

# Smaller = Weaker? Benchmarking Robustness of Quantized LLMs in Code Generation

Sen Fang, Weiyuan Ding, Antonio Mastropaolo and Bowen Xu

**Abstract**—Quantization has emerged as a mainstream method for compressing Large Language Models (LLMs), reducing memory requirements and accelerating inference without architectural modifications. While existing research primarily focuses on evaluating the effectiveness of quantized LLMs compared to their original counterparts, the impact on robustness remains largely unexplored. In this paper, we present the first systematic investigation of how quantization affects the robustness of LLMs in code generation tasks. Through extensive experiments across four prominent LLM families (LLaMA, DeepSeek, CodeGen, and StarCoder) with parameter scales ranging from 350M to 33B, we evaluate robustness from dual perspectives: adversarial attacks on input prompts and noise perturbations on model architecture. Our findings challenge conventional wisdom by demonstrating that quantized LLMs often exhibit superior robustness compared to their full-precision counterparts, with 51.59% versus 42.86% of our adversarial experiments showing better resilience in quantized LLMs. Similarly, our noise perturbation experiments also confirm that LLMs after quantitation generally withstand higher levels of weight disturbances. These results suggest that quantization not only reduces computational requirements but can actually enhance LLMs’ reliability in code generation tasks, providing valuable insights for developing more robust and efficient LLM deployment strategies.

## I. INTRODUCTION

Quantization [1, 2, 3] has gained significant attention as an effective method for compressing large language models (LLMs). This technique is advantageous not only for general-purpose LLMs but also for optimizing models designed for specialized tasks, such as large code models (LCMs)<sup>1</sup>. In particular, LCMs employ distinct optimization strategies to enhance the automation of software engineering (SE) tasks such as bug fixing, source code generation and code review, among others [4, 5].

The increasing demand for compressed LLMs has led to a surge in publicly available quantized models, many of which are hosted on platforms like Hugging Face. Notably, nearly all state-of-the-art LLMs now have quantized versions, enabling more efficient deployment without significant performance loss. Examples include DeepSeek [6] and LLaMA [7], both of which have been successfully quantized and made publicly accessible for broader use.

Unlike knowledge distillation [8, 9], another model compression technique, which requires extensive retraining, quantization directly converts high-precision weights to lower-precision representations, reducing memory footprint and ac-

celerating inference without architectural modifications. Moreover, it leverages widespread hardware support for low-bit operations, enabling efficient deployment on edge devices [10]. Its non-parametric nature preserves model structure while offering flexible precision-performance trade-offs through various granularities and bit widths, making it particularly effective for LLMs [11, 12].

Existing studies pay more attention to evaluate the effectiveness of quantized LLMs compared to their original versions. For example, [13] compared the effectiveness of the original and quantized model in code summarization task. [14] studied multiple parameter-efficient fine-tuning techniques and verified that LLMs fine-tuned with QLoRA [11] could achieve competitive performance compared with full fine-tuning. Overall, many existing works [15, 16, 12, 17, 18] demonstrate that the quantized LLMs can achieve performance comparable to their original versions in terms of effectiveness.

However, from the analysis we conducted, a question naturally emerges: *Does “free lunch” really exist?* We look into this question through the lens of another critical property of LLMs: *i.e.*, robustness. As illustrated in the Figure 1, we apply a sentence-level adversarial attack [15] to generate an adversarial prompt for the code generation task. The adversarial attack translates the original English prompt into German and then back to English, creating a semantically equivalent prompt with minor differences—specifically, replacing the word “given” with “specified”. When we presented both prompts to DeepSeek-Coder-6.7B-base<sup>2</sup>, we observed an intriguing phenomenon. While the model pre- and post-quantization both correctly handle the original prompt, their behaviors diverge significantly on the adversarial prompt. The original model misinterprets the adversarial prompt entirely, generating a solution that counts the length of a given list rather than counting the number of list elements in the given list. In contrast, the quantized model maintains consistent and correct behavior across both prompts, generating identical solutions that properly count the number of lists. Moreover, same situation is also observed in LLaMA models. Our observation is counter-intuitive as quantization is typically viewed as a compression technique that trades off model performance for efficiency, actually enhances model robustness – raising an intriguing question about the relationship between quantization and LLMs’ behavior.

Hence, motivated by our observation, we aim to answer the following key question,

S. Fang, W. Ding, and B. Xu are with NC State University, USA. Antonio Mastropaolo is with William & Mary, USA.

E-mail: {sfang9, wding8, bxu22}@ncsu.edu, amastropaolo@wm.edu

<sup>1</sup>In this paper, we use LLMs and LCMs interchangeably

<sup>2</sup><https://huggingface.co/deepseek-ai/deepseek-coder-6.7b-base>

**Original prompt:**

```
"""
Write a python function to count the number of lists in a given number of
lists.
assert count_list([[1, 3], [5, 7], [9, 11], [13, 15, 17]]) == 4
"""


```

**Adversarial prompt:**

```
"""
Write a python function to count the number of lists in a specified number
of lists.
assert count_list([[1, 3], [5, 7], [9, 11], [13, 15, 17]]) == 4
"""


```

Listing 1. The prompt from MBPP<sup>+</sup> No. 792 task. We perform sentence level adversarial attack by translating the prompt into German then translating back.

```
def count_list ( lst ):
    count = 0
    for i in lst :
        if isinstance ( i, list ):
            count += 1
    return count
```

Listing 2. Output of original model with original prompt.



```
def count_list ( lst ):
    count = 0
    for i in lst :
        if type(i) == list :
            count += 1
    return count
```

Listing 4. Output of quantized model with original prompt.



```
def count_list ( lst ):
    return len ( lst )
```

Listing 3. Output of original model with adversarial prompt.

```
def count_list ( lst ):
    count = 0
    for i in lst :
        if type(i) == list :
            count += 1
    return count
```

Listing 5. Output of quantized model with adversarial prompt.



Figure 1. The generated solutions for MBPP<sup>+</sup> No. 792 task by DeepSeek-Coder-6.7B pre- and post-quantization with prompts before and after attack.

*How does quantization impact LLMs’ robustness in code generation task?*

We specifically focus on code generation for several compelling reasons [19]. First, code generation serves as a rigorous benchmark, requiring both precise logical reasoning and strict syntactic correctness—even minor errors can render the output completely non-functional. Second, the widespread adoption of AI-assisted programming tools [19, 20] in software development workflows has made code generation one of the most commercially significant applications of LLMs. This practical importance is reflected in the fact that coding ability has become a standard evaluation criterion for newly released LLMs [21, 22], which are routinely assessed on important benchmarks like HumanEval [23] and MBPP [24]. Third, the robustness of code generators has already been a thread of research [25, 15, 26, 27, 28, 29], highlighting the significance of this challenge and its broader implications. Moreover, code generation presents distinct challenges for model quantization, as it requires maintaining accuracy across both natural language understanding and formal programming language constraints, effectively providing a stringent test of how quantization affects LLMs’ robustness.

To answer the above question, we conduct a systematic

investigation to benchmark the robustness of LLMs pre- and post-quantization from two different perspectives: adversarial attack on the input prompt and noise perturbation on the model architecture. We consider three different adversarial attacks from different levels, as well as two mainstream noise perturbation methods, performing experiments on four popular LLM families (*i.e.*, LLaMA [22], DeepSeek [21], CodeGen [30], and StarCoder [31]) with scales from 350M to 33B.

Surprisingly, our results reveal that quantized LLMs often exhibit greater robustness than their original versions, both in handling adversarial inputs and withstanding architectural noise. Specifically, in our extensive adversarial experiments, quantized LLMs successfully defended against more adversarial attacks in a significant majority of cases. Similar patterns emerged in our noise perturbation experiments, where quantized LLMs generally withstood a broader spectrum of noise attacks from various sources, proving that the common sense that “*smaller=weaker*” may not be the case in the context of quantization LLM for code generation.

In summary, we make the following contributions:

- We present the first comprehensive investigation exploring how quantization affects the robustness of LLMs in code generation tasks. Our work challenges the conventional wisdom that quantization necessarily trades model quality for efficiency.
- We evaluate robustness through dual complementary perspectives: input-level robustness using three types of adversarial attacks and model-level robustness through two systematic noise perturbations.
- Our extensive experiments across four prominent LLM families (LLaMA, DeepSeek, CodeGen, and StarCoder) with parameter scales ranging from 350M to 33B provide compelling evidence that quantized LLMs often exhibit superior robustness compared to their full-precision counterparts.
- We develop and open-source a flexible, standardized tool <sup>3</sup> for evaluating LLM robustness pre- and post-quantization, which is applicable to any LLM, facilitating future research.

