DSBox: A Data Selection Framework for Efficient Deep Code Learning

Xinyang Liu[1], Lili Quan[1], Qiang Hu[1*]
[1]Tianjin University, China

***Abstract*—Deep Learning has achieved remarkable advancements in various software engineering tasks and gained huge attention in the community. Following a data-centric paradigm, the preparation of code models requires high-quality datasets for the model training. However, constructing such datasets, especially for software tasks, is costly mainly due to the data labeling process. To address this challenge, multiple data selection methods have been proposed to identify and label data samples that are important for training. Despite this potential, unfortunately, there are limited tools to support the flexible usage of data selection methods, hindering their practical usage and future research in this domain. To bridge this gap, we introduce *DSBox*, a lightweight yet extensible framework that unifies 20 published selection methods, covering three categories: uncertainty, representativeness, and quality-based methods. Evaluation demonstrates that active learning methods outperform recently proposed techniques (designed for large language models) on the code vulnerability detection task. The tool, as well as a demonstration video, are available on the project website https://sites.google.com/view/dsbox2025.**

I. Introduction

Deep learning models have revolutionized numerous application domains, from autonomous driving to software engineering (SE), achieving impressive performance across a wide range of tasks. This success is largely due to high-quality labeled datasets. However, generating such labels often requires intensive human effort, specialized expertise, and substantial financial investment. Especially in the SE domain, full-stack domain knowledge is required for the data labeling, which is extremely costly. For example, it takes more than one year to prepare the famous SE benchmark dataset BigCodeBench [1].

To alleviate this burden, *data selection* techniques have been developed to identify the most informative or representative samples within a dataset. These methods assign a numerical score to each data instance—based on their design mechanism—and then select a subset that maximizes expected performance under a given label or computational budget. By strategically selecting data points according to these scores, practitioners can significantly reduce labeling effort and computational overhead while maintaining—or even enhancing—downstream model accuracy and robustness.

With the growth of model size (e.g., large language models) and data requirements, data selection becomes more important. A unified data-selection framework is therefore vital for label-efficient learning: it 1) preserves the most informative examples as training datasets to reduce the cost; 2) enables researchers and users to compare alternative importance scoring heuristics on a new domain with a single configuration change, revealing the most cost-effective choice; 3) offers a common test-bed for adding and fairly evaluating novel selection methods. It is, therefore, crucial to develop a generic tool to support different data selection methods.

In this paper, we introduce a unified, plugin-oriented data-selection platform, *DSBox*, that integrates state-of-the-art data selection techniques within a single, coherent framework. To do so, we first classify existing methods into three categories based on their design spirits–uncertainty, representativeness, and quality assessment–based methods. Then, we implement methods per category with a specific interface for further integration of new methods. In the end, *DSBox* computes per-sample selection scores, ranks the dataset, and automatically chooses the top-ranked examples according to a user-defined budget. We demonstrate the utility of *DSBox* on the Devign [2] vulnerability detection task. Experimental results demonstrated that active learning methods, which have been widely studied in the conventional machine learning field, outperform recently proposed data selection methods that are designed for advanced models (i.e., large language models), highlighting the necessity for careful consideration when employing data selection methods.

In summary, this paper makes the following contributions:

- We propose *DSBox*, a generic and comprehensive framework, covering 20 representative data selection methods, to support label-efficient code model training.
- We evaluate the usefulness of *DSBox* on code vulnerability detection tasks and found that conventional active learning methods outperform recently proposed advanced data selection methods, emphasizing the importance of systematic evaluation when developing new methods.

II. The Framework *DSBox*

Fig. 1 illustrates the workflow of *DSBox*, which consists of three stages: method selection, data selection, and model training. Given a code model, a set of data under labeling, *DSBox* uses a data selection method (decided by users) to rank the data samples based on their potential contribution to the model training. Then, given the labeling budget (for example, label 10% of the data), *DSBox* picks out the budget number of samples based on their ranking. After that, these selected samples will be manually labeled by human annotators. Finally, the labeled data will be utilized as training data to optimize the parameters of code models.

