicitly provided sizes or a default 90/5/5 ratio. At training time, each sequence is tokenized with truncation to the maximum length, producing token identifiers and attention masks. All token positions corresponding to the system and user segments are assigned an ignore index in the label sequence so that only assistant-response tokens contribute to the loss. A bespoke collation routine then pads every batch to the length of its longest sequence, using the padding token for inputs, zeros for masks, and the ignore index for both prompt and padding positions in the labels.

B.6 Instruction tuning tasks: scaling to additional datasets

We extended our evaluation to 13 distinct instruction-following corpora, drawing ten thousand examples from each. These included the original AlpacaGPT4 benchmark[46], the SlimOrca set[32], the Alpaca collection[57], the GPTeacher suite[58], and nine multilingual variants of the Alpaca data.

In total, approximately 130 000 samples were processed with the identical tokenization, formatting, filtering, and batch-collation pipeline described above. The complete list of datasets, along with their sizes and licenses, is summarized in Table 7.

Dataset	Size	License
AlpacaGPT4 [46]	52K	Apache-2.0
SlimOrca [32]	518K	MIT
GPTeacher [58]	89K	MIT
Alpaca [57]	52K	Apache-2.0
Alpaca-es ¹	52K	CC-BY-4.0
Alpaca-de ²	50K	Apache-2.0
Alpaca-ja ³	52K	CC-BY-NC-SA-4.0
Alpaca-ko ⁴	50K	CC-BY-NC-4.0
Alpaca-ru ⁵	30K	CC-BY-4.0
Alpaca-it ⁶	52K	CC-BY-NC-SA-4.0
Alpaca-fr ⁷	55K	Apache-2.0
Alpaca-zh ⁸	49K	CC-BY-4.0
Alpaca-pt ⁹	52K	CC-BY-NC-4.0

Table 7: Summary of high-quality instruction-tuning datasets (top) and multilingual Alpaca variants (bottom).

C Log-linear surrogate of compute-constrained training loss

C.1 Data Collection

We collect estimates of the K -step (“compute-constrained”) training loss $\mathcal{L}_{\text{trn}}(\theta; |m|)$ at a range of subset sizes $|m|$. By default we choose nine relative fractions $\{0.01, 0.02, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9\}$ of the full training set (clamped to at least 50 examples on MNIST or the configured batch-size otherwise). For each size we train for a total compute budget N (so that each inner training run sees roughly the same total number of gradient steps), and record the final training loss. We then summarize each size by its empirical ℓ_j .

C.2 Surrogate model choice

We set $\varepsilon = 10^{-10}$ and fit the transformed log-loss $\log(l(|m|) + \varepsilon)$ using two options:

- **Linear:** $\log(l(|m|) + \varepsilon) = k|m| + b$.
- **Cubic spline interpolation:** fit a cubic spline through the points $(|m_j|, \log(l(|m_j|) + \varepsilon))$, yielding a piecewise-cubic function $\hat{f}(|m|)$.