## II. RELATED WORKS

### A. LLMs’ Robustness in Code Generation

Code generation has been identified as the most popular AI application among various coding tasks [19]. Hence, recent studies have extensively investigated the robustness of LLMs in code generation [15, 26, 25, 32, 33, 34, 29, 35, 36, 37, 28, 38, 39]. Mastropaolet al. empirically evaluated the robustness of GitHub Copilot by submitting semantically equivalent natural language description pairs. Their results indicated that Copilot produced different code recommendations in 46% of cases, with a corresponding 28% gap in correctness between these variants. Similarly, Shirafuji et al. and Zhuo et al. both proved that Codex is highly sensitive to even minor modifications in the natural language description [32, 33]. Moreover,

<sup>3</sup><https://github.com/safeai4code/adversarial-codegen>

Chen et al., Zhang et al., Zhang et al., and Improta et al. have shown that LLMs are vulnerable to adversarial attacks from various perspectives and have proposed different approaches to enhance their robustness [34, 29, 28, 39]. Notably, Wei et al. employed three distinct adversarial attacks to assess the robustness of LLMs pre- and post-quantization, finding that quantized models maintain robustness levels comparable to their original versions [15]. Besides, Qu et al. identified critical research directions for enhancing LLMs’ robustness in the code generation task, including developing realistic attack scenarios, improving risk assessment frameworks, and exploring to combine formal verification with neural networks to provide theoretical guarantees of robustness [27].

Our work advances the field in three key aspects. First, we conduct a comprehensive robustness evaluation of quantized LLMs from dual perspectives: input-level robustness through various adversarial attack methods, and model-level robustness through parameter weight perturbations. Second, through these complementary analyses, we demonstrate that quantized LLMs can achieve superior robustness compared to their original counterparts. Third, we present a detailed quantitative analysis explaining the mechanisms behind the enhanced robustness of quantized LLMs.

### B. Quantized LLMs for Software Engineering

While quantization has been widely adopted in general-purpose LLMs, its application in software engineering LLMs remains limited [15, 40]. Wei et al. conducted pioneering empirical studies on quantizing LLMs for code generation, demonstrating that 8-bit quantization can reduce model size and inference latency while maintaining competitive performance [15]. Their experiments across multiple programming languages showed a marginal 1.2% decrease in code generation correctness. In a parallel effort, Kaushal et al. proposed a low-rank decomposition method combined with quantization for LLMs, achieving up to a 4 $\times$  compression ratio with minimal impact on model effectiveness [40]. Weyssow et al. performed a comprehensive study on using QLoRA for code generation under resource constraints, demonstrating that QLoRA can reduce memory usage by up to 2 $\times$  compared to standard LoRA while maintaining or improving model effectiveness [14]. Their experiments showed that QLoRA enabled fine-tuning of large models up to 34B parameters using less than 24GB of GPU memory. For example, QLoRA-4bit achieved a remarkable 12.2% increase in performance over CodeLlama-7B-Python with LoRA on the Conala dataset, while CodeLlama-34B-Python fine-tuned with QLoRA-4bit achieved state-of-the-art performance on both Conala and CodeAlpacaPy benchmarks. More recently, Afrin et al. explored QLoRA for code summarization tasks, and their comprehensive study on CodeLlama and DeepSeek-Coder models showed that QLoRA not only matches but consistently outperforms full model fine-tuning, delivering superior results while significantly improving resource efficiency [13]. Specifically, QLoRA achieved a 2–3% improvement in METEOR scores across Python and Java tasks while reducing the memory footprint by approximately one-third. Their qualitative analysis

further revealed that QLoRA-optimized models can generate more accurate and informative code summaries than those written by developers in some specific cases.

Overall, our pioneering work is the first to systematically demonstrate that LLMs with 8-bit quantization not only retain but can enhance robustness compared to their full-precision counterparts in code generation tasks. Our findings strongly support the adoption of quantized LLMs in software engineering, not only to enhance model sustainability but also to strengthen the robustness of pipelines that integrate code generators as AI-powered development tools.

## III. METHODOLOGY

Our work consists of three fundamental stages: (S1) quantization; (S2) robustness attacks and; (S3) evaluation, as illustrated in Figure 2. The first stage focuses on the selection of target LLMs and appropriate quantization methods. The second stage applies selected attack strategies on LLMs both pre- and post-quantization, approaching from two distinct angles: adversarial attacks targeting the input, and noise attacks targeting the model architecture. In the final stage, we conduct a comprehensive evaluation of how quantization influences LLMs robustness by analyzing the outputs of each model under various attack conditions.

### A. S1: Selection of Quantization Method

Generally, quantization method could be categorized into the following two classes:

- **Dynamic Quantization** [41, 42] works at the inference phase, where scaling factors are computed dynamically based on the actual tensor values in each forward pass. For LLMs, this quantization strategy is particularly suitable for attention layers and feed-forward networks, where activation patterns can vary significantly across different inputs. Dynamic quantization offers flexibility in handling varying input distributions but may introduce additional computational overhead during inference. Most implementations focus on quantizing the weights and activations to INT8 or lower precision formats while keeping critical components like layer normalization in higher precision to maintain stability.

- **Static Quantization** [43, 44, 45, 17] determines quantization parameters prior to deployment, all of which remain fixed during inference. Currently, several advanced static quantization methods have been developed specially for LLMs. For instance, post-training quantization schemes like NF4 and FP4 use predefined parameters optimized for transformer architectures, eliminating the need for calibration data required in traditional techniques. In other hand, vector-wise quantization approaches such as GPTQ analyze entire weight matrices to preserve geometric properties. Generally, these specified quantization methods typically achieve better performance than dynamic quantization due to their optimization for transformer structures, reducing runtime overhead.

In this work, we select bitsandbytes<sup>4</sup> to quantize LLMs in the static way. There are some reasons [46, 47]: First,

<sup>4</sup><https://github.com/bitsandbytes-foundation/bitsandbytes>

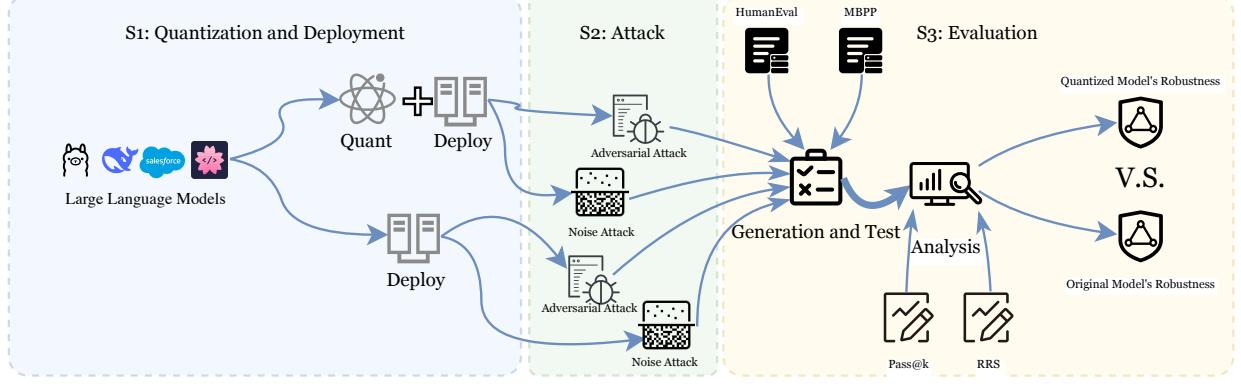


Figure 2. The overview of our methodology.

it implements specialized quantization schemes (NF4/FP4) that are specifically optimized for transformer architectures, maintaining the complex patterns and geometric properties learned during pre-training while achieving significant memory reduction. Second, it offers exceptional ease of integration within the modern LLMs development ecosystem, with seamless compatibility with the Hugging Face transformers<sup>5</sup>, thus it has been extensively used in LLMs quantization. Third, it provides a robust balance between memory efficiency and computational performance, achieving up to 4x memory reduction compared to FP16 models while maintaining competitive inference speed. Additionally, `bitsandbytes` eliminates the need for task-specific calibration data, which is particularly advantageous when working with LLMs where obtaining representative calibration datasets can be challenging. It also supports mixed-precision operations, allowing critical layers to remain in higher precision while aggressively quantizing less sensitive components, ensuring that accuracy is preserved for key model functionalities.

