

DOI: https://doi.org/10.48009/2_iis_2026_144

Performance and efficiency trade-offs in fine-tuned and pruned large language models

Jongho Seol, *Middle Georgia State University*, and jongho.seol@mga.edu

Woohyun Kang, *Brainworks Co., Ltd.*, and austin@brainworks.co.kr

Myungjae Kwak, *Middle Georgia State University*, and myungjae.kwak@mga.edu

Michael Do, *Georgia Institute of Technology*, and michaeldo1026@gmail.com

Abstract

Large Language Models (LLMs) achieve strong performance in natural language processing and code-generation tasks, but their high computational and memory requirements limit deployment in resource-constrained environments. Although compression techniques such as pruning can improve efficiency, their effects are often evaluated mainly through average performance metrics, which may overlook reliability degradation and stochastic variability. This study presents a reliability-aware empirical framework for analyzing LLM compression by jointly evaluating performance, stability, and efficiency across different structured pruning levels. A LLaMA-based model was fine-tuned on the CodeAlpaca dataset and then pruned at 10%, 20%, 50%, and 75% pruning ratios, followed by recovery fine-tuning. Model performance was evaluated using Pass@k on the HumanEval benchmark (Hugging Face Hub Data Set), with CodeBLEU and perplexity used as supplementary measures of code quality and language-modeling behavior. The results show that LLM compression exhibits a nonlinear degradation pattern. Performance remains close to the fine-tuned baseline at low and moderate pruning levels, particularly at or below 20%, but deteriorates sharply near 50% pruning, where Pass@k approaches zero and perplexity increases substantially. These findings identify three operational regimes: a stable region at or below 20% pruning, a transition region between 20% and 50%, and a collapse region at or above 50%. Theoretically, this suggests that compressed LLM behavior cannot be adequately explained by average performance metrics alone and should instead be analyzed through reliability-aware evaluation that accounts for nonlinear degradation and variability across runs. Practically, the identified pruning thresholds provide guidance for selecting pruning ratios that preserve functional performance while offering potential efficiency gains and avoiding unstable or unreliable deployment behavior.

Keywords: Large Language Models, Model Compression, Structured Pruning, Fine-Tuning, Pass@k, CodeBLEU, Perplexity

Introduction

Large Language Models (LLMs) have significantly advanced natural language processing by enabling high-quality text generation, reasoning, summarization, and code synthesis. Despite their strong performance, these models require substantial computational power, memory capacity, and storage resources, making deployment difficult in cost-sensitive and resource-constrained environments. As LLM architectures continue to grow in scale and complexity, efficient compression and optimization techniques have become increasingly important for practical deployment and inference. Moreover, prior works on LLM compression (Frantar & Alistarh, 2023; Zhang et al., 2024) treats compression-induced degradation

as gradual, without characterizing whether and where it accelerates, which limits the deployment guidance that can be derived from average performance metrics alone.

This study addresses these gaps through a reliability-aware empirical analysis of structured pruning for code-generation LLMs. The central finding is that compression-induced degradation is not gradual but nonlinear and regime-dependent. Performance and reliability remain near baseline up to a moderate pruning level, but collapse sharply once a critical threshold is crossed. Based on this behavior, three distinct operational regimes are identified: stable ($\leq 20\%$ pruning), transition (20–50%), and collapse ($\geq 50\%$). Each regime exhibits a different performance, reliability, and efficiency profile.

Fine-tuning, pruning, and quantization are among the most widely used approaches for improving LLM efficiency (Dettmers et al., 2023; Zhao et al., 2025). Fine-tuning adapts pretrained models to domain-specific tasks and improves task-specific performance. Pruning reduces model size and computational overhead by removing less important parameters or structures (Zhang et al., 2024), while quantization lowers numerical precision to decrease memory usage and accelerate inference (Egashira et al., 2024). These techniques can substantially reduce computational requirements while preserving acceptable performance levels. However, aggressive compression may lead to severe degradation in generation quality, reasoning capability, and overall model accuracy.

Most prior studies evaluate compressed LLMs primarily using deterministic metrics such as accuracy, perplexity, or benchmark scores (Xu et al., 2024; Gu et al., 2024). While these metrics are useful, they often fail to fully characterize the practical trade-offs between model performance and computational efficiency. In particular, there remains a need for empirical analysis that identifies compression levels capable of preserving acceptable performance while significantly reducing resource consumption. Understanding these trade-offs is essential for deploying LLMs in environments with limited hardware availability, memory constraints, or latency requirements.

This study presents a performance and efficiency trade-off analysis of fine-tuned and pruned large language models using structured compression strategies. Experiments are conducted using a LLaMA-based model fine-tuned on the CodeAlpaca dataset obtained from the Hugging Face Hub (Hugging Face Hub Data Set). Model performance is evaluated across multiple pruning ratios using Pass@k, CodeBLEU, and perplexity metrics to analyze the effects of compression on code generation quality and language modeling capability. The study further incorporates recovery fine-tuning after pruning to examine the extent to which performance degradation can be mitigated.

The primary objective of this work is to identify compression configurations that achieve an effective balance between model performance and computational efficiency. Experimental results demonstrate that low and moderate pruning levels preserve most baseline capabilities, whereas aggressive pruning causes substantial degradation in the benchmark metrics performance. These findings provide practical guidance for selecting efficient compression strategies for real-world LLM deployment scenarios.

The main contributions of this study are as follows:

- An empirical demonstration that structured-pruning degradation in code-generation LLMs is nonlinear and threshold-based rather than gradual, evidenced consistently across Pass@k, CodeBLEU, and perplexity.
- A reliability-aware evaluation framework that evaluates both average performance and repeated-experiment consistency to identify when pruning begins to make the model unstable.
- A regime-based interpretation (Stable, Transition, Collapse) that gives practitioners concrete guidance for selecting pruning levels that balance efficiency and robustness.