[*]corresponding author.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00375

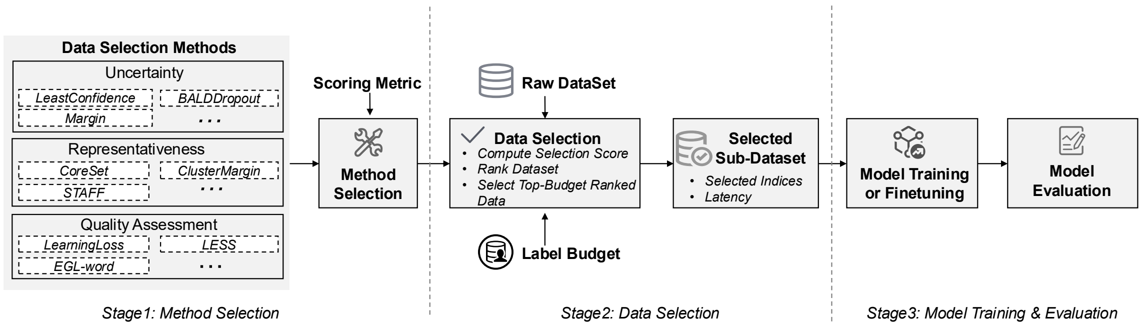

Fig. 1: The workflow of *DSBox*.

TABLE I: Comparison between data selection frameworks.

Benchmark	# Methods	Classification	Non-Classification
Li *et al.* [4]	19	✓	×
Hu *et al.* [5]	12	✓	×
Deep Active Learning [6]	8	✓	×
DSBox (Ours)	20	✓	✓

Framework Comparison. Table I compares existing data selection frameworks with *DSBox*, showing that our tool covers the most selection methods. Moreover, *DSBox* is the only framework that supports methods proposed for non-classification tasks, e.g., generation tasks, such as LESS [3]. This allows the usage of *DSBox* in more diverse code tasks.

A. Data Selection Methods

We collect data selection methods from recent active learning and label-efficient learning works. These methods fall into three principal categories: (i) *uncertainty-based* approaches that prioritize samples for which the current model exhibits low confidence; (ii) *representativeness-based* methods that aim to capture diverse examples across the data or feature space; and (iii) *quality assessment–based* techniques that estimate the potential influence of a sample on parameter updates or loss reduction of models. Mention that, the methods from the first two categories are mainly designed for classification tasks, and *quality assessment–based* methods can be used for non-classification tasks. By combining these three categories, *DSBox* can be applied to any code tasks such as code vulnerability detection and code generation. In total, *DSBox* supports 20 methods. We summarize the core ideas behind these methods as follows.

1) Uncertainty- based Method

- **Least Confidence** [7] selects samples on which the predicted confidence of the model is the lowest.
- **Entropy Sampling** [7] prefers samples whose prediction distributions exhibit the highest Entropy scores.
- **Margin Sampling** [7] targets samples with minimal probability gaps between the top two predicted classes.
- **BALD Dropout** [8] performs T stochastic forward passes with Dropout. For a candidate x, it records the predictions $\{y_x^{(1)}, \ldots, y_x^{(T)}\}$, determines their mode m_x, and assigns the score $\mathrm{BALD}(x) = 1 - \frac{1}{T}\sum_{t=1}^{T} \mathbb{I}[\, y_x^{(t)} = m_x\,]$. Samples with higher $\mathrm{BALD}(x)$–i.e., greater predictive disagreement–are selected.
- **BatchBALD** [9] greedily constructs a batch by maximizing the joint mutual information between the selected points and the model parameters, yielding a $1-1/e$ approximation guarantee that favours complementary—rather than redundant—samples within each acquisition round.
- **Expected Gradient Length (EGL)** [10] estimates a sample's potential impact on the model by considering all labels.
- **SPUQ** [11] generates perturbed versions of each input–e.g., temperature shifts or prompt paraphrases – to reveal epistemic uncertainty, then draws multiple samples per perturbation to capture aleatoric uncertainty; an aggregation module fuses the two signals into a calibrated confidence score.
- **UQ_ICL** [12] views in-context learning through a Bayesian lens, decomposing total predictive uncertainty into aleatoric and epistemic parts via mutual information. An entropy-based estimator quantifies both components without extra supervision.

2) Representativeness-based Method

- **Random** uniformly samples data points and thus provides a simple baseline.
- **BM25** [13] is a lexicon-based method that assigns each sample x a relevance score averaged over its BM25 similarity to all validation items X, i.e., $v_{\mathrm{BM25}}(x) = \frac{1}{|X|}\sum_{x' \in X} \mathrm{BM25}(x, x')$.
- **CoreSet** [14] selects samples via a farthest-first traversal that approximates the k-center objective, ensuring every data point in the feature space lies close to at least one chosen exemplar and thereby providing representational coverage.
- **KMeans** [15] partitions the feature embeddings into k clusters and, from each cluster, selects the sample closest to its centroid, yielding a diverse set.
- **ClusterMargin** [16] combines clustering with uncertainty to select each cluster's most uncertain data.
- **BADGE** [17] estimates gradients of the final-layer parameters for every sample to pick points that combine large gradient magnitudes with diverse directions.
- **Contrastive** [18] selects samples that are close in the embedding space yet exhibit the largest divergence in predicted probabilities.