### B. S2: Robustness Attack Methods

We attack LLMs pre- and post-quantization from two different perspectives, *i.e.*, adversarial attacks on input and noise attacks on model architecture. For adversarial attacks, we employ three widely-used methods that have been extensively established in prior work [15, 26, 25, 39]:

- **Character-level:** This method randomly converts selected characters in the input prompt to uppercase letters. While preserving the semantic meaning, this simple perturbation tests the model’s sensitivity to character case variations.
- **Word-level:** This method randomly substitutes words in the prompt with their WordNet [48] synonyms. It assesses whether the model maintains consistent performance when encountering semantically equivalent alternatives while preserving the original prompt’s meaning.
- **Sentence-level:** This method employs back-translation, converting the English prompt to German and then back to English using a task-specific LLM. The resulting paraphrased versions

maintain the core meaning while testing the model’s ability to handle semantic-preserving reformulations.

For noise attacks, we employ two methods that have been extensively used to perturb LLMs [49, 50, 51]:

- **Gaussian noise:** Add random perturbations drawn from a normal distribution ( $\mathcal{N}(\mu, \sigma^2)$ ) to the model weights. It follows a bell-shaped distribution and is widely used in statistical analysis, simulating natural random variations in the weight parameters while maintaining specific statistical properties.
- **Uniform noise:** Apply random perturbations sampled from a uniform distribution ( $\mathcal{U}(a, b)$ ) to the model weights. It provides equal probability for all values within a specified range, evaluating the model’s resilience to consistent magnitude perturbations across all weight parameters.

### C. S3: Evaluation

**Task effectiveness measurement.** For each programming problem in our dataset, a dedicated test suite is provided to assess the accuracy of the generated solutions. By following prior works on code generation [15, 21], we reuse the same metric to evaluate a model’s performance in code generation, *i.e.*, if a generated program successfully passes all tests in the suite, it is considered a pass. In addition, the `pass@k` metric quantifies the likelihood of finding a working solution among  $k$  generated samples. Specifically, for each problem, the model generates  $k$  samples, and `pass@k` is defined as the probability that at least one of these samples is correct.

**Robustness measurement.** Previous work [15] have evaluated the robustness of LLMs pre- and post-quantization by calculating the percentage change in performance between perturbed and unperturbed inputs (as shown in Equation 1),

$$\% \Delta = \frac{\text{pass}@1_{\text{clean}} - \text{pass}@1_{\text{attack}}}{\text{pass}@1_{\text{clean}}} \quad (1)$$

However, it has several limitations when analyzing the impact of quantization on LLMs’ robustness: (1) it requires separate calculations rather than providing a direct comparative measure; (2) although it calculates relative performance differences within each model, the different baseline performance

<sup>5</sup><https://huggingface.co/docs/transformersd>

levels between original and quantized models is overlooked; (3) it lacks a standardized threshold to determine whether quantization improves or degrades relative robustness across models; and (4) it does not provide a normalized comparison of robustness degradation.

Inspired by previous work that focus on normalized comparison [52, 53, 54, 55, 56], we propose the Relative Robustness Score (RRS), which directly quantifies the relationship between robustness degradation in original and quantized models:

$$\text{RRS} = \frac{\text{pass}@1_{\text{orig}}^{\text{clean}} - \text{pass}@1_{\text{orig}}^{\text{attack}}}{\text{pass}@1_{\text{quant}}^{\text{clean}} - \text{pass}@1_{\text{quant}}^{\text{attack}}}, \quad (2)$$

Here,  $\text{pass}@1_{\text{orig}}^{\text{clean}}$  and  $\text{pass}@1_{\text{quant}}^{\text{clean}}$  denote the  $\text{pass}@1$  scores of the original and quantized models under normal (unperturbed) conditions, respectively, while  $\text{pass}@1_{\text{orig}}^{\text{attack}}$  and  $\text{pass}@1_{\text{quant}}^{\text{attack}}$  represent their corresponding scores under perturbations. An RRS value greater than 1 indicates that the quantized model experiences less performance degradation than the original model, thereby demonstrating better robustness against perturbations.

RRS offers several advantages: First, it provides a direct comparative measure between original and quantized models. Second, it establishes a clear interpretative threshold where  $\text{RRS} > 1$  indicates enhanced robustness in the quantized model, facilitating straightforward assessment of quantization effects. Third, by normalizing performance degradation across both model variants, RRS accounts for baseline performance differences, ensuring fair comparisons even when original and quantized models exhibit different accuracy levels under clean conditions.

#### IV. RESEARCH QUESTIONS & SETTINGS

##### A. Research Questions

In this work, we focus on the following research questions:

- **RQ1 (Adversarial Robustness Evaluation):** How does quantization affect the robustness of LLMs against existing adversarial attacks in code generation?
- **RQ2 (Noise Perturbation Evaluation):** How does quantization affect the robustness of LLMs against noise attack on the weights of LLMs in code generation?
- **RQ3 (Quantization-Robustness Trade-off Evaluation):** How does quantization create a trade-off between performance and resilience for LLMs in code generation task?

##### B. Studied Models

We select LLMs by prioritizing models that have been extensively studied and validated for their performance in code generation tasks [57, 58, 21, 59, 60, 61, 62]. As a result, our selection includes the following models:

- **LLaMA Family:** LLaMA [22], developed by Meta AI, is a family of open-source LLMs that have achieved state-of-the-art performance across numerous evaluation benchmarks. In our experiments, we evaluate three variants: LLaMA-3.2-1B, LLaMA-3.2-3B, and LLaMA-3.1-8B.
- **DeepSeek Family:** DeepSeek [21], another state-of-the-art LLM series developed by DeepSeek AI, has demonstrated

superior performance across diverse NLP and SE tasks. In this work, we utilize three code-specific variants: DeepSeek-Coder-1.3B-base, DeepSeek-Coder-6.7B-base, and DeepSeek-Coder-33B-base.

- **CodeGen Family:** CodeGen [30], developed by Salesforce Research, is a family of LLMs specifically trained for code. They are trained on a large corpus of codebase from various programming languages and have shown strong performance in code completion and generation tasks. In this work, we include three variants: CodeGen-350M, CodeGen-2B, and CodeGen-6B, the mono version.

- **StarCoder Family:** StarCoder [31], a notable advancement in LCMs, demonstrates exceptional performance in code understanding and generation tasks across multiple programming languages. In our study, we evaluate three model variants - StarCoder2-3B, StarCoder2-7B, and StarCoder2-15B.

##### C. Datasets

We evaluate our approach on two mainstream code generation benchmarks: HumanEval [23] and MBPP [24]. HumanEval consists of 164 Python programming problems with test cases, including string manipulation, mathematical operations, and data structure algorithms. MBPP (Mostly Basic Programming Problems) contains 974 Python programming tasks, designed to test fundamental programming abilities. To enhance result reliability, we additionally incorporate HumanEval Plus and MBPP Plus<sup>6</sup> [63], which expand the original test suites by 80x and 35x respectively, providing more comprehensive validation of model performance.

##### D. Model Quantization

We leverage bitsandbytes library<sup>7</sup> for model quantization, which supports both 4-bit and 8-bit precision levels. For 4-bit quantization, we employed the NF4 (normalized float 4) format with double quantization enabled and float32 compute dtype to balance efficiency with numerical stability. The 8-bit quantization uses standard linear quantization applied uniformly across model weights, all of which are built through the Hugging Face transformers library's BitsAndBytesConfig. With this quantization method, we could maintain consistent tokenization and generation capabilities between original and quantized models, ensuring fair comparison when evaluating robustness.

##### E. Robustness Attacks

**Adversarial Attack Implementation** We implement three distinct adversarial attack methods in different levels:

- **Character-level Attack:** We implemented by identifying all alphabetic characters in the prompt and randomly selecting a subset for transformation to uppercase. The attack uses a configurable character change probability (set to 0.5 in our experiments) and enforces a maximum change limit (5

<sup>6</sup>MBPP Plus filters low-quality samples to 378 problems, so we evaluate identical samples across both versions.

<sup>7</sup><https://huggingface.co/docs/bitsandbytes>

characters) to control perturbation intensity. The implementation first identifies all eligible alphabetic positions, calculates the probable number of changes based on the probability parameter, and then applies random sampling to select which characters to modify.

- **Word-level Attack:** In our implementation, we replace content words with synonyms from WordNet based on their part-of-speech tags. The attack targets specific POS categories (nouns, verbs, adjectives, adverbs) while excluding stopwords to preserve meaning. For each replaceable word, the system retrieves up to a maximum number of synonyms (3 in our experiments) and randomly selects one, with each replacement occurring with a probability of 0.15. We maintain proper text reconstruction by handling punctuation spacing patterns.