The objective of this study is to investigate the performance–efficiency trade-offs associated with compressing large language models through fine-tuning and pruning techniques. In particular, the study evaluates how compression influences code-generation capability, model reliability, and deployment feasibility under varying sparsity levels. To achieve this objective, the study addresses the following central research question:

How do structured pruning and recovery fine-tuning affect the performance and efficiency of fine-tuned large language models for code-generation tasks across different compression levels?

The remainder of the paper is organized as follows. Section 2 reviews related work on LLM fine-tuning, pruning, and reliability-aware evaluation. Section 3 describes the methodology. Section 4 presents the experimental results and the regime-based interpretation. Section 5 discusses the implications and limitations. Section 6 concludes the paper and outlines future research directions.

Related Work

Large Language Models (LLMs) have become a major advancement in natural language processing due to their ability to perform a wide range of tasks, including text generation, reasoning, summarization, and code generation. Built on transformer architectures and trained on large-scale corpora, these models capture complex linguistic patterns and task-relevant structures. Despite their strong capabilities, LLMs require substantial computational and memory resources, making efficient adaptation and deployment a critical research challenge, particularly in cost-sensitive and resource-constrained environments.

LLM Fine-Tuning

Fine-tuning is a standard approach for adapting a pretrained LLM to a domain-specific task. Rather than training from scratch, fine-tuning leverages pretrained knowledge and updates the model using task-specific data, thereby improving performance on specialized applications such as code generation, instruction following, and domain-aware question answering. In this study, fine-tuning is used to adapt a LLaMA-based model for code generation using CodeAlpaca-style instruction data and Hugging Face tools. While full fine-tuning can yield strong task adaptation, it is computationally expensive and memory intensive. As a result, parameter-efficient fine-tuning (PEFT) methods such as LoRA (Li et al., 2023; Xia et al., 2024) have gained attention because they update only a small subset of parameters while keeping most of the base model fixed. Instruction tuning further improves alignment by organizing examples as instruction–response pairs, which is especially useful for code-oriented and conversational tasks. In practical settings, PEFT provides a favorable balance between performance and efficiency and is often more feasible than full fine-tuning in limited compute environments.

Model Compression Through Pruning

Pruning is one of the most widely used model compression strategies for reducing parameter count and computational cost. Its basic objective is to remove redundant or less important weights, neurons, or structures while preserving task performance as much as possible. Pruning can be applied at different granularities. Unstructured pruning removes individual weights, typically based on magnitude or contribution, and can achieve high sparsity, although the resulting irregular parameter pattern does not always translate into hardware acceleration (Frantar & Alistarh, 2023). Structured pruning removes larger components such as neurons, channels, attention heads, or layers, and is generally more hardware friendly and more suitable for real deployment scenarios. In LLMs, pruning presents an important trade-off between compression and performance retention: light or moderate pruning may reduce cost with minimal

degradation, whereas aggressive pruning can sharply reduce model quality once sparsity exceeds a stable threshold.

Recovery Fine-Tuning After Pruning

After pruning, models often experience performance degradation because important weights or structures may have been removed. Recovery fine-tuning is commonly used to restore lost capability by retraining the pruned model for a smaller number of steps (Yang et al., 2025). This step is particularly important in LLM compression because pruning can alter learned parameter distributions and reduce task-specific alignment. In the present workflow, pruning is followed by recovery fine-tuning in an iterative manner, reflecting a practical compression strategy in which performance loss is partially repaired after each pruning stage. This recovery stage is especially relevant for code generation tasks, where even modest losses in reasoning, syntax consistency, or structural coherence can substantially reduce output quality.

Quantization

Quantization is another important compression technique that reduces memory usage and inference cost by lowering numerical precision (Dettmers et al., 2023). Instead of storing model weights in higher-precision floating-point formats, quantized models use lower-bit representations such as INT8 or 4-bit formats. This can significantly reduce storage requirements and improve inference efficiency when supported by appropriate hardware. Post-training quantization is especially attractive because it can be applied after fine-tuning and pruning without retraining the full model. However, lower-bit representations may introduce approximation error and reduce numerical stability. In practice, moderate quantization such as INT8 is often safer, while more aggressive 4-bit quantization may offer greater efficiency at the expense of accuracy and robustness (Egashira et al., 2024). Accordingly, quantization is best understood not as an isolated compression step, but as part of a broader performance–efficiency trade-off in compressed LLM systems.

Performance Evaluation of Compressed LLMs

Evaluating compressed LLMs requires more than a single performance measure. For code generation tasks, multiple metrics capture different aspects of output quality. CodeBLEU measures similarity and code-related structure, perplexity measures predictive confidence (Xu et al., 2024), and Pass@k estimates the probability of generating at least one correct output across multiple attempts (Chen et al., 2025). Together, these metrics provide a more complete view of model behavior under compression, complementing broader practices of multi-signal LLM evaluation (Gu et al., 2024). Prior observations and the results of this study indicate that light pruning may produce only small degradation, whereas aggressive pruning causes sharp increases in perplexity and severe drops in Pass@k and CodeBLEU. This pattern suggests that compressed LLMs may appear acceptable under moderate sparsity yet deteriorate rapidly beyond a critical compression range.

Reliability and Stability in Compression

Most prior compression studies focus on average performance, model size, or latency. However, reliability is also critical, especially in deployment settings where consistent output quality is required (Naik, 2024; Schroeder & Wood-Doughty, 2024). In this study, reliability is defined as the stability of model performance across repeated runs, prompts, or experimental conditions, and is quantified through variance-based analysis. This perspective is important because stochastic gradient descent, data sampling, and reduced model capacity can introduce variability (Ozkara et al., 2025) that is not captured by average metrics alone. A compressed model with slightly lower average performance but consistent behavior may be more suitable for deployment than one with better average performance but unstable outputs. This motivates the treatment of reliability as a core analytical dimension rather than a secondary consideration.