- **Dataset Quantization** [19] selects data while quantizing large corpora for lossless training, optimizing efficiency and computational cost.
- **ZIP** [20] prioritizes samples exhibiting lower compression rates, capturing information-dense data.

3) Quality Assessment–based Method

- **Learning Loss** [21] trains an auxiliary predictor and ranks samples by this predicted loss, treating higher values as greater uncertainty.
- **LESS** [3] builds a gradient datastore, then measures each candidate's similarity to a handful of target examples in this gradient space; the samples whose gradients align most with the target gradients are deemed most influential.
- **STAFF** [22] employs smaller sibling models to estimate scores, verifies them with the target LLM, and allocates budget to challenging regions while maintaining coverage of simpler ones.

B. Implementation Details

DSBox is implemented in PYTHON 3.9 and built upon PYTORCH 2.1.2, benefiting from efficient tensor operations and automatic differentiation.

III. USAGE EXAMPLE

DSBox is provided as a command-line tool. Below is a basic usage example. For more detailed documentation and comprehensive usage examples, please refer to our website.

To directly obtain the indices of the selected data samples based on a specified metric and budget, the user executes:

```
python Select.py --data_dir "data_dir"\
                 --metric "metric"\
                 --budget "$budget"
```

In this command, DSBox loads the dataset located at `data_dir`, computes scores for each data sample using the selected scoring metric (`metric`), and returns the indices of the top-ranking samples according to the provided budget (`budget`).

Alternatively, to train a model on the selected subset and subsequently evaluate its downstream performance, the user can execute the following command:

```
python main.py --data_dir "data_dir" \
               --metric "metric" \
               --budget "$budget" \
               --output_dir "$outdir"
```

In this case, *DSBox* not only selects the samples as previously described but also uses these selected samples to train or fine-tune the downstream model. Upon completion, evaluation metrics such as accuracy and F1-score are computed and saved to the user-specified output directory (`outdir`).

IV. EXPERIMENTS

We employ *DSBox* to evaluate data selection methods in fine-tuning code vulnerability detection models.

TABLE II: Devign **Accuracy** under four label budgets

Method	1%	3%	5%	10%
Random	0.5582	0.5589	0.5666	0.5922
Margin	0.5414	0.5809	0.5942	0.6259
LeastConfidence	0.5348	0.5447	0.5815	**0.6479**
Learningloss	0.5548	0.5783	0.5930	0.6314
KMeans	0.5461	0.5482	0.5919	0.6266
Entropy	0.5583	0.5512	0.5823	0.6248
CoreSet	**0.5689**	**0.5988**	0.5963	0.6171
BALDDropout	0.5472	0.5677	0.5924	0.5996
ClusterMargin	0.5503	0.5410	0.5483	0.5835
BM25	0.5581	0.5663	**0.6028**	0.6102
LESS	0.5573	0.5547	0.5912	0.6225
STAFF	0.5433	0.5571	0.5787	0.6146
contrastive	0.5476	0.5480	0.5783	0.5487
BADGE	0.5460	0.5414	0.5813	0.6124
EGL	0.5487	0.5480	0.5805	0.6036
BatchBALD	0.5648	0.5465	0.5612	0.5988
Dataset Quantization	0.5401	0.5487	0.5964	0.6292
ZIP	0.5410	0.5472	0.5412	0.5993
SPUQ	0.5337	0.5644	0.5793	0.5959
UQ_ICL	0.5372	0.5592	0.5841	0.6074

