Text2Cypher: Data Pruning using Hard Example Selection

Makbule Gulcin Ozsoy
makbule.ozsoy@neo4j.com
Neo4j
London, UK

ABSTRACT

Database query languages such as SQL for relational databases and Cypher for graph databases have been widely adopted. Recent advancements in large language models (LLMs) enable natural language interactions with databases through models like Text2SQL and Text2Cypher. Fine-tuning these models typically requires large, diverse datasets containing non-trivial examples. However, as dataset size increases, the cost of fine-tuning also rises. This makes smaller, high-quality datasets essential for reducing costs for the same or better performance. In this paper, we propose five hard-example selection techniques for pruning the Text2Cypher dataset, aiming to preserve or improve performance while reducing resource usage. Our results show that these hard-example selection approaches can halve training time and costs with minimal impact on performance, and demonstrates that hard-example selection provides a cost-effective solution.

CCS CONCEPTS

• **Information systems → Query languages for non-relational engines**; **Graph-based database models**; *Data cleaning*; • **Computing methodologies → Machine translation**; *Supervised learning*.

KEYWORDS

Hard-Example Selection, Data Selection, Text2Cypher, LLMs

ACM Reference Format:
Makbule Gulcin Ozsoy. 2018. Text2Cypher: Data Pruning using Hard Example Selection. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In today's world, data and knowledge are stored, managed, and queried through databases, which are accessed using query languages such as SQL (for relational databases) or Cypher (for graph databases). Recent advancements in large language models (LLMs) have made it possible to interact with databases using natural language, allowing models like Text2SQL and Text2Cypher to translate natural language questions into database queries. A common approach for generating these queries is to fine-tune foundational models using question-query datasets. Effective fine-tuning of these models requires large, diverse datasets with non-trivial examples.

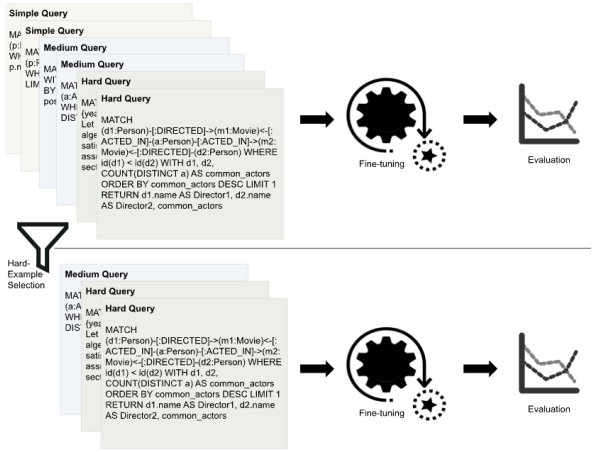


Figure 1: Hard-Example Selection for Dataset Pruning

With increased use of synthetic datasets, it is now possible to automatically generate larger datasets. However, these datasets often suffer from quality and redundancy issues. Recent research suggests that small, high-quality datasets can outperform larger ones when fine-tuning LLMs [22, 24]. Additionally, the cost of fine-tuning LLMs increases as the dataset size grows. One way to address these challenges is to prune or select a subset of the data. This process should be automated to ensure that the resulting dataset (i) maintains high performance and (ii) minimizes costs, achieving greater efficiency [8]. Figure 1 shows a hard-example selection procedure. Initially, we start with a larger dataset containing simple, medium, and hard Cypher queries used for fine-tuning a Text2Cypher model. After applying hard-example selection, the dataset is reduced in size and predominantly retains medium and hard queries.

In this paper, we apply five hard-example selection approaches to prune the Text2Cypher dataset: three approaches for selecting challenging instances from a larger training dataset to enhance model performance and two approaches that combine the proposed hard-example selection methods. We evaluate their impact on a Text2Cypher dataset, analyzing training time (in terms of training steps) and Cypher generation performance. Our main contributions are:

- We propose hard-example selection techniques specifically for the Text2Cypher task. Three approaches leverage prior analysis results and heuristics to identify challenging (hard)

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

examples and prune the training dataset, while two additional approaches combine these methods to improve performance.
- We analyze their impact on the Text2Cypher task on training time (measured in steps), loss values, and Cypher generation performance.
- Our results show that hard-example selection approaches reduce resource usage — both in elapsed time and total cost— by more than half while minimally affecting Cypher generation performance. Although there is room for improvement in matching the performance of training on the full dataset, hard-example selection presents a cost-effective solution.