- **Sentence-level Attack:** We implemented back-translation using the Facebook mbart-large-50-many-to-many-mmt<sup>8</sup> model in a two-step pipeline. The attack first translates English text to German using the “en\\_XX” to “de\\_DE” configuration, then translates the result back to English using “de\\_DE” to “en\\_XX” setting. This creates natural paraphrases with subtly different word choices and sentence structures while maintaining semantic equivalence. We build it by leveraging Hugging Face transformers pipeline API.

All attacks are designed to only perturb the natural language portion of prompts by preserving code structure and test case. We set a consistent random seed to ensure reproducibility.

**Noise Perturbation Implementation** We evaluate architectural robustness by implementing controlled parameter noise perturbations across multiple intensity levels. By using PyTorch’s native functions, we produce two distinct noise sources: Gaussian noise via `torch.randn_like()` for normal distribution perturbations; and Uniform noise using `torch.randn_like()` transformed to balanced [-noise\_level, +noise\_level] ranges through `(torch.rand_like(param) * 2 - 1) * noise_level`. We systematically test each noise type at five increasing intensities ( $1e-4, 1e-3, 3e-3, 5e-3, \text{ and } 1e-2$ ), providing a granular assessment across a 100x range.

We optimize efficiency by wrapping all operations in PyTorch’s `no_grad()` context, selectively targeting only gradient-related parameters, and applying perturbations through in-place `add_()` operations to minimize memory overhead. This methodical approach enables precise identification of critical thresholds where model performance degrades, yielding insights into how quantization affects robustness against parameter perturbations.

## F. Evaluation Framework

Our evaluation framework assesses LLMs robustness through a two-step process. We leverage EvalPlus to evaluate the functional correctness of generated solutions against comprehensive test suites. It executes generated code in a secure sandbox environment, comparing outputs against ground-truth references with appropriate tolerance parameters. For each model variant and perturbation technique, we calculate the

pass@1 metric to measure base performance. To quantify robustness differences between original and quantized models, we implemented a custom Python script to calculate our proposed Relative Robustness Score (RRS). This score directly compares performance degradation ratios between original and quantized models under identical perturbation conditions.

## G. Experimental Environment

We conducted all experiments on a Ubuntu server with an AMD Ryzen Threadripper PRO 7955WX CPU, 256 GB RAM, and dual NVIDIA A6000 GPUs. We use Hugging Face transformers to use LLMs under different attack settings for code generation tasks.

## V. RESULTS

### A. RQ1: Adversarial Robustness Evaluation

The evaluation results are presented in Table I. We comprehensively compare the robustness of 12 (4 model families  $\times$  3 scales) LLMs pre- and post-quantization under different adversarial attack settings, including character-, word-, and sentence-level on four datasets. We use RRS to measure the robustness between LLMs pre- and post-quantization. For some special cases<sup>9</sup>, we directly use  $\uparrow$  or  $\downarrow$  without specific values to denote the robustness. Based on our results, we find that, 51.59% of cases show that quantized LLMs are more robust than their original versions, 42.86% of cases show that original LLMs are more robust than their quantization versions, and 5.56% of experiments show that there are no robustness difference between LLMs pre- and post-quantization. One notable exception is the HumanEval dataset, where the original LLaMA family demonstrates solid robustness, with 15 out of 18 experiments showing greater robustness compared to its quantized version, distinguishing from other model families.

a) *Analysis by attack methods:* We find that sentence-level attacks generally produce the most pronounced differences in robustness between original and quantized models, with some extreme RRS values, while character-level attacks show more moderate impacts across model families. For character-level attacks, we observe mixed results across different model families. Quantized LLMs show better robustness in many cases (e.g., LLaMA-1B, DeepSeek-6.7B, andCodeGen-6B), with RRS values greater than 1. However, for larger models like LLaMA-8B and StarCoder-7B, the original versions exhibit better robustness. On HumanEval datasets, most original LLMs present better robustness than their quantized versions, particularly in the LLaMA family. This suggests that the impact of quantization on character-level robustness is both model and dataset-dependent. For word-level attacks, there is a clear pattern across model scales. Smaller models (1-3B parameters) generally show improved robustness after quantization, as evidenced by LLaMA-1B, LLaMA-3B, and StarCoder-3B. However, larger models ( $>7$ B parameters) tend to maintain better robustness in their original form, such as LLaMA-8B, StarCoder-15B, and DeepSeek-33B. An exception is theCodeGen family, where quantization improves

<sup>9</sup>For example, when applying attack, quantized model gets improvement but original model gets deduction, the RRS would be a negative value.

<sup>8</sup><https://huggingface.co/facebook/mbart-large-50-many-to-many-mmt>

Table I

ROBUSTNESS COMPARISON OF ORIGINAL AND QUANTIZED LLMs UNDER DIFFERENT ADVERSARIAL ATTACKS. CH, W, AND S REPRESENT CHARACTER-LEVEL, WORD-LEVEL, AND SENTENCE-LEVEL ATTACKS, RESPECTIVELY.  $\uparrow$  MEANS ORIGINAL LLMs ARE MORE ROBUST, AND  $\downarrow$  MEANS QUANTIZED LLMs ARE MORE ROBUST. FOR STARCODER, WE DISCARD IT ON HUMANEVAL DATASET SINCE WE CANNOT GET ANY OUTPUTS.

Model	Size	MBPP			MBPP <sup>+</sup>			HumanEval			HumanEval <sup>+</sup>		
		Ch	W	S	Ch	W	S	Ch	W	S	Ch	W	S
LLaMA	1B	1.092 $\downarrow$	1.862 $\downarrow$	1.333 $\downarrow$	1.053 $\downarrow$	1.917 $\downarrow$	1.900 $\downarrow$	1.500 $\downarrow$	$\uparrow$	0.714 $\uparrow$	1.333 $\downarrow$	$\uparrow$	0.714 $\uparrow$
	3B	1.136 $\downarrow$	1.630 $\downarrow$	0.769 $\uparrow$	1.000	1.258 $\downarrow$	0.559 $\uparrow$	0.625 $\uparrow$	$\uparrow$	0.833 $\uparrow$	0.750 $\uparrow$	0.111 $\uparrow$	0.857 $\uparrow$
	8B	0.429 $\uparrow$	0.500 $\uparrow$	$\uparrow$	0.621 $\uparrow$	0.594 $\uparrow$	$\uparrow$	1.000	$\uparrow$	0.786 $\uparrow$	0.400 $\uparrow$	$\uparrow$	0.929 $\uparrow$
DeepSeek	1.3B	1.000	0.700 $\uparrow$	1.333 $\downarrow$	1.219 $\downarrow$	0.652 $\uparrow$	1.167 $\downarrow$	0.750 $\uparrow$	$\downarrow$	1.000	0.750 $\uparrow$	$\downarrow$	1.250 $\downarrow$
	6.7B	1.778 $\downarrow$	1.444 $\downarrow$	1.133 $\downarrow$	1.600 $\downarrow$	1.555 $\downarrow$	1.571 $\downarrow$	$\downarrow$	$\downarrow$	5.000 $\downarrow$	2.000 $\downarrow$	$\uparrow$	4.500 $\downarrow$
	33B	$\downarrow$	0.718 $\uparrow$	0.488 $\uparrow$	$\downarrow$	0.600 $\uparrow$	0.467 $\uparrow$	0.500 $\uparrow$	0.889 $\uparrow$	1.400 $\downarrow$	0.429 $\uparrow$	1.167 $\downarrow$	1.000
CodeGen	350M	0.942 $\uparrow$	0.758 $\uparrow$	0.857 $\uparrow$	1.017 $\downarrow$	1.000	6.997 $\downarrow$	0.875 $\uparrow$	1.200 $\downarrow$	1.250 $\downarrow$	0.833 $\uparrow$	2.000 $\downarrow$	2.000 $\downarrow$
	2B	1.029 $\downarrow$	2.437 $\downarrow$	1.437 $\downarrow$	0.829 $\uparrow$	1.661 $\downarrow$	1.083 $\downarrow$	0.400 $\uparrow$	3.000 $\downarrow$	1.667 $\downarrow$	0.455 $\uparrow$	5.999 $\downarrow$	1.200 $\downarrow$
	6B	1.315 $\downarrow$	1.286 $\downarrow$	1.667 $\downarrow$	1.289 $\downarrow$	1.410 $\downarrow$	1.571 $\downarrow$	0.600 $\uparrow$	3.666 $\downarrow$	0.714 $\uparrow$	0.909 $\uparrow$	2.667 $\downarrow$	0.833 $\uparrow$
StarCoder	3B	1.579 $\downarrow$	1.278 $\downarrow$	3.999 $\downarrow$	1.211 $\downarrow$	1.364 $\downarrow$	1.000	—	—	—	—	—	—
	7B	0.483 $\uparrow$	0.684 $\uparrow$	$\uparrow$	0.593 $\uparrow$	1.273 $\downarrow$	$\uparrow$	—	—	—	—	—	—
	15B	$\downarrow$	0.148 $\uparrow$	1.857 $\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	—	—	—	—	—	—

word-level robustness for 2B and 6B variants on HumanEval, with RRS values as high as 5.999. Regarding sentence-level attacks, we observe pronounced differences in robustness. The impact varies significantly across model families, with some showing extreme values (*e.g.*, StarCoder-3B with RRS=3.999 on MBPP and CodeGen-2B with RRS=5.999 on HumanEval). Overall, we observe that larger models tend to show more stable robustness between their original and quantized versions, while smaller models exhibit more dramatic differences.