TABLE III: Devign **F1-score** under four label budgets

Method	1%	3%	5%	10%
Random	0.5055	0.4194	0.4598	0.5848
Margin	0.3818	0.5397	0.5606	0.5984
LeastConfidence	0.3952	**0.5789**	0.5381	**0.6343**
Learningloss	0.5027	0.5140	0.5745	0.6138
KMeans	0.4358	0.5059	0.5362	0.6056
Entropy	0.3884	0.5518	0.5552	0.6090
CoreSet	0.3986	0.4125	0.5601	0.6098
BALDDropout	**0.5573**	0.5549	0.5575	0.5816
ClusterMargin	0.4951	0.5415	0.5193	0.5827
BM25	0.4365	0.5376	0.5751	0.5802
LESS	0.4847	0.5214	0.5541	0.6098
STAFF	0.4468	0.5210	0.5426	0.6091
contrastive	0.5307	0.5687	0.4650	0.5501
BADGE	0.4826	0.5543	0.5539	0.6011
EGL	0.4888	0.5744	0.5663	0.6085
BatchBALD	0.5627	0.5655	0.5279	0.5596
Dataset Quantization	0.4564	0.5303	**0.5836**	0.6031
ZIP	0.4099	0.4931	0.5274	0.5526
SPUQ	0.4648	0.5284	0.5227	0.5942
UQ_ICL	0.4829	0.5143	0.5241	0.5866

A. Experiment Setup

Dataset. We consider the widely used vulnerability detection dataset Devign [2] in our evaluation. *Labeling budget.* The labeling budgets are set as 1%, 3%, 5%, and 10%. *Model.* The famous code model, CodeBERT [23] is employed as the backbone model. *Evaluation metric.* We adopt Accuracy and F1-score to evaluate the performance of models trained on each selected dataset. Accuracy measures the overall correctness, while F1 better captures the trade-off between precision and recall. To ensure a fair comparison, all models are evaluated on the same held-out test set. All experiments were conducted with a batch size of 32 and a learning rate of 5e-5, optimized using AdamW. We trained each model for 10 epochs with early stopping based on validation loss. The loss function was cross-entropy, and no learning rate scheduler was applied.

B. Results

Table II and Table III summarize the results. *CoreSet* delivers strong performance under lower annotation budgets (1% and 3%) for accuracy, while *LeastConfidence* and *DatasetQuantization* achieve the highest F1-scores at larger budgets (5% and 10%). Traditional uncertainty-based methods such as *Margin* and *BALDDropout* remain competitive but exhibit less stability across budgets compared to representativeness-based strategies like *CoreSet* and *BM25*, which provide more stable performance. However, recent methods such as *LESS* and *STAFF* do not show clear advantages on this benchmark. This indicates that in the code vulnerability detection task, active learning methods are better choices than the advanced methods proposed for large language models.

Notably, random selection provides a surprisingly strong baseline in both accuracy and F1-score when the labeling budget is low (i.e., 1%), but it is consistently surpassed by stronger methods as the label budget increases. Furthermore, we consistently observe that increasing the annotation budget leads to improved model performance, aligning with expectations that more labeled data enables more robust learning.

C. Discussion

Efficiency Analysis. The computational cost of data selection in DSBox varies significantly depending on the chosen method. While simple heuristics like *Random* selection incur negligible overhead, more sophisticated, gradient-based methods such as *LESS* can become computationally prohibitive when applied to large-scale datasets. For enterprise-scale datasets, scoring the entire unlabeled pool can become a bottleneck. Practitioners could employ hybrid strategies, such as performing an initial down-selection with a fast method before applying a more computationally expensive one to the reduced candidate pool.

V. Related Work

Multiple data selection methods have been proposed for ranking training data. *LESS* [3] estimates each example's gradient influence and shows that using only 5 % of the most influential data can outperform training on the full corpus. *QuRating* [24] trains a "quality-rater" that scores documents on writing style, factual content, and educational value, yielding competitive language-model pre-training with 30 B high-rated tokens. *PreSelect* [25] retains texts whose normalised language-model loss best predicts downstream accuracy, matching a 300 B-token baseline with only 30 B tokens and 10× lower compute cost. Although these studies illuminate specific criteria (influence, quality, loss predictiveness, compressibility), their code bases are tightly coupled to single strategies, hindering rapid, head-to-head comparison. Libraries for interactive active learning, such as *small-text* [26], provide pool-based query loops but are geared toward iterative human label rather than one-shot scoring of fully-labelled corpora.

Our framework, DSBox, integrates more than twenty published scoring metrics for data selection. This design allows practitioners to easily switch strategies or budgets.

VI. Conclusion

We present *DSBox*, a lightweight yet extensible framework for label-efficient learning of code models, covering 20 data selection methods. *DSBox* is the first framework that supports both conventional active learning methods and recently proposed data selection methods designed for large language models. We believe that *DSBox* can facilitate future research in label-efficient learning domain, and provide opportunities to resource-constrained developers to contribute to the LLM era.