¹<https://huggingface.co/datasets/bertin-project/alpaca-spanish>

²https://huggingface.co/datasets/mayflowergmbh/alpaca-gpt4_de

³https://huggingface.co/datasets/fujiki/japanese_alpaca_data

⁴https://huggingface.co/datasets/Bingsu/ko_alpaca_data

⁵https://huggingface.co/datasets/IlyaGusev/ru_turbo_alpaca

⁶https://huggingface.co/datasets/mchl-labs/stambecco_data_it

⁷<https://huggingface.co/datasets/jpacifico/French-Alpaca-dataset-Instruct-55K>

⁸<https://huggingface.co/datasets/llm-wizard/alpaca-gpt4-data-zh>

⁹<https://huggingface.co/datasets/dominguesm/alpaca-data-pt-br>

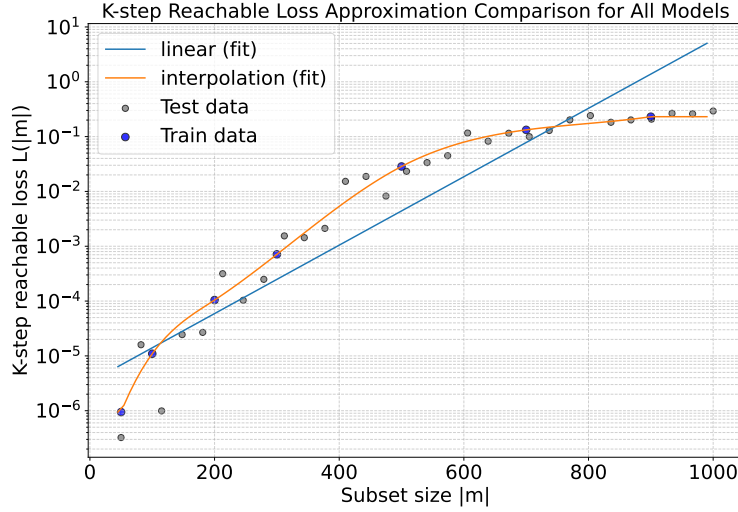


Figure 7: All-models comparison of the K-step reachable loss as a function of subset size $|m|$. Blue circles denote observed mean loss on training subsets; gray squares denote observations on held-out test subsets. Solid curves show the fitted approximation functions for each candidate model. The vertical axis is plotted on a logarithmic scale.

After fitting in log-space, we recover the original-scale surrogate via $l(|m|) = \exp(\hat{f}(|m|))$. Figure 7 overlays these fitted curves on the empirical training-loss measurements. The cubic spline interpolation achieves a noticeably lower MSE, motivating its use when maximum fidelity is desired, while the linear surrogate offers simplicity and a closed-form gradient. **Since CADs never computes $\partial l(|m|)/\partial |m|$, we exclusively employ the cubic spline interpolation due to its superior fit.**

C.3 Analysis of Sampling Efficiency for Loss Estimation

To evaluate the sample efficiency of our loss estimation method, we conducted an analysis to determine the minimum number of subset samples (K) required for reliable interpolation.

Experimental Design On the CIFAR-10 dataset with a ResNet-18 model, we evaluated the loss estimator’s performance under varying sampling densities, where $K \in \{4, 5, 6, 7, 8\}$. For each value of K , we sampled K distinct subset sizes to train corresponding models, yielding K pairs of (size, loss) data points. These points were used to fit the loss estimator. Subsequently, we tested each fitted estimator on a held-out set of 100 separately sampled (size, loss) pairs to measure its interpolation accuracy.

Results The results, summarized in Table 8, demonstrate that the estimator’s performance improves significantly up to $K = 5$ sampling points, after which the returns diminish. This indicates that our method is highly sample-efficient. Notably, increasing K further (e.g., $K = 7$) can slightly degrade performance, likely due to overfitting the estimator to a specific set of sample points, which hinders its ability to generalize to unseen subset sizes.

D Visualizations on truncated Gaussian distribution

In this section we provide visual intuition for the source-level sampling prior by plotting the truncated Gaussian density $p(r | s)$ on the interval $[0, 1]$ for a variety of center values s and scales σ . Figures 8, 9, and 10 illustrate how annealing σ concentrates the distribution around its mean.

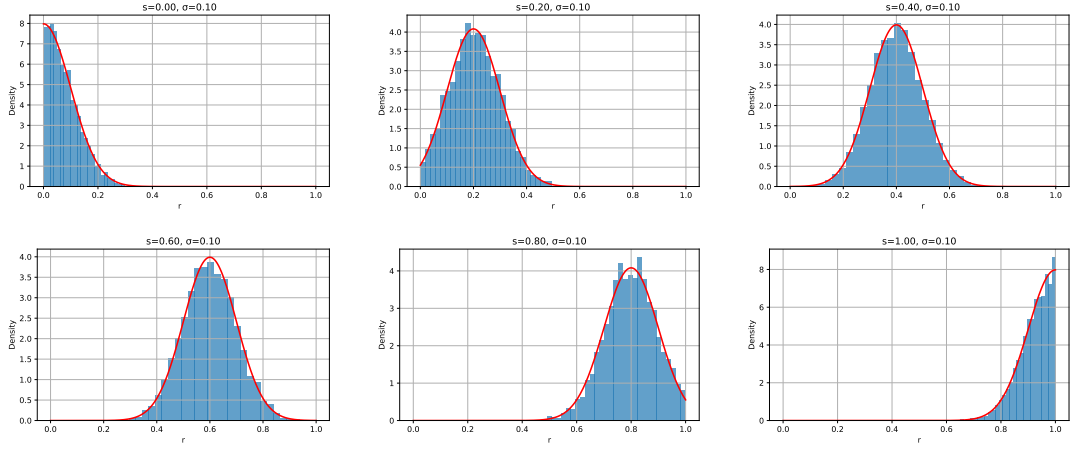


Figure 8: Sampling prior $p(r | s)$ on $[0, 1]$ for various center values s with scale $\sigma = 0.1$.

