GENEPRUNE : AUTOMATED PRUNING OF LARGE LANGUAGE MODELS FOR CODE USING GENETIC ALGORITHM

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) for code generation exhibit remarkable capabilities but face deployment challenges due to their high computational and memory demands. Traditional pruning methods, often based on static heuristics like magnitude-based weight pruning, fail to effectively balance sparsity and performance, particularly for structured tasks such as code generation. To address this, we propose GenePrune, a novel genetic algorithm-based pruning framework that optimizes pruning masks for pre-trained Code LLMs without requiring costly retraining. GenePrune iteratively refines pruning configurations through evolutionary operations such as crossover and mutation, guided by a fitness function that balances model sparsity and task-specific performance. Experiments on open-source models like CodeT5 demonstrate that GenePrune achieves superior pruning efficiency, significantly reducing model size while maintaining high BLEU scores for code generation tasks. Our results highlight GenePrune as a promising approach for efficient LLM compression, with potential applications in optimizing inference speed and deployment in resource-constrained environments.

1 INTRODUCTION

Large Language Models (LLMs) have revolutionized code generation, with models like Codex (Chen et al., 2021) and CodeT5 (Wang et al., 2021) demonstrating strong capabilities in generating, completing, and refactoring code. However, these models are highly resource-intensive, requiring substantial memory and computational power (Bommasani et al., 2022), making deployment in real-world, latency-sensitive environments challenging. While model pruning offers a potential solution, existing approaches often degrade performance, particularly in maintaining syntactic and semantic accuracy in code generation tasks (Zafrir et al., 2021).

Recent post-training pruning techniques in LLMs, such as weight-magnitude pruning (Sun et al., 2024), structured pruning (Ma et al., 2023), and gradient-based pruning (Das et al., 2024), have demonstrated varying levels of success. SparseGPT (Frantar & Alistarh, 2023) achieves up to 50% sparsity while retaining generation quality, while Wanda (Sun et al., 2024) improves pruning efficiency using weight-aware neuron dynamics. However, these methods are limited to general tasks and rely on heuristic-driven rules that assume certain weight properties universally correlate with unimportance (Gale et al., 2019). This assumption often results in suboptimal pruning decisions, particularly for code generation, where preserving structural integrity and program logic is crucial.

To address these limitations, we propose a novel genetic algorithm-based pruning technique, **GenePrune**, which introduces a data-driven and adaptive approach to model pruning for code generation LLMs. GenePrune deviates from traditional rule-based methods by employing a genetic algorithm (GA) to explore a wide search space of potential pruning configurations, allowing the model to dynamically adapt its pruning strategy across different layers and submodules. Using evolutionary operations, such as crossover and mutation, GenePrune iteratively generates and evaluates candidate pruning masks based on feedback from task-specific performance metrics. This enables the system to optimize for sparsity while maintaining high performance, effectively discovering pruning patterns that are better suited to code generation tasks. The adaptive nature of GenePrune allows it to

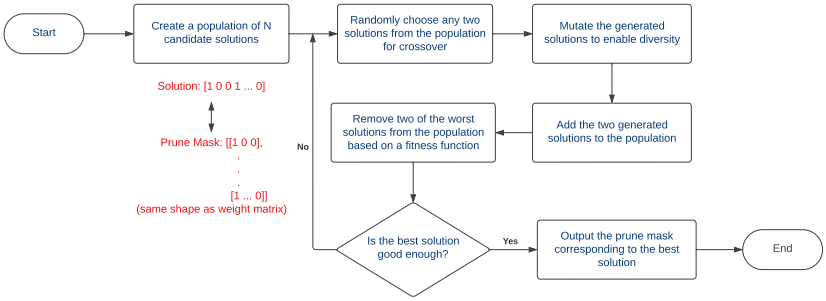

Figure 1: Flowchart of the algorithm proposed for pruning LLM using genetic algorithm.

tailor its pruning strategy to the unique requirements of each model layer, thereby preserving critical task-specific parameters and enhancing generalization across models and tasks.

Our work offers the following contributions:

- We explore use of pruning, specifically for Code LLMs, focusing on code generation. To the best of our knowledge, this is the first work to study model compression in this context.
- We introduce GenePrune, the first framework to utilize genetic algorithms for optimizing pruning masks in LLMs, enabling data-driven, adaptive pruning strategies that are specifically tailored for code generation.
- We demonstrate GenePrune's ability to maintain high performance while significantly reducing computational overhead across multiple Code LLMs.

2 METHODOLOGY

2.1 GENETIC ALGORITHM FOR LAYER-WISE PRUNING

We propose a genetic algorithm-based pruning framework that iteratively refines pruning masks for each layer, optimizing sparsity while preserving model performance. By evolving populations of pruning solutions, the algorithm adaptively selects optimal pruning configurations without requiring expensive retraining.

Population: A population of binary pruning masks is initialized, where each element represents the presence (1) or absence (0) of a connection in the layer. Masks are structured according to the layer's dimensions and initialized based on a predefined sparsity ratio, which dictates the probability of weight removal while ensuring a balance between compression and accuracy.