Cost-Benefit Perspective

Beyond technical performance, compression must also be evaluated in terms of practical trade-offs. In real deployment scenarios, decisions depend on the balance between computational savings and performance loss. A configuration that reduces memory and latency while preserving most of the original capability may be desirable, whereas a highly compressed model with unacceptable degradation may not be useful despite large cost reductions. This study adopts an explicit cost–benefit perspective by comparing pruning levels and identifying moderate compression as a practically favorable trade-off between efficiency and performance. Such a viewpoint is especially relevant in high-stakes or resource-limited settings where both efficiency and reliability are required.

Research Gap

Although prior work has examined fine-tuning, pruning, and quantization separately, fewer studies consider their combined effects from the integrated perspective of performance, reliability, and deployment trade-offs. In particular, reliability is often underemphasized relative to average accuracy or loss, and compression behavior is frequently evaluated empirically without an accompanying analytical model. This creates a need for a more unified framework that can explain not only whether compression improves efficiency, but also whether the resulting model remains stable and practically usable, including the cost trade-offs of replacing larger models with compressed ones in deployment (Chen et al., 2023; Irugalbandara et al., 2024).

Methodology

This study employs an empirical evaluation framework to analyze the effects of compression on large language models designed for code-generation tasks. The methodology examines how varying compression strategies and pruning intensities influence functional accuracy, semantic code quality, robustness, and computational efficiency.

Data and Models

All datasets used in this study are publicly available: HumanEval is accessed via OpenAI’s public repository, and CodeAlpaca is obtained from the Hugging Face Hub. The HumanEval benchmark datasets (Hugging Face Hub Data Set) used in this study were obtained directly from the Hugging Face Hub to support supervised fine-tuning, recovery training, and evaluation of compressed large language models. For supervised fine-tuning, the CodeAlpaca dataset, publicly available via the Hugging Face Hub (Hugging Face Hub Data Set), was utilized to adapt the pretrained LLaMA-based model for code-generation tasks. The dataset was tokenized using the Alpaca instruction format, which organizes each sample into instruction, input, and response components. These formatted examples were subsequently rendered using the Llama-3.2 chat template to ensure compatibility with conversational instruction-tuning workflows.

To support recovery fine-tuning after pruning, the pruned models were retrained using 500 training samples and 100 validation samples extracted from the CodeAlpaca dataset. The same tokenization and formatting procedure used during supervised fine-tuning was maintained throughout the recovery stage to ensure consistency across all experimental configurations.

For performance evaluation, OpenAI’s HumanEval dataset was used to compute Pass@k metrics. HumanEval consists of 164 Python programming problems with corresponding reference solutions and is widely used for evaluating functional correctness in code-generation models. In addition, perplexity evaluation was conducted using 50 samples from the CodeParrot-clean dataset (Codeparrot-clean Data Set), a publicly available large-scale dataset of Python source files designed for training and evaluating code-

generation systems. These datasets collectively enabled comprehensive evaluation of functional accuracy, structural code quality, and language modeling capability under varying compression levels.

Compression Pipeline

This study presents an empirical framework for evaluating the performance and efficiency trade-offs of fine-tuned and pruned large language models in code-generation tasks. The experimental workflow integrates supervised fine-tuning, structured pruning, recovery fine-tuning, quantization, and multi-metric evaluation within a unified compression pipeline. Structured pruning ratios of 10%, 20%, 50%, and 75% were investigated to analyze how increasing compression severity affects model behavior, computational efficiency, and code-generation quality.

Figure 1 illustrates the overall experimental compression workflow used in this study. The process sequentially applies supervised fine-tuning, structured pruning, and recovery fine-tuning to a pretrained LLaMA-based model, followed by evaluation using multiple code-generation benchmarks and performance metrics.

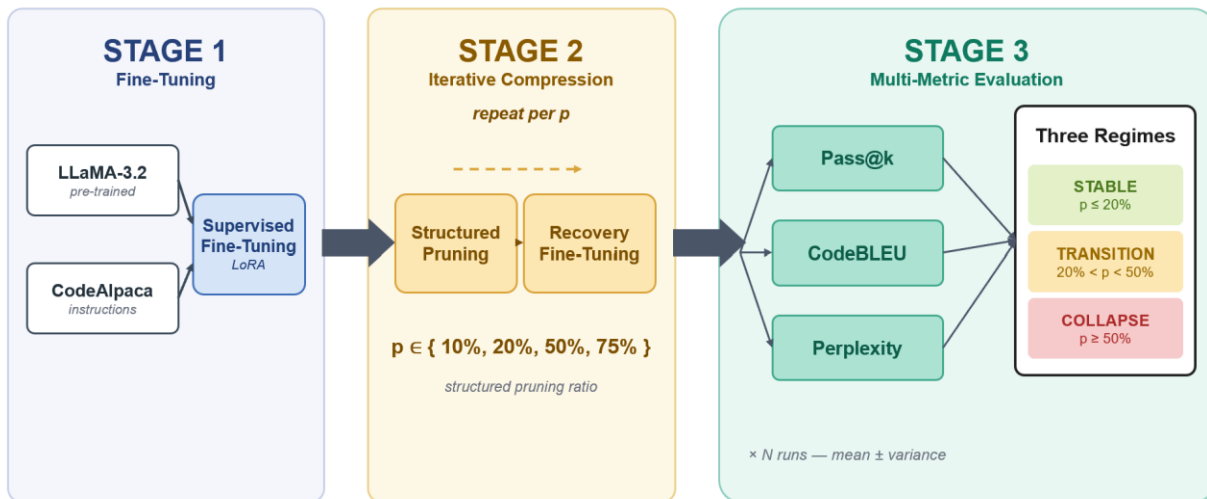


Figure 1. Experimental LLM Compression and Evaluation Framework

As shown in Figure 1, the experimental framework consists of three sequential stages that together form a reliability-aware compression pipeline. The first stage establishes a fine-tuned baseline by combining the pretrained LLaMA-3.2 model with the CodeAlpaca instruction dataset through supervised fine-tuning. The resulting model serves as the unpruned reference against which all subsequently compressed models are compared.