The structure of the paper is as follows: Section 2 reviews related work on data subset selection and pruning, particularly for fine-tuning large language models. Section 3 details the hard-example selection approaches applied to the Text2Cypher task. Section 4 outlines our experimental setup and presents the evaluation results. Finally, Section 5 provides the conclusion.

2 RELATED WORK

Several approaches for data selection or pruning have been proposed in the literature [1, 17], ranging from the use of baseline LLM models to decide which instances to select or create embeddings [3–5, 9], to methods that rely on instance-level scores based on system indicators like diversity or difficulty. For example, Maharana et al. [10] use graph-based techniques to reduce redundancy by iteratively selecting diverse and challenging instances. Lin et al. [8] utilize influence and effort scores to prioritize influential and difficult samples for fine-tuning. Zhang et al. [22] identify diverse, difficult, and dependable data iteratively. In each iteration, they evaluate the distinctiveness, difficulty (through uncertainty-based prediction), and dependability (using an external LLM) of instances, then apply a weighted function to select a subset. Tan et al. [15] propose InfoMax, selecting samples based on informativeness and overlap between pairwise samples.

Other approaches include training a model on a small subset, then using it to prune the data. For example, Li et al. [7] fine-tune a model on a randomly sampled subset of data, then use the fine-tuned model to calculate Instruction Following Difficulty (IFD) scores for each instance. Instances with greater difficulty, based on IFD score, are selected for final fine-tuning. Xu et al. [19] focuses on differentiating informative hard samples from misleading ones in model training. In their HardPT framework, they utilize reinforcement learning and adaptive contrastive learning techniques. Azeemi et al. [2] employ cross-entropy scores to select harder instances. In their experiments they observe that selecting more difficult instances results in improved model performance. Xia et al. [18] introduce the LESS algorithm, an optimizer-aware approach for efficient data selection. It uses a warm-up training phase to generate low-dimensional gradient features, which are stored and later used by models for training. Finally, Yang et al. [20] focus on diversity-aware selection using sparse autoencoders and either greedy-sampling approach (SAE-GreedSelect) or similarity-based sampling (SAE-SimScale) approach.

Although data selection or pruning are well-studied in machine learning, their application to natural language to query language tasks, such as Text2SQL and Text2Cypher, remains largely unexplored. SE-HCL [23] applies curriculum learning to the Text2SQL task by training the model progressively, starting with easy instances and gradually moving to more difficult ones. This approach involves iterative steps that begin with simplifying the data, gradually increasing its complexity, and evaluating the difficulty of individual instances. Some Text2SQL datasets, such as Spider [21] and IndDB [11], provide difficulty labels based on SQL constructs like GROUP BY clauses and nested subqueries, where more complex constructs indicate higher difficulty. However, these difficulty annotations are primarily used for analyzing evaluation outputs rather than for data selection. In this work, we explore data pruning for the Text2Cypher task by focusing on hard-example selection based on instance difficulty.

3 HARD-EXAMPLE SELECTION FOR TEXT2CYPHER

We introduce five methods for selecting hard examples. Three of them focus on finding more challenging instances, while the other two combine these approaches to improve selection.

3.1 Selecting Challenging Instances

In our previous work [12], we have executed a comprehensive analysis of model performance on the Neo4j Text2Cypher (2024) dataset [13]. This analysis explored evaluation results from multiple perspectives, including key metrics (such as Google-Bleu and Exact Match), assigned complexity levels, and breakdowns by data source, database type, and fine-tuned model. The statistical analyses (e.g., averages, standard deviations) revealed consistent patterns of model struggle, particularly on examples from specific databases and data sources. Further error analysis highlighted that these challenges were often due to inconsistencies in ground-truth Cypher queries (such as varying use of WHERE clauses or aggregation methods), limitations of existing evaluation metrics, and underlying model weaknesses. These findings directly motivated our hard-example selection strategies, which aim to construct a more informative and targeted training subset by focusing on the most challenging instances.

In this section, we describe three approaches for selecting challenging instances from a larger training dataset to enhance model performance.