Selection: Each iteration selects two parent masks uniformly at random for crossover. The population is progressively refined by retaining only the top k solutions based on fitness scores, ensuring convergence toward optimal pruning patterns over multiple generations.

Crossover: Single-point crossover is applied to parent masks, swapping segments beyond a randomly chosen crossover point. This operation generates offspring that inherit structural properties from both parents while introducing variations, promoting an effective exploration of the pruning space. The crossover rate controls how frequently new solutions are generated, maintaining a balance between exploration and stability.

Mutation: To prevent premature convergence and local optima, mutation is applied with a small probability, flipping certain elements to zero to enforce sparsity. This stochastic alteration enhances diversity, allowing the algorithm to explore alternative pruning configurations and refine weight selection dynamically.

Fitness Function: Pruning masks are evaluated using a weighted fitness function:

$$f = \lambda s + (1 - \lambda)p \tag{1}$$

where s represents the sparsity induced by the mask, p denotes model performance on a validation set, and λ, a hyperparameter, 'governs the trade-off between compression and accuracy. A well-tuned λ ensures a balance between reducing computational cost and maintaining task-specific performance.

The genetic algorithm iteratively refines pruning masks by selecting, recombining, and mutating candidates, progressively optimizing the trade-off between sparsity and performance. A detailed flowchart of the pruning process is provided in the Figure 1.

2.2 Multi-Layer Pruning

To extend pruning across multiple layers, the genetic algorithm operates independently on each layer, adapting sparsity to structural and functional differences. This sequential approach prevents abrupt performance degradation while maintaining stability across layers. The cumulative effect progressively reduces model complexity while preserving accuracy.

Fine-Tuning for Post-Pruning Recovery. After pruning, model performance may degrade due to the loss of critical connections. To mitigate this, we apply fine-tuning to recalibrate active weights while maintaining the pruned architecture. The optimal pruning masks are applied during retraining, ensuring that pruned connections remain inactive. The model is fine-tuned using a small learning rate, enabling gradual weight adjustments without destabilization. This adaptation allows the model to compensate for lost connections, restoring performance while retaining a significantly lighter architecture.

3 Experiments

Datasets. We use two datasets: one for fine-tuning Code LLMs during pruning and another for evaluation. For fine-tuning, we use the Re-sampled Python3.7 API Knowledge dataset (Xu et al., 2020), which integrates API documentation and StackOverflow code snippets to enhance code generation models by capturing diverse Python API usage patterns. For evaluation, we use the MBPP dataset (Austin et al., 2021), a benchmark consisting of 974 Python programming problems, designed to assess functional correctness and code generation accuracy.

Models. We fine-tune and prune CodeT5 (Wang et al., 2021), an open-source model pre-trained on CodeSearchNet (Husain et al., 2020). It is further fine-tuned on conala-mined, Re-sampled Python3.7 API Knowledge, and MBPP datasets.

Metrics. We evaluate pruned models using BLEU. BLEU measures syntactic accuracy by computing n-gram overlap (1-gram to 4-gram) and their average in generated code.

Hyperparameter	Value
λ	0.02
Population Size	10
Number of Generations	5
Initial Sparsity	5%
Mutation Rate	5%
Crossover Rate	100%
Sparsity Threshold	20% - 50%

Table 1: Hyperparameters for the Genetic Algorithm

Procedure. We prune CodeT5 using a genetic algorithm, exploring different sparsity-accuracy trade-offs. Specifically, we experiment with $\lambda \in \{0.02, 0.03, 0.05\}$ to balance accuracy and sparsity, while varying population sizes (5 and 10) to analyze convergence speed versus exploration. We test 2 and 3 generations to assess the impact of evolutionary depth and apply pruning to 9 selected layers, comparing their effects on sparsity and performance. Additionally, we explore the impact of fine-tuning the pruned models using two different datasets, one in-domain and one out-of-domain, to analyze its effect on performance recovery. We observe that fine-tuning with an out-of-domain dataset helps retain performance better, suggesting that the model benefits from additional exposure to diverse patterns beyond the original training distribution. Based on experimental results provided

Sparsity Level	GenePrune		Magnitude-based Pruning	
	W/O Fine-tuning	W/ Fine-tuning	W/O Fine-tuning	W/ Fine-tuning
20%	0.1520	0.1512	0.1348	0.1348
30%	0.1420	0.1575	0.1249	0.1366
40%	0.1158	0.1340	0.1258	0.1308
50%	0.1399	0.1523	0.1255	0.1309

Table 2: Comparison of GenePrune and Magnitude-based Pruning on MBPP dataset using Avg BLEU score at different Sparsity Levels for 9 layers

Sparsity Level	GenePrune	Random Mask	
	W/O Fine-tuning	W/O Fine-tuning	W/ Fine-tuning
20%	0.1520	0.1275	0.1296
30%	0.1420	0.1218	0.1275
40%	0.1158	0.0957	0.0987
50%	0.1499	0.0964	0.1119

Table 3: Ablation study showing the effectiveness of GenePrune over randomly generated masks for 9 layers at different sparsity levels using BLEU score as the performance metric. The BLEU score of the original model is 0.1389.

in the appendix A.1), we determine optimal hyperparameters (Table 1) and evaluate GenePrune against the baseline at different sparsity levels. Due to resource constraints, we restrict pruning to a single block of 9 layers, including dense, attention, and normalization layers.