*b) Analysis by model families:* The LLaMA family shows a clear model size-dependent pattern in robustness changes on MBPP datasets. The 1B and 3B variants generally become more robust after quantization across all attack types ( $\downarrow$ ), while the 8B variant consistently maintains better robustness in its original version ( $\uparrow$ ). This suggests that the effectiveness of quantization on robustness may have an inverse relationship with model size in the LLaMA family. However, as we mentioned early, original LLaMA models exhibit amazing robustness on HumanEval datasets.

**• DeepSeek Family.** They exhibit interesting patterns where the mid-sized 6.7B variant consistently shows better robustness after quantization ( $\downarrow$ ). The smallest 1.3B model shows mixed results, with better original robustness for word-level attacks on MBPP datasets and character-level attacks on HumanEval, but improved post-quantization robustness for sentence-level attacks. The larger 33B model maintains better robustness in its original form for word and sentence-level attacks, while showing improved quantized robustness for character-level attacks. This indicates that the relationship between model size and quantization impact on robustness is not strictly linear in the DeepSeek family.

**• CodeGen Family.** While not as uniform as initially described, these models show a tendency toward improved robustness after quantization, particularly for word and sentence-level attacks on HumanEval datasets. The 2B and 6B variants demonstrate notably high RRS values for word-level attacks on HumanEval datasets after quantization. However, a clear exception exists for character-level attacks on HumanEval datasets, where all original CodeGen models are more robust than their quantized versions. An exception is from char-level attack on HumanEval datasets, all original CodeGen models

are more robust than their quantized versions.

**• StarCoder Family** These models present the most varied results, with significant fluctuations in robustness across different sizes and attack types. The 3B variant predominantly shows improved robustness after quantization, while the 7B variant displays better original robustness under character-level attacks and some sentence-level attacks but mixed results for word-level attacks. The 15B variant exhibits inconsistent patterns, with better original robustness for word-level attacks on MBPP but improved post-quantization robustness for character and sentence-level attacks. This variability suggests that StarCoder’s robustness characteristics may be more sensitive to both quantization and attack type compared to other model families.

**Answer Summary to RQ1:** Quantization’s impact on LLM adversarial robustness varies significantly by model size and attack type. Smaller models (1-3B parameters) generally become more robust post-quantization, whereas larger models (>7B) maintain better robustness in their original version; Effects vary by attack type and model family: character-level attacks show model-dependent patterns, word-level attacks reveal size-dependent trends, and sentence-level attacks demonstrate the greatest differences. Model families exhibit distinct patterns: LLaMA shows an inverse size-dependent relationship, DeepSeek displays non-linear patterns, CodeGen generally improves after quantization, and StarCoder shows variable results.

### B. RQ2: Noise Robustness Evaluation

We evaluate the robustness of LLMs pre- and post-quantization against noise perturbations applied to their model weights. We apply two different types of noise (*e.g.*, Gaussian and Uniform) at five increasing intensity levels, *i.e.*, 1e-4, 1e-3, 3e-3, 5e-3, and 1e-2 (where higher values indicate stronger noise perturbations), to assess how quantization affects the LLMs’ resilience to weight perturbations. The results are presented in Figure 3 and we use pass@1 to measure the performance of each model.

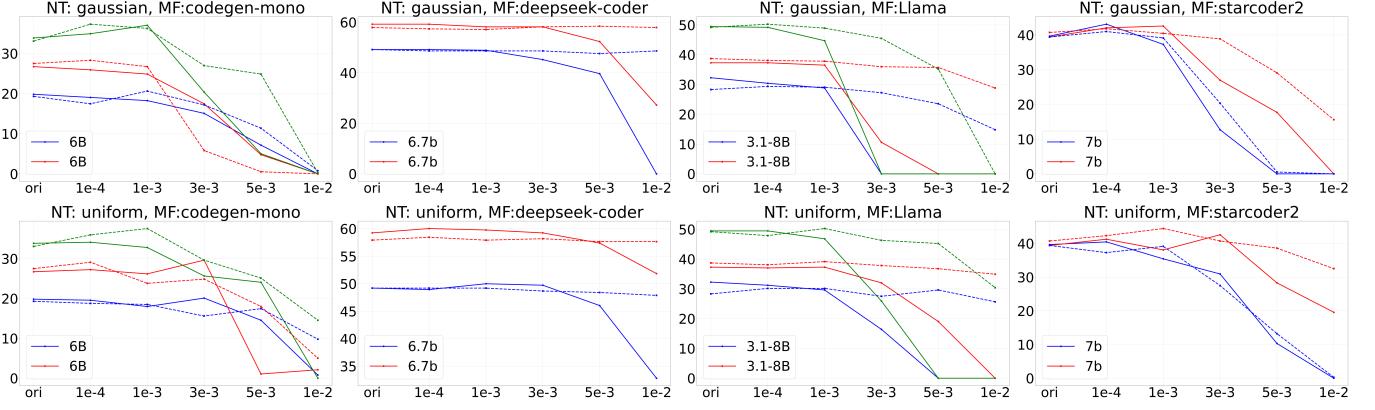


Figure 3. The results of noise robustness evaluation.

As shown in Figure 3, we observe a consistent pattern across different LLM families and model sizes: quantized models generally exhibit superior robustness against weight perturbations compared to their full-precision counterparts. Such enhanced robustness is particularly evident when noise levels increase ( $3e-3$  and above), where the performance gap between original and quantized LLMs becomes more pronounced.

*a) Analysis by model families:* CodeGen family shows mixed results regarding robustness enhancement through quantization. While CodeGen-350M and CodeGen-6B exhibit improved robustness after quantization, CodeGen-2B presents an interesting exception where the original model demonstrates better resilience to noise than its quantized counterpart. At moderate to high noise levels ( $3e-3$  and above), the original CodeGen-2B maintains higher pass rates than its quantized version. This contrasting behavior within the same model family suggests that the relationship between quantization and robustness may depend on specific architectural characteristics or parameter distributions that vary across model scales, even within the same family.

- **DeepSeek family.** We observe that DeepSeek pre- and post-quantization has a similar ability to defend against noise attacks. When the noise level continuously increases, the quantized model could still maintain a relatively high performance while the original model starts to crash. This indicates that quantization improves the range of the noise resilience.

- **LLaMA family.** We notice that models display consistent trends across different sizes. For LLaMA-3.2-1B and LLaMA-3.2-3B, the quantized versions consistently outperform their original counterparts at high noise levels ( $3e-3$  and above). For LLaMA-3.1-8B, both the original and quantized models show similar degradation patterns, with performance dropping to nearly zero at the highest noise level ( $1e-2$ ). However, at the intermediate noise level ( $5e-3$ ), the quantized LLaMA-3.1-8B maintains significantly better performance compared to its original counterpart. Overall, smaller LLaMA models appear to benefit more consistently from the regularizing effect of quantization than the larger 8B variant.

- **StarCoder family.** Our results show a clearly improved robustness through quantization. For both StarCoder-3B and StarCoder-7B variants, the quantized versions consistently

maintain higher pass rates at increased noise levels compared to their original counterparts. The difference is particularly pronounced at noise levels  $3e-3$  and above, where the original models exhibit steep performance degradation while the quantized versions maintain relatively stable performance. For example, at noise level  $5e-3$ , the quantized StarCoder-3B maintains significantly higher performance than its original version.