- **Complexity-Based Hard-Example Selection**: Guided by our analysis [12], we identified data sources and databases where fine-tuned models struggled most. Based on this analysis: (i) The chosen databases are three demonstration databases of Neo4j [1] [2], namely "recommendations, companies, neoflix", and (ii) The selected data-sources are: "functional_cypher", "synthetic_gemini", and "text2cypher2023_train".
  For the selection of these instances, we used a logical "OR" to include instances from either the selected databases or data sources. While this results in a diverse set of challenging instances, we observe an imbalance with many instances coming from a single data source. To address this, we performed additional sampling, limiting each group to a maximum of

[1] Neo4j Text2Cypher Crowdsourcing App: https://text2cypher.vercel.app/
[2] Neo4j Browser Demo: https://demo.neo4jlabs.com:7473/browser/

4,000 instances (the average group size). This resulted in a total of 16,173 instances, less than the half of the original training dataset, of approximately 40K instances.

- **Length-Based Hard-Example Selection**: This heuristic approach assumes that longer ground-truth Cypher queries are more challenging for a language model to generate owing to their increased complexity. Longer queries often involve multiple clauses, making them harder to replicate accurately. Therefore, this approach selects instances based on the length of the Cypher query. To ensure consistency with other selection methods, we maintained a final dataset size of 16,173 instances.
- **Cypher-Specific Hard-Example Selection**: This heuristic method focuses on the presence of Cypher-specific terms (e.g., MATCH, WHERE, RETURN), under the assumption that queries containing more such terms are more complex. Unlike the length-based approach, which prioritizes the length of queries, this method selects instances based on the count of Cypher terms, i.e., which are likely to be more complex by containing multiple clauses. To ensure fairness with other hard-instance selection methods, we restricted this dataset to 16,173 instances.

3.2 Combining Selection Methods

We combined the proposed hard-example selection approaches as follows:

- **Complexity-Based & Length-Based Hard-Example Selection**: After selecting hard examples using the Complexity-Based approach, we took an additional step to further refine the selection process. Specifically, we sorted the chosen instances in descending order based on the length of the Cypher queries. This step follows the methodology of the Length-Based approach, which assumes that longer queries tend to be more complex and, therefore, more challenging for the model to generate. By prioritizing longer queries, we made sure that the final set of hard examples was both challenging and diverse in terms of complexity.
- **Complexity-Based & Cypher-Specific Hard-Example Selection**: Similar to the previous combined approach, after selecting hard examples using the Complexity-Based approach, we ranked them by the number of Cypher-specific terms in descending order, aligning with the Cypher-Specific approach. This method emphasizes instances with more Cypher-specific terms, as these tend to be more complex and involve multiple clauses. The final subset, therefore, includes challenging instances and have a diverse set of complexities.

3.3 Baseline Approaches

We used the following baseline approaches:

- **Original Data**: This baseline uses the training data without any modifications which provides a reference point for performance comparisons.
- **Randomly-Sampled**: In this approach, we randomly sampled instances from the original data. To ensure fairness with the Complexity-Based approach, we aimed to create a balanced dataset across data source groups. We first sampled each group (based on the data-source field) to a size of 2,755, representing the 75th percentile of data source group sizes. We then refined the sample to 16,173 instances to match the size used in the hard-instance selection methods.

4 EXPERIMENTAL SETUP AND RESULTS

4.1 Experimental Setup and Evaluation Metrics

For our experiments, we used the publicly available Text2Cypher dataset [13], which contains 44,387 instances—39,554 for training and 4,833 for testing. This dataset is a cleaned and combined version of multiple data sources, most of which were synthetically generated.