Baselines. We compare GenePrune against Magnitude-based pruning (Han et al., 2016), which removes weights based on magnitude. Currently, we have implemented and compared Magnitude-based pruning, with plans to evaluate on other pruning methods of general LLMs.

4 Results

Table 2 presents the results of our controlled pruning experiments, comparing **GenePrune** with the baseline **Magnitude-based Pruning** on nine pruned layers. GenePrune consistently outperforms the baseline across all sparsity levels in terms of Avg BLEU Score, particularly when fine-tuning is applied. Even without fine-tuning, it maintains competitive performance, surpassing the baseline in most cases. These results highlight GenePrune's adaptive pruning strategy, which effectively redistributes remaining weights to preserve task-specific performance. Fine-tuning further enhances performance, emphasizing the importance of weight redistribution in mitigating the impact of pruned parameters.

Table 3 provides an ablation study comparing GenePrune to randomly generated pruning masks over nine layers. GenePrune demonstrates superior performance at all sparsity levels, reinforcing that effective weight selection is nontrivial and that its genetic algorithm successfully preserves crucial model parameters. These results confirm GenePrune's effectiveness in maintaining model performance while significantly reducing computational complexity, making it a strong candidate for efficient, deployable pruning in code LLMs.

5 Conclusion and Future Work

We introduced GenePrune, a novel genetic algorithm-based pruning framework for optimizing sparsity in code generation LLMs while preserving accuracy. Our method dynamically searches for optimal pruning masks through evolutionary operations, outperforming magnitude-based pruning in controlled experiments across multiple sparsity levels, as evidenced by BLEU score improvements. Ablation studies confirmed the effectiveness of our approach over random masks, while fine-tuning further enhanced performance recovery. In future work, we aim to extend GenePrune to larger code models, integrate reinforcement learning to refine the fitness function dynamically, and explore pruning techniques that optimize inference speed and energy efficiency for real-time code generation applications.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2022. URL https://arxiv.org/abs/2108.07258.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Rocktim Jyoti Das, Mingjie Sun, Liqun Ma, and Zhiqiang Shen. Beyond size: How gradients shape pruning decisions in large language models, 2024. URL https://arxiv.org/abs/2311.04902.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be pruned in a single shot. In *Proceedings of the 2023 International Conference on Machine Learning*, 2023.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks, 2019. URL https://arxiv.org/abs/1902.09574.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016. URL https://arxiv.org/abs/1510.00149.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search, 2020. URL https://arxiv.org/abs/1909.09436.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, 2023.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models, 2024. URL https://arxiv.org/abs/2306.11695.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation, 2021. URL https://arxiv.org/abs/2109.00859.

Frank F. Xu, Zhengbao Jiang, Pengcheng Yin, Bogdan Vasilescu, and Graham Neubig. Incorporating external knowledge through pre-training for natural language to code generation, 2020. URL https://arxiv.org/abs/2004.09015.

Ofir Zafrir, Ariel Larey, Guy Boudoukh, Haihao Shen, and Moshe Wasserblat. Prune once for all: Sparse pre-trained language models, 2021. URL https://arxiv.org/abs/2111.05754.

A Appendix

A.1 Hyperparameter Tuning

Table 4 presents a study on GenePrune's configuration, varying key hyperparameters such as sparsity-accuracy coefficient (λ), population size, pruned layers, and datasets. Higher λ values increase sparsity but at a cost to BLEU scores, with $\lambda = 0.05$ reducing BLEU to 0.049 while sparsity rises to 0.28. Selective layer pruning mitigates accuracy degradation, as aggressive pruning across all layers results in substantial performance loss. A larger population size stabilizes BLEU scores while maintaining sparsity, suggesting that broader search exploration benefits pruning effectiveness.

Using the Python-API dataset during pruning leads to higher stability, achieving a BLEU score of 0.124 at $\lambda = 0.03$, outperforming MBPP. The larger dataset prevents overfitting, allowing better generalization. These findings indicate that moderate λ values (0.02–0.03), a larger population size, and selective layer pruning yield the best balance between sparsity and performance. Excessive sparsity levels degrade accuracy, underscoring the importance of structured pruning strategies in maintaining code generation quality.

Generation	λ **(Sparsity-Accuracy)**	**Population Size**	**Layers Pruned**	**BLEU Score**	**Sparsity**
MBPP used while Pruning					
2	0.02	5	80	0.1155	0.174
2	0.03	5	80	0.0883	0.20
2	0.02	10	80	0.115	0.19
2	0.05	10	80	0.0490	0.28
2	0.02	5	All	0.0135	0.29
3	0.02	5	80	0.1017	0.207
3	0.02	5	All	0.0883	0.28
PythonAPI used while Pruning					
2	0.03	10	80	0.124	0.208
3	0.02	10	80	0.1117	0.207

Table 4: Ablation study of different settings of the GenePrune method. We present the Average BLEU score on the MBPP Dataset and the Sparsity % achieved.