*b) Analysis by noise types:*

- **Gaussian noise.** We observe that quantized models generally maintain better performance as noise intensity increases across most model families. In the CodeGen family, quantized versions of CodeGen-350M and CodeGen-6B show superior resilience, while CodeGen-2B exhibits the opposite pattern with the full-precision model performing better. DeepSeek models maintain consistent performance until higher noise levels ( $5e-3$ ), where quantized variants demonstrate significantly better robustness as the original models' performance drops sharply. For LLaMA models, quantized versions consistently outperform their original counterparts at moderate to high noise levels, though both variants of LLaMA-8B degrade substantially at the highest noise levels. StarCoder shows similar benefits from quantization, with StarCoder-7B displaying the most significant improvement in noise resilience compared to its original version.

- **Uniform noise** We observe similar robustness patterns under uniform noise, with quantized models generally demonstrating superior resilience. The CodeGen family shows consistent results with the Gaussian case: quantized versions of CodeGen-350M and CodeGen-6B maintain higher performance at elevated noise levels, while CodeGen-2B shows better robustness in its full-precision form. DeepSeek models exhibit comparable performance between quantized and original variants at low noise levels, but as noise intensity increases to  $5e-3$  and beyond, the quantized models retain significantly more functionality. LLaMA models follow a similar pattern to their Gaussian noise response, with quantized versions of LLaMA-1B and LLaMA-3B consistently outperforming their original counterparts. StarCoder-7B demonstrates particularly significant benefits from quantization, maintaining substantially higher pass rates even at extreme noise levels, while StarCoder-3B shows minimal difference between its original

and quantized versions.

**Answer Summary to RQ2:** *Quantization generally improves LLMs’ robustness against weight perturbations (both in Gaussian and Uniform noise), with the effect becoming more prominent at higher noise levels ( $\geq 3e - 3$ ). Results vary by model family: CodeGen shows mixed results, DeepSeek maintains performance post-quantization under increasing noise, smaller LLaMA models benefit more from quantization, and StarCoder consistently shows enhanced robustness. These variations suggest quantization’s impact on robustness depends on specific architectural characteristics across model scales and families.*

### C. Quantization-Robustness Trade-off Evaluation

Our investigation reveals a relationship between quantization and robustness in LLMs. We conducted an extensive evaluation campaign spanning 252 experimental configurations (4 model families  $\times$  3 sizes  $\times$  3 precision levels  $\times$  (6 noise levels + 1 adversarial attack)) to understand this relationship across different model architectures and sizes. Each model is evaluated at three precision levels (full precision, 8-bit, and 4-bit quantization) while being tested against five increasing levels of Gaussian noise perturbation (1e-4, 1e-3, 3e-3, 5e-3, and 1e-2) plus one adversarial attack (sentence-level), allowing us to identify critical thresholds where performance begins to degrade significantly.

Figure 4 presents a visualization of how different models respond to noise under varying quantization levels. Quantization introduces a modest initial performance penalty—8-bit quantization decreases clean performance by just 0.36% on average, while 4-bit causes a more noticeable 1.19% reduction. However, this small sacrifice yields substantial robustness benefits when models encounter noisy inputs. We could observe three distinct regions of behavior: at low noise levels (below 1e-3), all precision variants perform comparably; at moderate noise levels (1e-3 to 5e-3), quantized models begin demonstrating superior resilience; and at high noise levels (5e-3 and above), original full-precision models suffer catastrophic performance collapse while quantized versions—especially 8-bit variants—maintain competitive performance.

Figure 5 presents the differential impacts of 8-bit and 4-bit quantization through Relative Robustness Scores (RRS). The comparison between blue bars (8-bit BNB) and green bars (4-bit BNB) reveals clear patterns across model families. 8-bit quantization delivers superior robustness benefits in several models, including CodeGen-350M, CodeGen-6B, LLaMA-3.2-1B, DeepSeek-1.3B, and DeepSeek-6.7B, where they have a higher RRS ( $>1$ ). In contrast, 4-bit quantization proves more beneficial for CodeGen-2B and StarCoder-3B. Interestingly, StarCoder-15B shows comparable robustness gains from both quantization levels, suggesting that for some models, the degree of quantization may not significantly impact robustness beyond a certain threshold. Most notably, several models—LLaMA-3.2-3B, LLaMA-3.1-8B, DeepSeek-

33B, and StarCoder-7B—actually demonstrate higher robustness in their original versions. These varied responses across model families and sizes highlights that optimal quantization strategies for maximizing both performance and resilience must be determined on a case-by-case basis through empirical evaluation. Generally, smaller LLMs tend to obtain robustness after quantization while larger LLMs have mixed tendency of robustness, which depends on specific model architecture.

**Answer Summary to RQ3:** *Across 252 experimental configurations, quantization could enhance LLM robustness against noise perturbations and adversarial attacks, with model-specific trade-offs. Generally, 8-bit quantization offers optimal balance, reducing performance by only 0.36% while substantially improving noise resistance. 4-bit quantization provides greater robustness at a higher cost (1.19% reduction). Smaller models (<3B parameters) handle aggressive quantization better than larger ones, which show architecture-dependent responses.*

## VI. DISCUSSION

### A. Quantization for Robustness?

**Quantization as noise.** Quantization introduces controlled noise into model parameters, which can be regarded as an implicit form of regularization. By reducing the precision of weights, quantization constrains the parameter space, limiting the model’s ability to overfit to non-robust features in the data. Such a regularization encourages LLMs to rely on more robust features that are better for generalization under adversarial conditions. As shown in Listing 6, our adversarial attack only introduce slight perturbation, transforming “p” in python into “P” and adding a word “the” to the front of the word “number”, but original LLMs cannot generate correct output for it while their quantized versions could do that. Our experimental results align with the observations from Tsipras et al., who show that robust models learn fundamentally different features than standard models, and quantization could potentially be regarded as robustness features [64].

**Reduced parameter sensitivity.** Quantization potentially reduces parameter sensitivity through its effect on feature representation. When model weights are quantized to lower bit precision, the model is forced to represent information more efficiently, focusing on the most essential features in code while discarding subtle or spurious correlations that full-precision models might overfit to. Such a feature simplification process naturally reduces sensitivity to parameter perturbations. With fewer bits to represent weights, quantized models develop more distinct and robust decision boundaries that are less affected by small adversarial shifts. Our noise perturbation experiments demonstrate this phenomenon clearly—quantized LLMs withstand higher levels of weight disturbances. The discrete nature of quantized weights creates natural thresholds to shield model behavior, whereas full-precision models can be influenced by infinitesimal changes to their parameters. As shown in Listing 7, when we replace some words in original

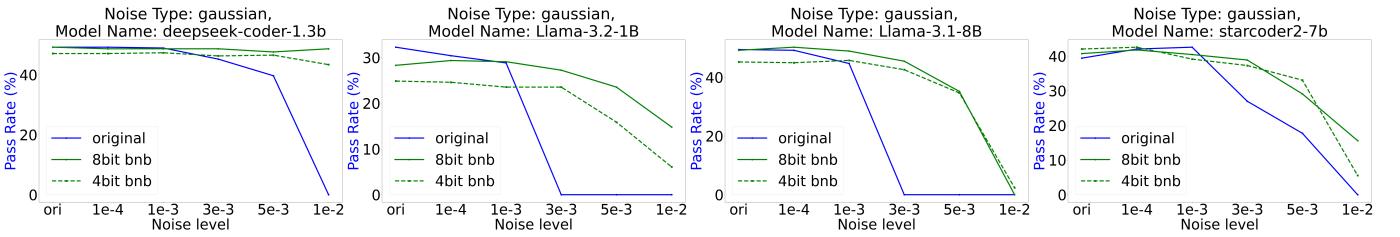


Figure 4. The results of quantization-robustness trade-off evaluation.

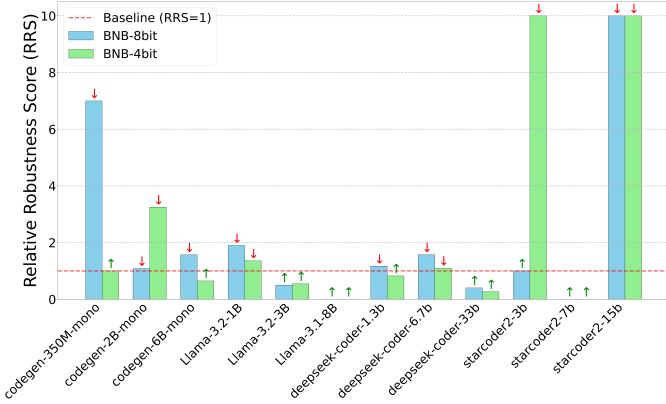


Figure 5. The results of quantization-robustness trade-off evaluation of translation attack.

prompt, it actually introduces slight semantical ambiguity. However, original LLMs are sensitive to this change and generate wrong solution, while their quantized versions could still generate accurate solutions by mitigating the ambiguity with the reference of the given test case example.