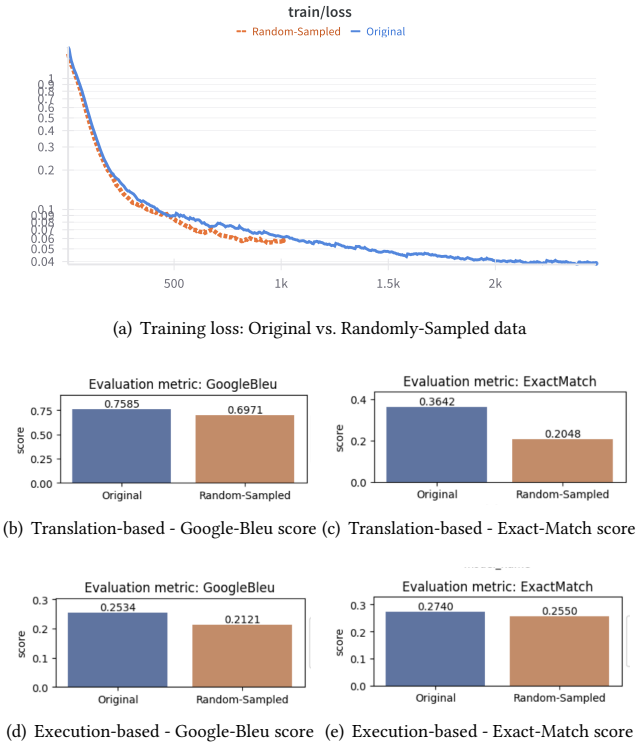
We employed two evaluation procedures to measure model performance: (i) Translation-Based (Lexical) Evaluation: This method compares generated Cypher queries with ground-truth queries at the textual level. (ii) Execution-Based Evaluation: This method executes both the generated and ground-truth Cypher queries on the target database and compares their outputs, sorted lexicographically. This approach requires an active target database, where about 50% of the dataset has such references. As a result, it evaluates only a subset of the data. To compute these evaluation metrics, we used the Hugging Face Evaluate library [6]. We report the Google-Bleu and Exact Match scores as the primary evaluation metrics.

We fine-tuned a baseline model, 'unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit', using various training datasets prepared according to the proposed hard-example selection methods. During evaluation, we used the test set and fine-tuned models to generate Cypher queries based on input natural language questions and corresponding database schemas. After generating the Cypher queries, we applied a post-processing step to remove unwanted text, such as the 'cypher:' prefix. Details of the fine-tuning setup and parameters are provided in Appendix B.

4.2 Evaluation Results

We analyzed the impact of (i) using a subset of the full dataset, assessing both training efficiency and model accuracy, and (ii) applying different hard-example selection approaches on performance.

4.2.1 Impact of Training Data Reduction. The original 40K-instance training dataset was reduced to 16,173 instances through random-sampling or hard-example selection. As shown in Figure 2, training the full dataset required around 2.5K steps (batch size 16), while the subset datasets needed only 1K steps. This reduction significantly cut fine-tuning time and costs. Using subset data achieved comparable or better training loss at 1K steps. However, over the full 2.5K steps, the original full dataset achieved a better final loss: 0.0387 versus 0.0569 for random sampling. Translation-based evaluation, which is based on token prediction accuracy, aligns closely with the loss function. The original dataset achieved a Google-Bleu score of 0.75 and an Exact Match score of 0.36, whereas the random sampling approach scored lower at 0.69 and 0.20, respectively. Execution-based evaluation showed smaller drops, with the full dataset scoring 0.25 (Google-Bleu) and 0.27 (Exact Match) versus 0.21 and 0.25 for the randomly sampled dataset. In summary, using subsets cuts training time and costs by over half but reduces