The second stage applies compression to the fine-tuned baseline. Structured pruning is performed at four pruning ratios $p \in \{10\%, 20\%, 50\%, 75\%\}$ and recovery fine-tuning is applied after each pruning step to partially restore the performance lost during pruning. This stage yields four compressed model checkpoints, each retaining a different proportion of the original parameters, in addition to the unpruned baseline.

The third stage evaluates each model checkpoint using three complementary metrics: Pass@k for functional correctness, CodeBLEU for structural and semantic code quality, and perplexity for language-modeling capability. Each configuration is evaluated multiple times under stochastic decoding so that both mean performance and run-to-run variance can be recorded. The resulting performance summaries are then

aggregated across pruning levels to characterize three operational regimes that are summarized at the right of the figure (stable, transition, and collapse) and analyzed in detail in Section 4.

Evaluation Metrics

Model performance was evaluated using three complementary metrics, Pass@k, CodeBLEU, and perplexity to capture different aspects of compressed model behavior. These metrics were selected because code-generation performance cannot be fully assessed using a single score. Pass@k measures functional correctness, CodeBLEU captures structural and semantic code quality, and perplexity reflects language-modeling capability. Together, these metrics provide task-level, code-quality-level, and token-prediction-level evidence for evaluating how pruning affects model behavior.

- Pass@k measures functional correctness in code-generation tasks. It estimates the probability that at least one of the top k generated solutions passes the corresponding unit tests. A higher Pass@k value indicates that the model is more likely to generate functionally correct solutions across multiple attempts. In this study, Pass@5, Pass@10, and Pass@20 were computed using the HumanEval benchmark to evaluate whether the compressed models could generate correct Python solutions across multiple attempts. A higher Pass@k value indicates stronger functional correctness and greater likelihood of producing a valid solution.
- CodeBLEU evaluates the quality of generated code by extending traditional BLEU with code-specific information. It considers lexical similarity, syntax structure, and semantic/code-flow features between generated code and reference solutions. A higher CodeBLEU score indicates stronger structural and semantic similarity to the reference code. In this study, CodeBLEU was used as a supplementary metric to assess whether pruning degraded the structural and semantic quality of generated code, even when outputs were not fully captured by functional correctness alone. A higher CodeBLEU score indicates stronger similarity to the reference solution in both form and code-related meaning.
- Perplexity measures the model's language-modeling capability by quantifying how well the model predicts a sequence of tokens. Lower perplexity indicates that the model assigns higher probability to the evaluated code sequence, suggesting better predictive confidence and token-prediction performance. In this study, perplexity was used to examine whether pruning disrupted the model's underlying token-prediction ability, particularly under aggressive compression levels.

Together, these metrics provide a multi-dimensional assessment of performance degradation resulting from different pruning levels.

Reliability Aware Evaluation

To ensure robust evaluation, each pruning configuration was tested across multiple experimental runs under stochastic decoding conditions. This repeated-run design was necessary because stochastic decoding introduces run-to-run variability. Without accounting for this variability, observed differences across pruning regimes could be artifacts of randomness rather than genuine compression effects. Therefore, the repeated-run evaluation was used to verify whether the threshold-based degradation pattern remained robust to sampling variance. Performance metrics were aggregated, and descriptive measures, including mean and variance, were computed to assess consistency and reliability. Variance across runs was used as an indicator of model stability under different compression levels.

The methodology further incorporates comparative analysis across pruning regimes to identify nonlinear degradation patterns and critical compression thresholds. This approach enables simultaneous evaluation of accuracy, robustness, and computational cost, providing a comprehensive framework for understanding the practical limits and deployment feasibility of compressed large language models.

Implementation

All training, pruning, recovery fine-tuning, and evaluation procedures were conducted in Google Colab using Python 3. The experiments utilized the high-RAM NVIDIA A100 GPU runtime, which provided 167.1 GB of system RAM, 80.0 GB of GPU memory, and 235.7 GB of available disk storage.

The software environment consisted of publicly available Python libraries, including Hugging Face Transformers, Accelerate, Datasets, PEFT, TRL, and BitsAndBytes. Supervised fine-tuning employed the nightly build of the Unsloth library to enable memory-efficient LoRA-based training. Evaluation procedures utilized the HumanEval framework, CodeBLEU library, and Tree-sitter library for code-quality and syntax analysis.

All model checkpoints and LoRA adapters were hosted on the Hugging Face Hub. Authentication and model access were managed securely through Hugging Face access tokens stored using Google Colab's userdata secrets manager.

Results and Analysis

This section presents the empirical evaluation of compressed large language models and analyzes the impact of pruning on code-generation performance, reliability, and computational efficiency. The results are organized into three components: performance evaluation, reliability assessment, and regime-based interpretation. Together, these analyses provide a comprehensive understanding of how different compression levels influence model behavior and deployment feasibility.

Pass@k Evaluation

Table 1 demonstrates that low and moderate pruning preserve strong code-generation capability, whereas aggressive pruning causes substantial performance collapse. At 50% pruning, Pass@k values decrease dramatically, and at 75% pruning the model fails to generate functionally correct outputs.

Table 1. Experimental configuration matrix for evaluating pruning strategies utilizing Pass@k Evaluations

Model	Purpose	Pass@5	Pass@10	Pass@20
Fine-Tuned	Performance baseline	58.94%	68.26%	76.83%
10% Pruning	Low compression	55.79%	64.67%	71.95%
20% Pruning	Moderate compression	57.52%	67.10%	75.00%
50% Pruning	High compression	2.17%	3.52%	4.88%
75% Pruning	Highest compression	0.0%	0.0%	0.0%

Figure 2 illustrates the impact of pruning on model performance using Pass@5, Pass@10, and Pass@20 metrics. Performance remains relatively stable under low (10%) and moderate (20%) pruning levels, but degrades sharply at higher ratios. At 50% pruning, all metrics exhibit significant performance collapse, and at 75% pruning, the model fails completely, with performance approaching zero.