### B. Implications

Quantization can serve as a dual-benefit strategy that not only reduces computational requirements but also enhances model robustness, particularly for small to medium-sized models (1-7B parameters). Generally, our findings suggest size-dependent quantization recommendations: **smaller models (350M-3B) benefit from quantization with minimal performance impact; medium-sized models (3B-7B) achieve optimal balance with 8-bit quantization; and larger models (>7B) require careful consideration as effects vary by model family.** For application-specific strategies, **general-purpose code generation benefits the most from 8-bit quantization and resource-constrained deployments could consider aggressive quantization of smaller models.** Different model families show distinct patterns: CodeGen exhibits consistent improvement, LLaMA shows size-dependent benefits, DeepSeek displays non-linear improvement, and StarCoder presents variable results in different sizes. These insights can guide in deploying LLMs that are both efficient and reliable for code generation tasks.

### C. Threats to Validity

Our study on quantized LLMs' robustness in code generation faces several potential threats to validity that warrant

acknowledgment:

**Threats to internal validity.** The observed robustness improvements might be influenced by factors beyond the quantization itself, such as our chosen quantization method, which could affect results differently than alternative quantization techniques. Additionally, our implementation choices for the three adversarial attacks may impact results: the specific parameter settings such as the character-level attack's specific transformation parameters (0.5 probability, 5-character maximum). To mitigate these internal threats, we maintained consistent hyperparameters across all experiments for both original and quantized models, verified semantic preservation through manual inspection of attack outputs, calibrated parameters to ensure comparable perturbation intensities across attack levels, and designed our RRS metric specifically to normalize performance changes and enable fair comparisons between original and quantized models, reducing the impact of baseline performance differences.

**Threats to construct validity.** Discrepancies between theoretical expectations and observed outcomes can impact the validity of our investigation. A primary concern in this context is the selection of metrics used to evaluate the robustness of LMCs, as well as the methodologies employed to implement adversarial attacks. To mitigate these risk, we rely on well-established metrics and state-of-the-art approaches to attack LLMs. In particular, we implement multiple types of adversarial attacks (character-level, word-level, and sentence-level) and noise perturbations (Gaussian and Uniform) to ensure our robustness assessments were comprehensive across different perturbation types. While domain limitations remain, these measures help to keep the broader applicability of our findings within the code generation domain.

## VII. CONCLUSION

In this paper, we conducted the first comprehensive investigation of how quantization affects the robustness of LLMs in code generation tasks. Through extensive experiments across four prominent LLM families with multiple scales, we analyzed the robustness from dual perspectives: adversarial attacks on input prompts and noise perturbations on model weights. Our findings challenge the conventional wisdom that quantization necessarily involves a trade-off between efficiency and model quality. We discovered that quantized LLMs frequently demonstrate superior robustness compared to their full-precision counterparts, with 59.72% of our adversarial experiments showing better resilience in quantized models. This

```

Original prompt:
"""
Write a python function to count number of digits in a given
string.
assert number_ctr('program2bedone') == 1
"""

Adversarial prompt:
"""
Write a Python function to count the number of digits in a
given string.
assert number_ctr('program2bedone') == 1
"""

```

Listing 6. The prompt from MBPP<sup>+</sup> No. 764 task.Figure 6. The examples in MBPP<sup>+</sup> before/after sentence-level adversarial attack.

pattern was particularly pronounced in smaller to medium-sized models (1-7B parameters). Besides, quantization could consistently enhance robustness across different attack methods. Similarly, our noise perturbation experiments confirmed that quantized models could generally withstand higher levels of weight disturbances, suggesting that quantization not only reduces computational requirements but can actually improve model reliability.

In the future, we plan to extend this investigation with additional programming languages, model architectures, and quantization techniques. We also plan to explore more theoretical research to fully understand the mechanisms by which quantization enhances model robustness, prompting advanced quantization techniques specifically designed to optimize for both efficiency and robustness.

## REFERENCES

- [1] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” *arXiv preprint arXiv:1806.08342*, 2018.
- [2] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, “Pruning and quantization for deep neural network acceleration: A survey,” *Neurocomputing*, vol. 461, pp. 370–403, 2021.
- [3] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A survey of quantization methods for efficient neural network inference,” in *Low-power computer vision*. Chapman and Hall/CRC, 2022, pp. 291–326.
- [4] C. Watson, N. Cooper, D. N. Palacio, K. Moran, and D. Poshyvanyk, “A systematic literature review on the use of deep learning in software engineering research,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–58, 2022.
- [5] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, “Large language models for software engineering: A systematic literature review,” *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, pp. 1–79, 2024.
- [6] PrunaAI, “Deepseek coder 1.3b base bnb 4bit,” Hugging Face, 2023. [Online]. Available: <https://huggingface.co/PrunaAI/deepseek-ai-deepseek-coder-1.3b-base-bnb-4bit>
- [7] Unslloth, “Meta-llama-3.1-8b bnb 4bit,” Hugging Face, 2023. [Online]. Available: <https://huggingface.co/unslloth/Meta-Llama-3.1-8B-bnb-4bit>
- [8] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.
- [9] N. Jiang, T. Lutellier, Y. Lou, L. Tan, D. Goldwasser, and X. Zhang, “Knod: Domain knowledge distilled tree decoder for automated program repair,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1251–1263.
- [10] Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao, “Post-training quantization for vision transformer,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 092–28 103, 2021.
- [11] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “Qlora: Efficient finetuning of quantized llms,” *Advances in neural information processing systems*, vol. 36, pp. 10 088–10 115, 2023.
- [12] K. Egashira, M. Vero, R. Staab, J. He, and M. Vechev, “Exploiting LLM quantization,” in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. [Online]. Available: <https://openreview.net/forum?id=ISa7mMe7Vg>
- [13] S. Afrin, J. Call, K.-N. Nguyen, O. Chaparro, and A. Mastropaoletti, “Resource-efficient & effective code summarization,” *arXiv preprint arXiv:2502.03617*, 2025.
- [14] M. Weyssow, X. Zhou, K. Kim, D. Lo, and H. Sahraoui, “Exploring parameter-efficient fine-tuning techniques for code generation with large language models,” *ACM Transactions on Software Engineering and Methodology*, 2023.
- [15] X. Wei, S. K. Gonugondla, S. Wang, W. Ahmad, B. Ray, H. Qian, X. Li, V. Kumar, Z. Wang, Y. Tian *et al.*, “Towards greener yet powerful code generation via quantization: An empirical study,” in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 224–236.
- [16] A. Imani, I. Ahmed, and M. Moshirpour, “Context conquers parameters: Outperforming proprietary llm in commit message generation,” *arXiv preprint arXiv:2408.02502*, 2024.

```

Original prompt:
"""
Write a function to check if the given tuple has any none value or
not.
assert check_none((10, 4, 5, 6, None)) == True
"""

Adversarial prompt:
"""
Write a function to check if the specified tuple has no value or not.
assert check_none((10, 4, 5, 6, None)) == True
"""

```

Listing 7. The prompt from MBPP<sup>+</sup> No. 744 task.