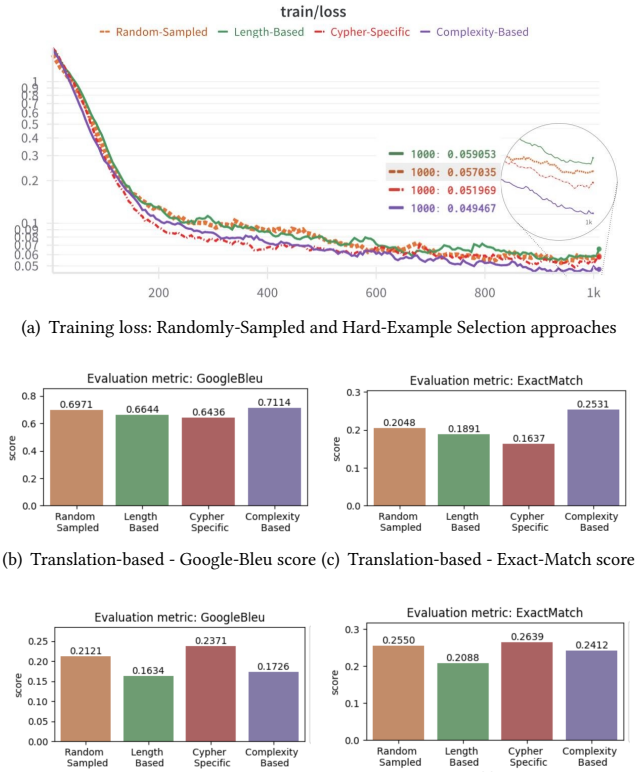


(a) Training loss: Original vs. Randomly-Sampled data

(b) Translation-based - Google-Bleu score (c) Translation-based - Exact-Match score

(d) Execution-based - Google-Bleu score (e) Execution-based - Exact-Match score

Figure 2: Original vs. Randomly-Sampled data



(a) Training loss: Randomly-Sampled and Hard-Example Selection approaches

(b) Translation-based - Google-Bleu score (c) Translation-based - Exact-Match score

(d) Execution-based - Google-Bleu score (e) Execution-based - Exact-Match score

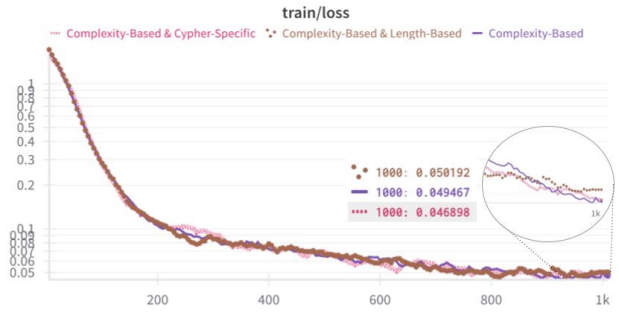
Figure 3: Randomly-Sampled and Hard-Example Selection approaches

performance. We next explore whether hard-example selection can retain efficiency while improving outcomes.

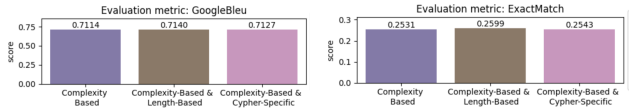
4.2.2 Impact of Hard-Example Selection. When fine-tuning the baseline model with datasets prepared using random sampling or hard-example selection approaches, training times remain similar since the dataset sizes were kept equal, as shown in Figure 3. All methods achieve comparable loss values, ranging between 0.05 and 0.06. However, closer inspection reveals a ranking from highest (worst) to lowest (best) loss: Length-Based → Random-Sampled → Cypher-Specific → Complexity-Based. In translation-based evaluation, the Complexity-Based approach performs best, achieving 0.71 Google-Bleu and 0.25 Exact Match, bringing it closer to the performance of the original dataset. Interestingly, execution-based evaluation, which is run on a subset of data that has access to active demonstration databases, follows a different pattern. In this case, the Cypher-Specific approach yields the best results, with Google-Bleu and Exact Match scores of 0.23 and 0.26, respectively.

4.2.3 Impact of Combining Approaches on Performance. Combining the Complexity-Based approach with either the Length-Based or Cypher-Specific approach did not result in significantly different loss values, as shown in Figure 4. For translation-based evaluation, all approaches performed similarly, with Google-Bleu and Exact Match scores around 0.71 and 0.25, respectively. However, execution-based evaluation revealed some variation: The best Google-Bleu score (0.24) is achieved by Complexity-Based & Length-Based approach, and the best Exact Match score (0.25) is achieved by Complexity-Based & Cypher-Specific approach. These findings suggest that although combining approaches does not drastically impact performance, some combinations may offer slight advantages depending on the evaluation method.