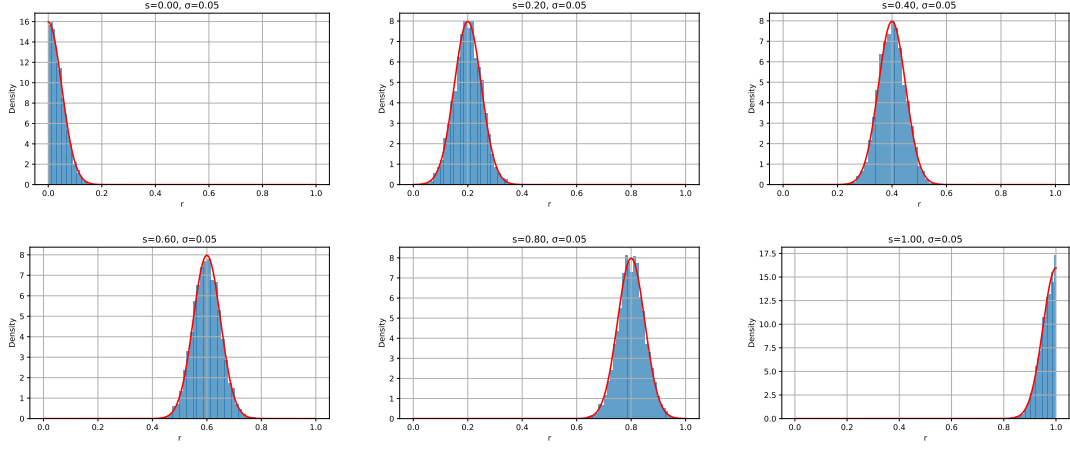


Figure 9: Sampling prior $p(r | s)$ on $[0, 1]$ for various center values s with scale $\sigma = 0.05$.

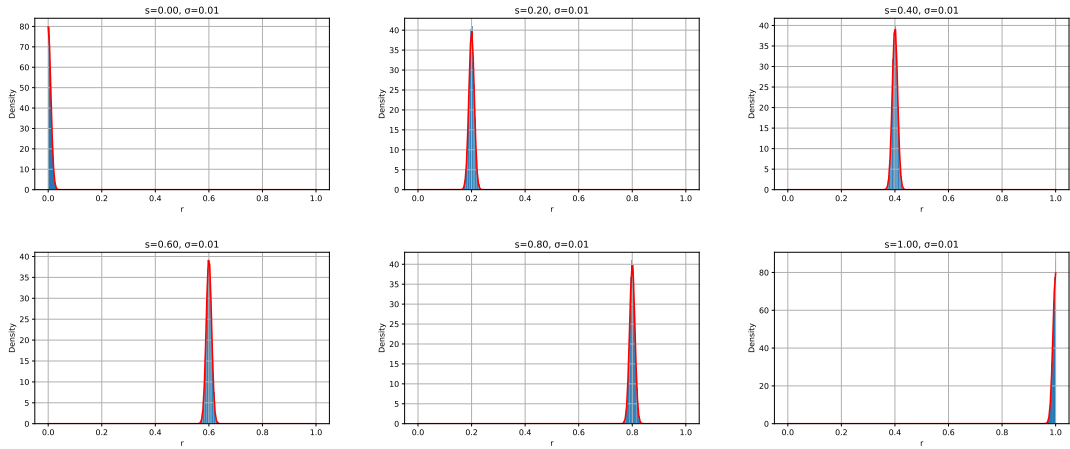


Figure 10: Sampling prior $p(r | s)$ on $[0, 1]$ for various center values s with scale $\sigma = 0.01$.

Table 8: Mean Square Error (MSE) of the loss estimator with respect to the number of sampling points (K). Performance saturates at $K = 5$, demonstrating high sample efficiency.

Sampling Points (K)	Mean Square Error
4	0.057668
5	0.016326
6	0.018594
7	0.020031
8	0.017905

E Additional experimental results

Additional result on MNIST The purpose of this experiment is to investigate the effect of compute budget on data selection within the MNIST dataset. We vary the compute budget in terms of total forward counts (10 000, 20 000, 50 000 and 100 000). At each budget we compare four sampling strategies—random, PBCS, CADS-E and our Bilevel-CADS. As shown in Table 9, our Bilevel-CADS consistently outperforms all baselines in all compute budgets.