References

[1] T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Widyasari, I. N. B. Yusuf, H. Zhan, J. He, I. Paul *et al.*, "Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions," *CoRR*, 2024.

[2] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," *NIPS*, vol. 32, 2019.

[3] M. Xia, S. Malladi, S. Gururangan, S. Arora, and D. Chen, "Less: Selecting influential data for targeted instruction tuning," in *ICLR 2024*.

[4] Y. Li, M. Chen, Y. Liu, D. He, and Q. Xu, "An empirical study on the efficacy of deep active learning for image classification," *arXiv preprint arXiv:2212.03088*, 2022.

[5] Q. Hu, Y. Guo, M. Cordy, X. Xie, W. Ma, M. Papadakis, and Y. Le Traon, "Towards exploring the limitations of active learning: An empirical study," in *ASE*. IEEE, 2021.

[6] "deep active leanring homepage," 2022. [Online]. Available: https://github.com/ej0cl6/deep-active-learning

[7] D. Wang and Y. Shang, "A new active labeling method for deep learning," in *IJCNN*. IEEE, 2014, pp. 112–119.

[8] A. Siddhant and Z. C. Lipton, "Deep bayesian active learning for natural language processing: Results of a large-scale empirical study," in *EMNLP*, 2018, pp. 2904–2909.

[9] A. Kirsch, J. Van Amersfoort, and Y. Gal, "Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning," *NIPS*, vol. 32, 2019.

[10] B. Settles and M. Craven, "An analysis of active learning strategies for sequence labeling tasks," in *EMNLP*, 2008.

[11] X. Gao, J. Zhang, L. Mouatadid, and K. Das, "Spuq: Perturbation-based uncertainty quantification for large language models," in *EACL*, 2024, pp. 2336–2346.

[12] C. Ling, X. Zhao, X. Zhang, W. Cheng, Y. Liu, Y. Sun, M. Oishi, T. Osaki, K. Matsuda, J. Ji *et al.*, "Uncertainty quantification for in-context learning of large language models," in *ACL*, 2024, pp. 3357–3370.

[13] S. Robertson, H. Zaragoza *et al.*, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[14] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A core-set approach," in *ICLR*, 2018.

[15] F. Zhdanov, "Diverse mini-batch active learning," *arXiv preprint arXiv:1901.05954*, 2019.

[16] G. Citovsky, G. DeSalvo, C. Gentile, L. Karydas, A. Rajagopalan, A. Rostamizadeh, and S. Kumar, "Batch active learning at scale," *NIPS*, vol. 34, pp. 11 933–11 944, 2021.

[17] J. T. Ash, C. Zhang, A. Krishnamurthy, J. Langford, and A. Agarwal, "Deep batch active learning by diverse, uncertain gradient lower bounds," in *ICLR*.

[18] K. Margatina, G. Vernikos, L. Barrault, and N. Aletras, "Active learning by acquiring contrastive examples," in *EMNLP*, 2021.

[19] D. Zhou, K. Wang, J. Gu, X. Peng, D. Lian, Y. Zhang, Y. You, and J. Feng, "Dataset quantization," in *ICCV*, 2023.

[20] M. Yin, C. Wu, Y. Wang, H. Wang, W. Guo, Y. Wang, Y. Liu, R. Tang, D. Lian, and E. Chen, "Entropy law: The story behind data compression and llm performance," *arXiv preprint arXiv:2407.06645*, 2024.

[21] D. Yoo and I. S. Kweon, "Learning loss for active learning," in *CVPR*, 2019, pp. 93–102.

[22] X. Zhang, J. Zhai, S. Ma, C. Shen, T. Li, W. Jiang, and Y. Liu, "Staff: Speculative coreset selection for task-specific fine-tuning," in *ICLR*, 2025.

[23] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," in *EMNLP 2020*, 2020, pp. 1536–1547.

[24] A. Wettig, A. Gupta, S. Malik, and D. Chen, "Qurating: Selecting high-quality data for training language models," in *ICML*.

[25] K. Shum, Y. Huang, H. Zou, Q. Ding, Y. Liao, X. Chen, Q. Liu, and J. He, "Predictive data selection: The data that predicts is the data that teaches," *CoRR*, 2025.

[26] C. Schröder, L. Müller, A. Niekler, and M. Potthast, "Small-text: Active learning for text classification in python," in *EACL*, 2023, pp. 84–95.