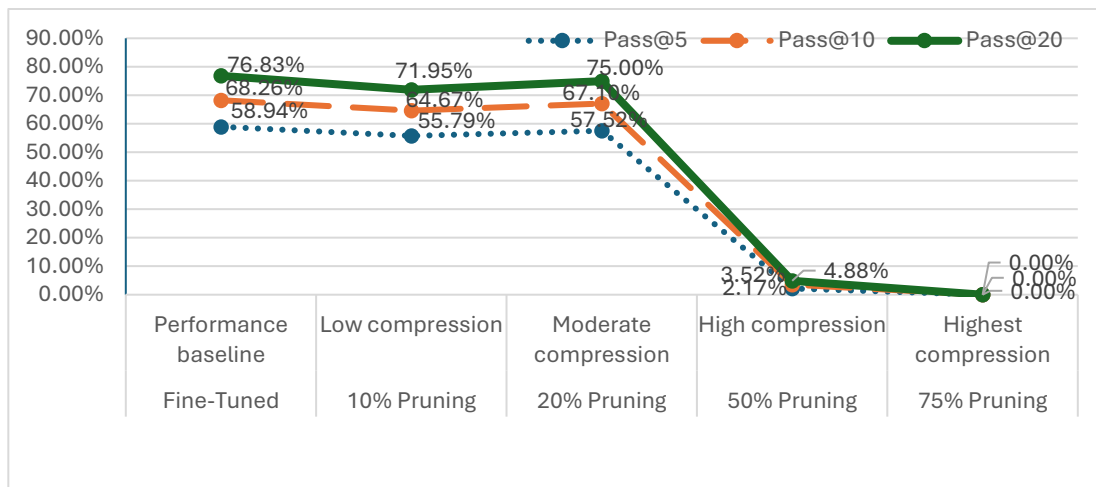


Figure 2. Pass@k Performance vs. Pruning Ratio

CodeBLEU Evaluation

CodeBLEU exhibits the same regime pattern observed for Pass@k. As shown in Table 2, the CodeBLEU score decreases only slightly from 0.3401 at 10% pruning and 0.3205 at 20% pruning, indicating that low and moderate pruning preserve much of the structural and semantic quality of generated code. However, the score declines substantially under more aggressive pruning, dropping to 0.1802 at 50% and 0.0834 at 75%. These results suggest that higher pruning ratios significantly reduce code quality and coherence.

Table 2. CodeBLEU evaluation of pruned models compared to the fine-tuned model using the HumanEval benchmark

Model	CodeBLEU Score
10% Pruned	0.3401
20% Pruned	0.3205
50% Pruned	0.1802
75% Pruned	0.0834

Figure 3 visualizes the CodeBLEU trend across pruning ratios. The score remains relatively stable between 10% and 20% pruning but declines sharply at higher pruning levels. This pattern indicates that aggressive pruning substantially degrades the structural, syntactic, and semantic quality of generated code. The sharp decrease after 20% pruning is consistent with the threshold-based degradation pattern observed in the Pass@k and perplexity results.

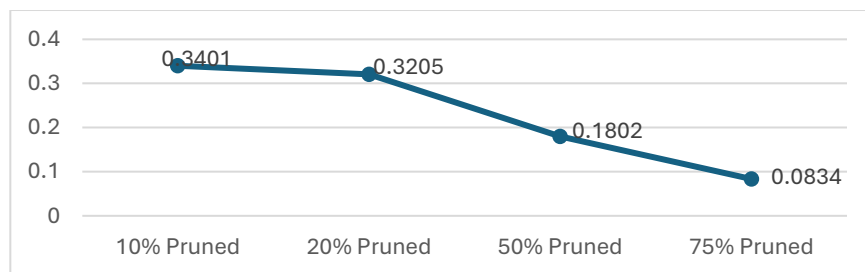


Figure 3. CodeBLEU vs. Pruning Ratio

Perplexity Evaluation

Table 3 summarizes the perplexity results across the fine-tuned baseline and each pruning configuration. Perplexity remains relatively stable under low and moderate pruning, increasing from 2.6600 for the fine-tuned baseline to 2.7435 at 10% pruning and 2.9064 at 20% pruning. These changes correspond to increases of 3.14% and 9.27%, respectively, indicating that the model largely preserves its language-modeling capability under moderate compression. However, perplexity increases sharply under aggressive pruning, rising to 16.7751 at 50% pruning and 13,324.6973 at 75% pruning. These values represent increases of 530.65% and 500,832.37% relative to the fine-tuned baseline, showing severe degradation in predictive confidence and language-modeling behavior.

Table 3. Experimental configuration matrix for evaluating pruning strategies utilizing Perplexity Evaluations

Model	Perplexity	Change in Perplexity vs Fine-tuned Model	% Change in Perplexity vs Fine-tuned Model
Fine-Tuned	2.6600	+0.0000	+0.00%
10% Pruning	2.7435	+0.0835	+3.14%
20% Pruning	2.9064	+0.2464	+9.27%
50% Pruning	16.7751	+14.1152	+530.65%
75% Pruning	13,324.6973	+13,322.0374	+500,832.37%

Figure 4 visualizes the perplexity trend across pruning ratios. The figure shows a relatively flat pattern from the fine-tuned baseline through 20% pruning, followed by a sharp increase at 50% pruning and an extreme increase at 75% pruning. This trend supports the threshold-based interpretation observed in the Pass@k and CodeBLEU results. Specifically, the model remains stable under low and moderate pruning but loses predictive confidence rapidly once pruning becomes aggressive.