Table 9: Results on MNIST with various compute budgets.

Method	Compute budget				Average
	10,000	20,000	50,000	100,000	
Random	87.36	88.05	87.71	87.24	87.59
PBCS	89.61	90.48	90.96	90.41	90.37
CADS-E	90.62	91.48	92.45	93.17	91.93
Bilevel-CADS	90.70	92.44	93.09	93.61	92.46

F Time efficiency analysis

F.1 Computational complexity compare to standard bilevel optimization

We compare the per-iteration cost of our single-epoch inner loop against the full-training inner loop used in bilevel-CADS, and then account for the one-time cost of fitting the loss estimator $l(|\mathbf{m}|)$.

Let

- $|\mathcal{D}|$ be the size of the training set,
- $T_{\text{ep}} = O(|\mathcal{D}|)$ the cost of one full epoch (one forward+backward pass),
- N the number of epochs used by bilevel-CADS in its inner optimizer,
- K the number of subsets sampled per outer iteration,
- $|\mathbf{m}|$ the average subset size,
- M the total number of outer iterations in a run.

Bilevel-CADS inner solve (per outer step)

$$\text{Cost}_{\text{bilevel}} = K \times N \times T_{\text{ep}} = O(KN|\mathcal{D}|).$$

Our CADS-E/S inner loop (per outer step, ignoring estimator)

$$\text{Cost}_{\text{CADS}}^{\text{base}} = K \times T_{\text{ep}} = O(K|\mathcal{D}|).$$

Estimator fitting overhead To fit the mapping $l(|\mathbf{m}|)$ we train 8 models for N epochs each (As detailed in D), incurring a one-time cost

$$\text{Cost}_{\text{est}} = 8 \times N \times T_{\text{ep}}.$$

Amortized over M outer steps, this adds

$$\frac{\text{Cost}_{\text{est}}}{M} = \frac{8 N T_{\text{ep}}}{M}$$

to each iteration.

Total CADS cost (amortized)

$$\text{Cost}_{\text{CADS}} = K \times T_{\text{ep}} + \frac{8 N T_{\text{ep}}}{M} = T_{\text{ep}} \left(K + \frac{8 N}{M} \right).$$

Speed-up factor

$$\frac{\text{Cost}_{\text{bilevel}}}{\text{Cost}_{\text{CADS}}} = \frac{K N T_{\text{ep}}}{T_{\text{ep}} \left(K + \frac{8 N}{M} \right)} = \frac{K N}{K + 8 N/M}.$$

In our experiments $K = 5$ and $M = 100$, giving

$$\frac{\text{Cost}_{\text{bilevel}}}{\text{Cost}_{\text{CADS}}} = \frac{1}{1/N + 0.016}.$$

In practice, larger N yields an even greater speed-up.

F.2 Empirical Validation

We measure end-to-end wall-clock time of bilevel-CADS and CADS-E on MNIST as a function of compute budget C (total epochs). We sweep C over $\{5, 50, 100, 200, 500, 1000, 2000, 5000\}$ and record the runtime of each method under identical hardware. Figure 11 confirms a roughly linear scaling for both methods, with CADS-E exhibiting a much smaller slope.

F.3 Analysis of Selection Algorithm Runtime

A potential concern regarding our method is that the runtime of the selection algorithm may scale with the number of sampled subsets (K), which could offset the computational efficiency gains. We address this concern by highlighting several key aspects of our approach.

Minimal Sampling for Practical Implementation Our policy gradient approach for coreset selection requires only a small number of samples (K) to ensure training stability. For instance, the comparable PBCS method [74] utilizes $K = 1$ for all its experiments. Our results show that using $K = 2$ is sufficient to achieve strong performance while maintaining high computational efficiency. In cases where smaller K values might lead to higher gradient variance (e.g., in more complex problems), this can be mitigated using established variance reduction techniques. A common approach is to use a self-critical baseline, which adjusts the reward signal as follows:

$$\tilde{\mathcal{L}}(S_i) = \mathcal{L}(S_i) - \frac{1}{K} \sum_{j=1}^K \mathcal{L}(S_j). \quad (12)$$

Amortized Cost of Coreset Selection It is important to emphasize that the core contribution of this work is the introduction of computational budget constraints into the coreset selection process, rather than optimizing the selection runtime itself. Once generated, a coreset is a reusable asset that can be applied across various training scenarios, different model architectures, and subsequent experiments. Consequently, the initial computational investment in the selection process is amortized over these multiple uses. This makes the selection efficiency a secondary consideration compared to the quality and utility of the resulting budget-constrained coreset.