- [17] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, “Awq: Activation-aware weight quantization for on-device llm compression and acceleration,” *Proceedings of Machine Learning and Systems*, vol. 6, pp. 87–100, 2024.
- [18] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, “Atom: Low-bit quantization for efficient and accurate llm serving,” *Proceedings of Machine Learning and Systems*, vol. 6, pp. 196–209, 2024.
- [19] J. T. Liang, C. Yang, and B. A. Myers, “A large-scale survey on the usability of ai programming assistants: Successes and challenges,” in *Proceedings of the 46th IEEE/ACM international conference on software engineering*, 2024, pp. 1–13.
- [20] N. Davila, I. Wiese, I. Steinmacher, L. Lucio da Silva, A. Kawamoto, G. J. P. Favaro, and I. Nunes, “An industry case study on adoption of ai-based programming assistants,” in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, 2024, pp. 92–102.
- [21] Q. Zhu, D. Guo, Z. Shao, D. Yang, P. Wang, R. Xu, Y. Wu, Y. Li, H. Gao, S. Ma *et al.*, “Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence,” *arXiv preprint arXiv:2406.11931*, 2024.
- [22] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.
- [23] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, “Evaluating large language models trained on code,” *arXiv preprint arXiv:2107.03374*, 2021.
- [24] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le *et al.*, “Program synthesis with large language models,” *arXiv preprint arXiv:2108.07732*, 2021.
- [25] A. Mastropaoletti, L. Pascarella, E. Guglielmi, M. Ciniselli, S. Scalabrino, R. Oliveto, and G. Bavota, “On the robustness of code generation techniques: An empirical study on github copilot,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2149–2160.
- [26] A. Jha and C. K. Reddy, “Codeattack: Code-based adversarial attacks for pre-trained programming language models,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 12, 2023, pp. 14 892–14 900.
- [27] Y. Qu, S. Huang, and Y. Yao, “A survey on robustness attacks for deep code models,” *Automated Software Engineering*, vol. 31, no. 2, p. 65, 2024.
- [28] C. Zhang, Z. Wang, R. Zhao, R. Mangal, M. Fredrikson, L. Jia, and C. Pasareanu, “Attacks and defenses for large language models on coding tasks,” in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 2268–2272.
- [29] C. Zhang, Z. Wang, R. Zhao, R. Mangal, M. Fredrikson, L. Jia, and C. S. Pasareanu, “Attacks and defenses for large language models on coding tasks,” in *ASE*, 2024, pp. 2268–2272. [Online]. Available: <https://doi.org/10.1145/3691620.3695297>
- [30] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, “Codegen: An open large language model for code with multi-turn program synthesis,” in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: [https://openreview.net/forum?id=iaYcJKpY2B\\_](https://openreview.net/forum?id=iaYcJKpY2B_)
- [31] A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtin, J. Liu, Y. Wei *et al.*, “Starcoder 2 and the stack v2: The next generation,” *arXiv preprint arXiv:2402.19173*, 2024.
- [32] A. Shirafuji, Y. Watanobe, T. Ito, M. Morishita, Y. Nakamura, Y. Oda, and J. Suzuki, “Exploring the robustness of large language models for solving programming problems,” *arXiv preprint arXiv:2306.14583*, 2023.
- [33] T. Y. Zhuo, Z. Li, Y. Huang, F. Shiri, W. Wang, G. Haffari, and Y.-F. Li, “On robustness of prompt-based semantic parsing with large pre-trained language model: An empirical study on codex,” *arXiv preprint arXiv:2301.12868*, 2023.
- [34] N. Chen, Q. Sun, J. Wang, M. Gao, X. Li, and X. Li, “Evaluating and enhancing the robustness of code pre-trained models through structure-aware adversarial samples generation,” in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 14 857–14 873.
- [35] Z. Yang, Z. Sun, T. Z. Yue, P. Devanbu, and D. Lo, “Robustness, security, privacy, explainability, efficiency, and usability of large language models for code,” *arXiv preprint arXiv:2403.07506*, 2024.
- [36] Y. Ge, W. Sun, Y. Lou, C. Fang, Y. Zhang, Y. Li, X. Zhang, Y. Liu, Z. Zhao, and Z. Chen, “Demonstration attack against in-context learning for code intelligence,” *arXiv preprint arXiv:2410.02841*, 2024.
- [37] D. Liu and S. Zhang, “Alanca: Active learning guided adversarial attacks for code comprehension on diverse pre-trained and large language models,” in *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2024, pp. 602–613.
- [38] X. Li, G. Meng, S. Liu, L. Xiang, K. Sun, K. Chen, X. Luo, and Y. Liu, “Attribution-guided adversarial code prompt generation for code completion models,” in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 1460–1471.
- [39] C. Improta, P. Liguori, R. Natella, B. Cukic, and D. Cotroneo, “Enhancing robustness of ai offensive code generators via data augmentation,” *Empirical Software Engineering*, vol. 30, no. 1, p. 7, 2025.
- [40] A. Kaushal, T. Vaidhya, and I. Rish, “LoRD: Low-rank decomposition of monolingual code LLMs for one-shot compression,” in *ICML 2024 Workshop on Foundation Models in the Wild*, 2024. [Online]. Available: <https://openreview.net/forum?id=br49PQvuMp>
- [41] S. Kim, C. Hooper, T. Wattanawong, M. Kang, R. Yan, H. Genc, G. Dinh, Q. Huang, K. Keutzer, M. W. Ma-

- honey *et al.*, “Full stack optimization of transformer inference,” in *Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023)*.
- [42] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, “A white paper on neural network quantization,” *arXiv preprint arXiv:2106.08295*, 2021.
- [43] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, “Q-bert: Hessian based ultra low precision quantization of bert,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 8815–8821.
- [44] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, “OPTQ: Accurate quantization for generative pre-trained transformers,” in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: <https://openreview.net/forum?id=tcbBPnfwxS>
- [45] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, “Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale,” *Advances in neural information processing systems*, vol. 35, pp. 30 318–30 332, 2022.
- [46] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, “8-bit optimizers via block-wise quantization,” in *International Conference on Learning Representations*, 2022. [Online]. Available: <https://openreview.net/forum?id=shpkpVXzo3h>
- [47] H. Face. (2023) Quantization with bitsandbytes. [Online]. Available: <https://huggingface.co/docs/transformers/main/en/quantization/bitsandbytes>
- [48] G. A. Miller, “Wordnet: a lexical database for english,” *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.
- [49] M. Liu, S. Liu, H. Su, K. Cao, and J. Zhu, “Analyzing the noise robustness of deep neural networks,” in *2018 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 2018, pp. 60–71.
- [50] Z. Gong, J. Liu, J. Wang, X. Cai, D. Zhao, and R. Yan, “What makes quantization for large language model hard? an empirical study from the lens of perturbation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 18 082–18 089.
- [51] F. Galli, L. Melis, and T. Cucinotta, “Noisy neighbors: Efficient membership inference attacks against LLMs,” in *Proceedings of the Fifth Workshop on Privacy in Natural Language Processing*, I. Habernal, S. Ghanavati, A. Ravichander, V. Jain, P. Thaine, T. Igamberdiev, N. Mireshghallah, and O. Feyisetan, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 1–6. [Online]. Available: <https://aclanthology.org/2024.privatenlp-1.1/>
- [52] D. Hendrycks and T. Dietterich, “Benchmarking neural network robustness to common corruptions and perturbations,” in *International Conference on Learning Representations*, 2019. [Online]. Available: <https://openreview.net/forum?id=HJz6tiCqYm>
- [53] F. Tramer, N. Carlini, W. Brendel, and A. Madry, “On adaptive attacks to adversarial example defenses,” *Advances in neural information processing systems*, vol. 33, pp. 1633–1645, 2020.
- [54] A. Subbaswamy, R. Adams, and S. Saria, “Evaluating model robustness and stability to dataset shift,” in *International conference on artificial intelligence and statistics*. PMLR, 2021, pp. 2611–2619.
- [55] J. Zhang, X. Xu, B. Han, G. Niu, L. Cui, M. Sugiyama, and M. Kankanhalli, “Attacks which do not kill training make adversarial learning stronger,” in *International conference on machine learning*. PMLR, 2020, pp. 11 278–11 287.
- [56] D. Hendrycks and K. Gimpel, “A baseline for detecting misclassified and out-of-distribution examples in neural networks,” in *International Conference on Learning Representations*, 2017. [Online]. Available: <https://openreview.net/forum?id=Hkg4Tl9xl>
- [57] A. Deroy and S. Maity, “Code generation and algorithmic problem solving using llama 3.1 405b,” *arXiv preprint arXiv:2409.19027*, 2024.
- [58] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez *et al.*, “Code llama: Open foundation models for code,” *arXiv preprint arXiv:2308.12950*, 2023.
- [59] H. Yu, B. Shen, D. Ran, J. Zhang, Q. Zhang, Y. Ma, G. Liang, Y. Li, Q. Wang, and T. Xie, “Codereval: A benchmark of pragmatic code generation with generative pre-trained models,” in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–12.
- [60] J. Feng, J. Liu, C. Gao, C. Y. Chong, C. Wang, S. Gao, and X. Xia, “Complexcodeeval: A benchmark for evaluating large code models on more complex code,” in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 1895–1906.
- [61] Z. Zeng, Y. Wang, R. Xie, W. Ye, and S. Zhang, “Coderujb: An executable and unified java benchmark for practical programming scenarios,” in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 124–136.
- [62] D. Zheng, Y. Wang, E. Shi, R. Zhang, Y. Ma, H. Zhang, and Z. Zheng, “Towards more realistic evaluation of llm-based code generation: an experimental study and beyond,” *arXiv preprint arXiv:2406.06918*, 2024.
- [63] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, “Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [64] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, “Robustness may be at odds with accuracy,” in *International Conference on Learning Representations*, 2019. [Online]. Available: <https://openreview.net/forum?id=SyxAB30cY7>