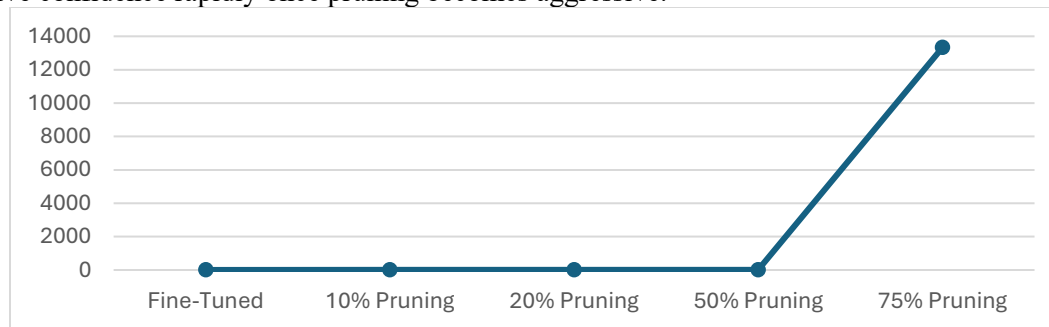


Figure 4. Perplexity vs. Pruning Ratio

Regime-Based Interpretation

Based on the empirical findings presented in Sections 4.1–4.3, the compression process can be interpreted as consisting of three distinct operational regimes. Table 4 summarizes this regime-based classification by integrating performance quality, reliability, and computational efficiency across pruning levels.

The stable regime ($\leq 20\%$ pruning) maintains high functional performance, low variance, and strong reliability while suggesting possible reductions in memory usage and computational cost. Models operating within this region remain practical for deployment and preserve most of their code-generation capability.

The transition regime (20–50% pruning) is characterized by accelerated performance degradation and increasing instability. Within this region, small increases in pruning intensity produce disproportionately larger declines in Pass@k and CodeBLEU performance, accompanied by rapidly rising perplexity and variance. These behaviors indicate growing structural sensitivity and reduced robustness under compression.

The collapse regime ($\geq 50\%$ pruning) exhibits near-zero task performance, extremely high variability, and practical infeasibility despite substantial computational savings. Models within this regime fail to maintain coherent language representations or reliable code-generation behavior, rendering them unsuitable for deployment in real-world applications.

Overall, the regime-based interpretation highlights the strongly nonlinear nature of large language model compression and identifies a critical threshold beyond which performance and reliability degrade disproportionately. The results provide empirical evidence for distinct operating regions and offer practical guidance for selecting compression strategies that effectively balance accuracy, robustness, and computational efficiency.

Table 4. Integrated Performance-Efficiency Trade-off Across Pruning Levels

Pruning Regime	Pruning Range	Performance (Pass@k / CodeBLEU)	Model Stability	Cost Efficiency
Stable (Low Compression)	$\leq 20\%$	High, near baseline	Consistent performance	High efficient
Transition (Moderate Compression)	20–50%	Noticeable degradation	Reduced consistency	Moderate to high efficiency
Aggressive Compression	$\geq 50\%$	Severe performance degradation	Unstable outputs	Very high efficiency but limited usability

Discussion

The experimental results demonstrate that large language model compression exhibits strongly nonlinear and regime-dependent behavior. Performance remains relatively stable under low (10%) and moderate (20%) pruning levels, where these evaluation measures show only minor deviations from the fine-tuned baseline, indicating that moderate compression can preserve functional code-generation capability while improving computational efficiency. However, once pruning exceeds moderate sparsity levels ($\geq 50\%$), performance deteriorates rapidly across all evaluation metrics, with sharp declines in functional correctness and semantic quality accompanied by substantial increases in perplexity and output variance.

These findings indicate that compression effects are not gradual but instead follow a threshold-based degradation pattern in which small increases in sparsity beyond a critical point lead to disproportionately large reductions in accuracy, reliability, and model stability. The results therefore support the existence of distinct operational compression regimes and establish a practical boundary between safe and unstable compression configurations, highlighting that aggressive pruning introduces structural instability that significantly limits deployment feasibility despite potential computational savings.

Practical Implications

For practitioners and industry LLM developers, the proposed regime structure offers a concrete guideline for compression-aware LLM deployment. In the stable regime, defined as pruning at or below 20%, compression can be applied with relatively low risk to model functionality, making it suitable for achieving potential efficiency gains through reduced deployment cost and resource consumption. As shown in Table 1, at 20% pruning, the model achieved a Pass@20 score of 75.00%, compared with 76.83% for the fine-tuned baseline. This corresponds to 97.6% baseline performance retention, calculated as $(75.00/76.83) \times 100$. The result demonstrates a clear practical advantage within the stable regime by

preserving functional correctness while offering potential efficiency gains through reduced model size. However, the transition regime, between 20% and 50% pruning, is less practical because model reliability becomes increasingly uncertain. Compression in this range requires deployment-specific evaluation rather than reliance on aggregate benchmark performance alone. The collapse regime, at 50% pruning or higher, should generally be avoided, as the computational savings are outweighed by severe instability and accuracy degradation.

Reliability Protocol and Practitioner Guidance

In this study, reliability is not treated as a single universal property of an LLM, but as an operational evaluation construct for compressed model deployment. Specifically, reliability refers to the extent to which a pruned LLM preserves functional correctness, output stability, and benchmark-level performance after compression. This definition is consistent with current LLM evaluation practice, where model trustworthiness is assessed using multiple performance dimensions rather than a single aggregate score. Accordingly, our reliability protocol evaluates pruning-induced degradation through task performance, benchmark accuracy, and stability across pruning regimes. The protocol is intended to provide practitioners with deployment-oriented guidance by identifying pruning ranges that remain functionally stable, require additional validation, or should be avoided.

Comparison with Prior Work

The nonlinear and threshold-based degradation observed in this experiment is related to prior studies on LLM pruning. Previous studies showed that structured pruning and magnitude-based pruning can reduce model size without serious quality loss when the pruning level is moderate. Our results show a similar pattern. The model performance does not drop significantly in the lower pruning range, but the degradation becomes much sharper as the sparsity level increases (Frantar & Alistarh, 2023). Also, our results are consistent with prior work showing that recovery fine-tuning has limited ability to restore performance when the model is pruned too aggressively (Yang et al., 2025).