G Scalability to Other Bilevel Optimization Problems

CADS is designed to address a fundamental challenge in bilevel optimization: scenarios where the inner-level problem’s convergence is hindered by computational budget constraints. Many practical bilevel problems suffer from this issue, where traditional methods that assume unlimited computational resources for inner-level convergence often fall short. We have also successfully applied a CADS-like method to resource-intensive diffusion models, demonstrating its potential beyond the scope of this work. The scalability of CADS to other bilevel problems primarily depends on the following factors:

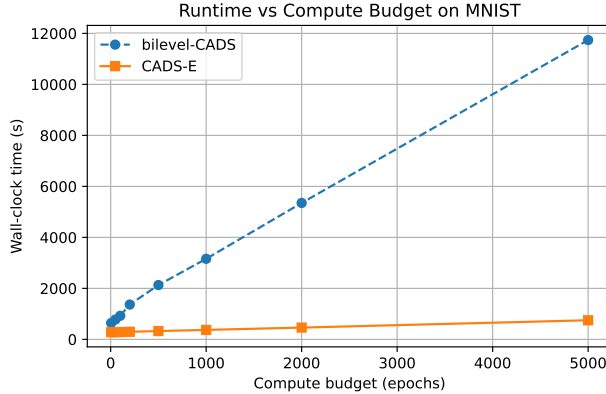


Figure 11: Wall-clock runtime vs. compute budget C for bilevel-CADS (dashed) and CADS-E (solid), MNIST.

Applicability Conditions The core premise of CADS is most relevant when the inner-level optimization is computationally expensive and cannot be fully converged. Our approach provides a framework to handle such resource-constrained scenarios, which are common in real-world applications but often overlooked by conventional methods.

Loss Estimation Accuracy The effectiveness of CADS in a new domain hinges on the accuracy of the loss estimator. As demonstrated in our analysis of loss estimation generalization, the proposed estimator maintains relatively stable predictive performance across the diverse experimental domains explored in this paper. This stability suggests that similar performance may be achievable in other domains with comparable data characteristics.

Validation Framework for New Domains To adapt CADS to a new bilevel optimization problem, we propose the following validation framework:

- **Loss Estimator Generalization Assessment.** Before full-scale implementation, it is crucial to assess whether the data distribution characteristics of the new domain can be reliably predicted. This can be achieved by conducting experiments similar to our generalization evaluation to validate the feasibility of loss estimation.
- **Domain-Adaptive Estimator Design.** While our current estimator is effective for data selection problems, different bilevel scenarios might benefit from specialized estimators. The architecture of the loss estimator should be tailored to the specific requirements and data characteristics of the target domain to ensure optimal performance.

H Limitations and Future Work

Similar to existing methods, our approach is based on standard training methodologies, but in the industry, large language models incorporate numerous engineering optimizations. Consequently, our budget is likely proportional to theirs, rather than perfectly aligned. For instance, in sparse mixture of experts (MoE), the budget is determined by model capacity and routing complexity, whereas we measure performance by the number of forward passes. This measurement may not fully correspond with the budget metrics used in industry. However, we firmly believe that our methods are applicable in industrial contexts; we simply need to adjust the budget measurements accordingly.

I Licenses for existing assets

We rely on several public datasets and open-source libraries, and all original authors are properly credited and their licenses fully respected. The vision benchmarks we employ are MNIST (public domain), CIFAR-10 (MIT License) and DomainNet (CC BY-NC-SA 4.0). Our instruction-tuning

corpora are summarized in Table 7, with licenses ranging from Apache 2.0 to various Creative Commons and MIT terms. On the software side, we build atop PyTorch and torchvision (BSD 3-Clause), SciPy (BSD), and HuggingFace Transformers (GPT-2, Apache 2.0); ResNet models are taken from torchvision (BSD 3-Clause). We confirm that every dataset and code dependency is used in accordance with its license and citation requirements.