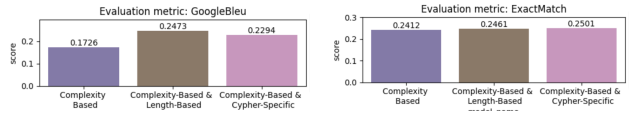
4.2.4 Overall. As shown in Table 1, while the full dataset achieves the highest Google-Bleu and Exact Match scores for both translation- and execution-based evaluation, hard-example selection outperforms random sampling. It also reduces resource usage—time and cost—by more than half, as presented in Figure 2, with minimal performance loss. We observe that fine-tuned models may still benefit from more data or better-tuned hyper-parameters, even with 16K instances. Future work will explore increasing data diversity and optimizing hyper-parameters to boost performance. Additionally, the difference between evaluation methods requires further investigation. While translation-based evaluation closely aligns with the loss function, reflecting token prediction accuracy, execution-based evaluation follows a different pattern. We attribute this behavior to the fact that execution-based evaluation is run on instances that have access to demonstration databases, which is around 50% of the dataset. In the future, we will analyze how different data subsets



(a) Training loss: Combined Hard-Example Selection approaches



(b) Translation-based - Google-Bleu score (c) Translation-based - Exact-Match score



(d) Execution-based - Google-Bleu score (e) Execution-based - Exact-Match score

Figure 4: Combined Hard-Example Selection approaches

Table 1: Performance Comparison: Original (2.5K steps) vs. Randomly-Sampled (1K steps) vs. Hard-Example Selection (best scores - 1K steps)

	Translation-Based		Execution-Based	
	Google-Bleu	Exact-Match	Google-Bleu	Exact-Match
Original	0.7585	0.3642	0.2534	0.2740
Randomly-Sampled	0.6971	0.2048	0.2121	0.2550
Hard Example Selection (best)	0.7140	0.2599	0.2473	0.2639

impact the model's ability to generate accurate Cypher queries during execution-based evaluation.

5 CONCLUSION

With models like Text2SQL and Text2Cypher, which translate natural language questions into database queries, it is now possible to interact with databases through natural language. In order to achieve this, foundational large language models (LLMs) are fine-tuned using large, diverse datasets containing non-trivial examples. However, the cost of fine-tuning these models can be significant, making it desirable to use smaller, high-quality datasets to optimize performance and resource usage. In this work, we explored hard-example selection for the Text2Cypher task, presenting five approaches to prune the training dataset. Our analysis demonstrates that selecting more complex or hard examples reduces resource usage, in terms of time and cost, by over half, while minimally affecting Cypher generation performance. This finding highlights the potential for smaller, high-quality datasets to optimize fine-tuning of large language models (LLMs), especially as a cost-effective strategy.

In this work, we focused on pruning the dataset by selecting the more complex (hard) instances. However, diversity of the data is also an important factor. In future research, we plan to explore pruning strategies that take both difficulty and diversity into account. We also aim to analyze how different subsets of the data affect the model's ability to generate accurate Cypher queries and improve the dataset based on that. While we mostly used heuristic-based methods in this study, we plan to investigate more advanced techniques in the future, such as different loss functions and training strategies, to further boost model performance.

REFERENCES

[1] Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, et al. 2024. A survey on data selection for language models. *arXiv preprint arXiv:2402.16827* (2024).

[2] Abdul Azeemi, Ihsan Qazi, and Agha Raza. 2023. Data pruning for efficient model pruning in neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 236–246.

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

[4] Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2024. AlpaGasus: Training a Better Alpaca with Fewer Data. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=FdVXgSJhvz

[5] Qianlong Du, Chengqing Zong, and Jiajun Zhang. 2023. Mods: Model-oriented data selection for instruction tuning. *arXiv preprint arXiv:2311.15653* (2023).

[6] HuggingFace. 2024. HuggingFace Evaluate. https://huggingface.co/evaluate-metric.

[7] Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2024. From Quantity to Quality: Boosting LLM Performance with Self-Guided Data Selection for Instruction Tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 7595–7628.

[8] Xinyu Lin, Wenjie Wang, Yongqi Li, Shuo Yang, Fuli Feng, Yinwei Wei, and Tat-Seng Chua. 2024. Data-efficient Fine-tuning for LLM-based Recommendation. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*. 365–374.

[9] Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. 2024. What Makes Good Data for Alignment? A Comprehensive Study of Automatic Data Selection in Instruction Tuning. In *The Twelfth International Conference on Learning Representations*.