This pattern is also consistent with several representative LLM pruning studies. SparseGPT showed that large language models can often tolerate substantial one-shot sparsity, including around 50% sparsity, with relatively limited degradation in some settings, suggesting that a meaningful portion of model weights can be removed without immediately destroying performance (Frantar & Alistarh, 2023). Wanda similarly demonstrated that pruning can be performed without retraining by using weight magnitude and activation information, further supporting the view that moderate compression can preserve model capability (Sun et al., 2024). However, these studies also indicate that pruning effectiveness depends on model architecture, pruning strategy, task type, and sparsity level. Our results follow the same general direction at lower pruning levels, but they show a much sharper collapse near 50% pruning in the code-generation setting. This difference may reflect the greater sensitivity of functional code generation, where small syntax or logic errors can cause a generated solution to fail completely. In addition, LLM-Pruner showed that structured pruning can reduce LLM size while preserving performance under moderate compression, but that performance degradation becomes more pronounced as pruning becomes more aggressive (Ma et al., 2023). Taken together, these prior findings support the general compression pattern observed in this study: moderate pruning can be feasible, whereas aggressive pruning may cross a threshold beyond which recovery becomes difficult and model behavior becomes unstable.

While prior work has reported threshold-like degradation behavior in general LLM pruning tasks, this study extends that finding to code-generation specifically and introduces a reliability-aware evaluation framework that captures not only mean performance degradation but also variance-based instability. This distinction is important because many prior pruning studies primarily report average benchmark scores, whereas our results show that pruning can also affect run-to-run consistency and output stability. A compressed model

may appear acceptable based on average performance alone, yet still behave unpredictably across repeated runs. From a practical deployment perspective, pruning should therefore be evaluated not only by accuracy or benchmark scores, but also by how stable and dependable the model remains after compression, especially for code-generation tasks where small output errors can cause functional failure.

Limitations

This study has several limitations. First, the experiment used only one model architecture, LLaMA-3.2, which limits the generalizability of the findings to other LLM families, model sizes, and architectural designs. Different models may respond differently to structured pruning because of differences in parameter distribution, layer sensitivity, attention structure, and training history. Therefore, the stable, transition, and collapse regimes identified in this study should be interpreted as empirical findings for the tested model rather than universal pruning thresholds for all LLMs.

Second, the evaluation focused on a single task domain: code generation. Although code generation is a useful setting for studying compression because functional correctness can be evaluated through benchmark-based testing, it may be more sensitive to pruning than some natural language tasks. Small syntax, logic, or formatting errors can cause a generated program to fail completely, even when the output appears partially correct. As a result, the sharp degradation observed near higher pruning levels may partly reflect the stricter correctness requirements of code-generation tasks. Future studies should examine whether similar threshold-based degradation patterns appear in other domains such as summarization, question answering, reasoning, and dialogue generation.

Third, the pruning ratios were sampled at only four levels: 10%, 20%, 50%, and 75%. This sparse sampling provides evidence of a broad degradation pattern but does not identify the exact critical threshold at which performance begins to collapse. The results suggest that the most important transition occurs between 20% and 50% pruning, but additional experiments using finer-grained pruning intervals are needed to locate the threshold more precisely. For example, testing smaller increments within the 20%–50% range would provide a clearer understanding of when degradation accelerates and whether the transition is gradual or abrupt.

Fourth, the cost-efficiency discussion in this study is primarily qualitative. Although pruning is expected to reduce computational and memory requirements, this study did not directly measure inference latency, peak memory usage, throughput, energy consumption, or FLOPs. Therefore, the exact computational benefit of each pruning configuration was not fully quantified. Future work should include hardware-level efficiency measurements to more precisely evaluate the trade-off between performance loss, reliability degradation, and practical deployment cost.

Future Work

Future work should address these limitations in five directions. First, future studies should conduct a finer-grained pruning sweep, especially between 20% and 50%, to identify the critical degradation threshold more accurately. For example, pruning experiments using smaller intervals, such as 1% increments, would help determine the point at which performance degradation begins to accelerate and whether the transition from stable behavior to collapse is gradual or abrupt.

Second, the proposed evaluation framework should be tested across other model families and model sizes. This study focused on LLaMA-3.2, but other architectures may respond differently to pruning because of differences in parameter distribution, layer organization, training data, and alignment procedures. Evaluating models from different LLM families would help determine whether the stable, transition, and collapse regimes observed in this study are architecture-specific or more broadly generalizable.

Third, future research should extend the analysis beyond code generation to other task domains. Code generation is highly sensitive to syntax and functional correctness, so pruning-induced degradation may appear more sharply than in some natural language tasks. Applying the same reliability-aware framework to summarization, question answering, reasoning, dialogue generation, and long-context tasks would clarify whether threshold-based degradation also occurs across broader LLM applications.

Fourth, future studies should examine pruning together with quantization in a unified compression framework. In practical deployment, pruning and quantization are often combined to reduce memory usage, inference cost, and hardware requirements. Evaluating their combined effects would provide a more realistic understanding of compression trade-offs, including whether quantization amplifies, mitigates, or interacts with pruning-induced degradation and instability.

Fifth, future work should investigate layer-adaptive and structure-aware pruning strategies. Different layers, attention heads, feed-forward modules, and transformer components may have different levels of sensitivity to pruning. Rather than applying a uniform pruning ratio across the model, adaptive methods could assign different pruning levels based on layer importance, activation behavior, or reliability impact. Such approaches may help preserve functional performance and output stability while achieving stronger compression.