[10] Adyasha Maharana, Prateek Yadav, and Mohit Bansal. 2024. D2 Pruning: Message Passing for Balancing Diversity & Difficulty in Data Pruning. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=thbtoAkCe9

[11] Eduardo R Nascimento, Grettel M Garcıa, Lucas Feijó, Wendy Z Victorio, Yenier T Izquierdo, Aiko R de Oliveira, GM Coelho, Melissa Lemos, RL Garcia, LAP Leme, et al. 2024. Text-to-SQL meets the real-world. In *Proceedings of the 26th international conference on enterprise information systems*, Vol. 1. 61–72.

[12] Makbule Gulcin Ozsoy. 2025. Neo4j Text2Cypher: Analyzing Model Struggles and Dataset Improvements. https://medium.com/p/0b965fd3ebfa.

[13] Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. 2025. Text2Cypher: Bridging Natural Language and Graph Databases. In *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)*. 100–108.

[14] RunPod. 2024. RunPod. https://www.runpod.io/.

[15] Haoru Tan, Sitong Wu, Wei Huang, Shizhen Zhao, and Xiaojuan Qi. 2025. Data Pruning by Information Maximization. In *The Thirteenth International Conference on Learning Representations*.

[16] Unsloth. 2024. Unsloth AI - Open Source Fine-Tuning for LLMs. https://unsloth.ai/.

[17] Jiahao Wang, Bolin Zhang, Qianlong Du, Jiajun Zhang, and Dianhui Chu. 2024. A survey on data selection for llm instruction tuning. *arXiv preprint arXiv:2402.05123* (2024).
[18] Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. 2024. LESS: Selecting Influential Data for Targeted Instruction Tuning. In *International Conference on Machine Learning (ICML)*.
[19] Yuanjian Xu, Qi An, Jiahuan Zhang, Peng Li, and Zaiqing Nie. 2023. Hard Sample Aware Prompt-Tuning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 12356–12369.
[20] Xianjun Yang, Shaoliang Nie, Lijuan Liu, Suchin Gururangan, Ujjwal Karn, Rui Hou, Madian Khabsa, and Yuning Mao. 2025. Diversity-driven data selection for language model tuning through sparse autoencoder. *arXiv preprint arXiv:2502.14050* (2025).
[21] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 3911–3921.
[22] Jia Zhang, Chen-Xi Zhang, Yao Liu, Yi-Xuan Jin, Xiao-Wen Yang, Bo Zheng, Yi Liu, and Lan-Zhe Guo. 2025. D3: Diversity, Difficulty, and Dependability-Aware Data Selection for Sample-Efficient LLM Instruction Tuning. *arXiv preprint arXiv:2503.11441* (2025).
[23] Yiyun Zhang, Sheng'an Zhou, and Gengsheng Huang. 2024. Se-hcl: Schema enhanced hybrid curriculum learning for multi-turn text-to-sql. *IEEE Access* 12 (2024), 39902–39912.
[24] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2023. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems* 36 (2023), 55006–55021.

A DECLARATION ON GENERATIVE AI USAGE

During the preparation of this work, the author(s) used Chat-GPT in order to: 'Improve writing style' and 'Paraphrase and reword'. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

B FINE-TUNING PARAMETERS

For fine-tuning, we used a RunPod [14] GPU environment with a single A40 machine. The fine-tuning process was conducted using the Unsloth[16] framework. The parameters used for fine-tuning are presented in Table 2.

Table 2: Fine-tuning Parameters

Model & Tokenizer Parameters	max_seq_length : 2048, $dtype$: $torch.bfloat16$, $load\_in\_4bit$: $True$, $truncation\_side$: $"left"$, $padding\_side$: $"left"$
PEFT Parameters	r : 8, $target\_modules$: $["q\_proj", "k\_proj", "v\_proj", "o\_proj"]$, $lora\_alpha$: 16, $lora\_dropout$: 0, $bias$: $"none"$, $random\_state$: 3407, use_rslora : $False$, $loftq\_config$: $None$
Training Arguments	$per\_device\_train\_batch\_size$: 2, $gradient\_accumulation\_steps$: 4, $warmup\_steps$: 5, num_train_epochs : 1, $learning\_rate$: $2e-4$, $fp16$: $notis\_bfloat16\_supported()$, $bf16$: $is\_bfloat16\_supported()$, $optim$: $"adamw\_8bit"$, $weight\_decay$: 0.01, $lr\_scheduler\_type$: $"linear"$, $seed$: 3407