Conclusions

This study investigated the effects of structured pruning on large language models through a reliability-aware empirical evaluation framework for code-generation tasks. Using a fine-tuned LLaMA-3.2 model and the HumanEval benchmark, model behavior was primarily evaluated using the performance metrics serving as supplementary measures of code quality and language-modeling capability. The experimental results demonstrate that compression-induced degradation is strongly nonlinear and highly dependent on pruning level.

The findings show that performance remains relatively stable under low and moderate pruning levels ($\leq 20\%$), where models retain functional code-generation capability while achieving reductions in computational cost and memory requirements. In contrast, performance deteriorates rapidly beyond a critical sparsity threshold near 50% pruning, where Pass@k values approach zero and reliability declines substantially. Supporting CodeBLEU and perplexity analyses further confirm significant degradation in semantic quality and predictive capability under aggressive compression. These results reveal three distinct operational regimes—stable, transition, and collapse—highlighting the existence of compression thresholds beyond which performance and reliability degrade disproportionately.

The results further indicate that recovery fine-tuning can partially mitigate compression-induced degradation at lower pruning levels but becomes increasingly ineffective as sparsity increases. Overall, the study demonstrates that LLM compression should be treated as a reliability-sensitive and nonlinear process rather than a purely deterministic optimization problem. The proposed framework provides practical guidance for identifying pruning configurations that balance functional correctness, computational efficiency, and deployment feasibility.

Future work will extend the framework to additional LLM architectures, datasets, and application domains to improve the generalizability of the findings. Further research is also needed to investigate structure-aware pruning techniques that account for layer sensitivity, attention mechanisms, and parameter

interactions within transformer architectures. In addition, future studies should incorporate broader evaluation dimensions, including robustness, calibration, reasoning capability, human evaluation, and long-context performance. Developing adaptive reliability-aware compression strategies capable of identifying critical sparsity thresholds before catastrophic performance degradation occurs remains an important direction for the efficient deployment of compressed large language models.

Acknowledgements

The authors gratefully acknowledge William Blocker and Bryan Nguyen (Houston County High School, Warner Robins, GA, USA) for their contributions to data collection and dataset preparation during this research.

References

CodeAlpaca Data Set:

<https://huggingface.co/datasets/sahil2801/CodeAlpaca-20k>

Codeparrot-clean Data Set :

<https://huggingface.co/datasets/codeparrot/codeparrot-clean>

Chen, Z., Qin, X., Wu, Y., Ling, Y., Ye, Q., Zhao, W. X., & Shi, G. (2025). Pass@ k training for adaptively balancing exploration and exploitation of large reasoning models. *arXiv preprint arXiv:2508.10751*.

Chen, L., Zaharia, M., & Zou, J. (2023). Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*.

Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36, 10088-10115.

Egashira, K., Vero, M., Staab, R., He, J., & Vechev, M. (2024). Exploiting llm quantization. *Advances in Neural Information Processing Systems*, 37, 41709-41732.

Frantar, E., & Alistarh, D. (2023, July). Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning* (pp. 10323-10337). PMLR.

Gu, J., Jiang, X., Shi, Z., Tan, H., Zhai, X., Xu, C., ... & Guo, J. (2024). A survey on llm-as-a-judge. *The Innovation*.

Hugging Face Hub Data Set:

https://huggingface.co/datasets/openai/openai_humaneval/viewer/openai_humaneval/test?p=1&row=100

Irugalbandara, C., Mahendra, A., Daynauth, R., Arachchige, T. K., Dantanarayana, J., Flautner, K., ... & Mars, J. (2024, May). Scaling down to scale up: A cost-benefit analysis of replacing OpenAI's

- LLM with open source SLMs in production. In *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (pp. 280-291). IEEE.
- Li, Z., Li, X., Liu, Y., Xie, H., Li, J., Wang, F. L., ... & Zhong, X. (2023). Label supervised llama finetuning. *arXiv preprint arXiv:2310.01208*.
- Ma, X., Fang, G., & Wang, X. (2023). LLM-Pruner: On the structural pruning of large language models. *Advances in Neural Information Processing Systems*.
- Naik, N. (2024). Probabilistic consensus through ensemble validation: A framework for llm reliability. *arXiv preprint arXiv:2411.06535*.
- Ozkara, K., Yu, T., & Park, Y. (2025). Stochastic rounding for llm training: Theory and practice. *arXiv preprint arXiv:2502.20566*.
- Schroeder, K., & Wood-Doughty, Z. (2024). Can you trust llm judgments? reliability of llm-as-a-judge. *arXiv preprint arXiv:2412.12509*.
- Sun, M., Liu, Z., Bair, A., & Kolter, J. Z. (2024). A simple and effective pruning approach for large language models. *International Conference on Learning Representations*.
- Xia, Y., Fu, F., Zhang, W., Jiang, J., & Cui, B. (2024). Efficient multi-task llm quantization and serving for multiple lora adapters. *Advances in Neural Information Processing Systems*, 37, 63686-63714.
- Xu, J., Zhang, H., Yang, Y., Yang, L., Cheng, Z., Lyu, J., ... & Chiew, T. K. (2024). One size does not fit all: Investigating efficacy of perplexity in detecting llm-generated code. *ACM Transactions on Software Engineering and Methodology*.
- Yang, K., Tao, G., Chen, X., & Xu, J. (2025, May). Alleviating the fear of losing alignment in llm fine-tuning. In *2025 IEEE Symposium on Security and Privacy (SP)* (pp. 2152-2170). IEEE.
- Zhang, N., Liu, Y., Zhao, X., Cheng, W., Bao, R., Zhang, R., ... & Chen, H. (2024, June). Pruning as a domain-specific llm extractor. In *Findings of the Association for Computational Linguistics: NAACL 2024* (pp. 1417-1428).
- Zhao, W., Zou, L., Wang, Z., Yao, X., & Yu, B. (2025). HAPE: Hardware-Aware LLM Pruning For Efficient On-Device Inference Optimization. *ACM Transactions on Design Automation of Electronic Systems*, 30(4), 1-18.