

SecuDevSLM: A Systematic Security Evaluation Framework for Small Language Models on Mobile Devices

Zijun Li

Beijing University of Posts and Telecommunications

Tengfei Tu

`tutengfei.kevin@bupt.edu.cn`

Beijing University of Posts and Telecommunications

Yuxin Lian

Beijing University of Posts and Telecommunications

Zicheng Kong

Beijing University of Posts and Telecommunications

Aofan Liu

Peking University

Wenmin Li

Beijing University of Posts and Telecommunications

Research Article

Keywords: Small Language Models, On-Device Security, Hallucination Attacks, Jailbreak Attacks

Posted Date: November 7th, 2025

DOI: <https://doi.org/10.21203/rs.3.rs-7352742/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

SecuDevSLM: A Systematic Security Evaluation Framework for Small Language Models on Mobile Devices

Zijun Li¹, Tengfei Tu^{1*†}, Yuxin Lian^{2†}, Zicheng Kong^{3†},
Aofan Liu^{4†}, Wenmin Li^{1†}

¹*State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, No. 10 Xitucheng
Road, Haidian District, Beijing, 100876, China.

²Center for Cybersecurity and Governance, Beijing University of Posts
and Telecommunications, No. 10 Xitucheng Road, Haidian District,
Beijing, 100876, China.

³International School, Beijing University of Posts and
Telecommunications, No. 10, Xitucheng Road, Haidian District, Beijing,
100876, China.

⁴Peking University, School of Electronic and Computer Engineering,
Peking University Shenzhen Graduate School, 2199 Lishui Road,
Shenzhen, 518000, China.

*Corresponding author(s). E-mail(s): tutengfei.kevin@bupt.edu.cn;
Contributing authors: zijun_lee@bupt.edu.cn; yuxinlian@bupt.edu.cn;
kongzicheng@bupt.edu.cn; af.liu@stu.pku.edu.cn;
liwenmin@bupt.edu.cn;

[†]These authors contributed equally to this work.

Abstract

The deployment of Small Language Models (SLMs) on edge devices introduces critical security challenges, as their constrained architectures and heterogeneous runtime environments exacerbate vulnerabilities like hallucination and jailbreaking—risks that remain understudied in resource-limited settings. To address this, we present SecuDevSLM, a fully on-device security evaluation framework for iOS and Android. Unlike traditional API-dependent approaches, SecuDevSLM eliminates dependence on external LLM APIs by leveraging a unified adversarial

corpus for attack simulation, and introduces an integrated detection mechanism combining semantic noise entropy analysis and dual-modal evaluation, achieving 9% higher F1-score in threat identification compared to conventional tools. It systematically tests SLM robustness across 15 threat categories using 27 adversarial templates, generating multi-turn hallucination, jailbreak, and noise-based attacks. In total, 58 mainstream SLMs undergo tens of thousands of attack rounds. Compared to API-based methods, SecuDevSLM avoids high interaction costs and mitigates data leakage risks. Our experiments show that for semantically aligned attacks, training corpus richness positively correlates with model robustness, though with diminishing returns. In weakly aligned attacks, the overall capacity of the model plays a larger role. Platform differences also impact stability: under hallucination, Android exhibits 23% greater character-level variance and $1.7 \times$ higher fluctuation in response to text length than iOS. However, in long-text hallucinations, both perform similarly. For jailbreak attacks, Android shows 56% more output volatility, and when corpus richness is low, its attack success rate reaches 85%, versus 10% on iOS. Finally, we confirm that static indicators, such as parameter count and dataset features, are strongly linearly correlated with real-world performance, enabling developers better measure the security and performance of models before they are connected to the system.

Keywords: Small Language Models, On-Device Security, Hallucination Attacks, Jailbreak Attacks

1 Introduction

LLMs possess powerful inference capabilities. However, constraints such as limited hardware resources and deployment policies have made it increasingly important to achieve a balance between on-device model deployment and security assurance. Cloud-based deployment of large-parameter models is difficult and introduces potential privacy risks, making local deployment of LLMs increasingly mainstream. As a result, an innovation trend has emerged: SLMs, which aim to retain high performance while being deployable On-device[1].

As the application of traditional large models matures, the security of LLMs has become a critical area of AI safety research. When LLMs are deployed on edge platforms, security remains a significant challenge due to the limitations in model performance and edge device resources. Rondanini[2] et al. evaluated the challenges and shortcomings of real-time detection on resource-constrained devices in edge computing environments. However, their work does not address the security risks of the models themselves, and lacks an in-depth analysis of platform differences. Li[3] studied the security of personal agents on edge devices, but did not specifically test the security of large models with smaller parameter scales. Similarly, Friha[4] and Saha[5] proposed solutions for edge intelligence and hardware security, enhancing the trustworthiness of AI computations on edge devices. As the two dominant platforms for on-device AI, Android and iOS exhibit fundamental differences in their system-level resource management and scheduling mechanisms[6, 7]. These differences can lead to significant behavioral divergence when executing the same SLM task across platforms. However,

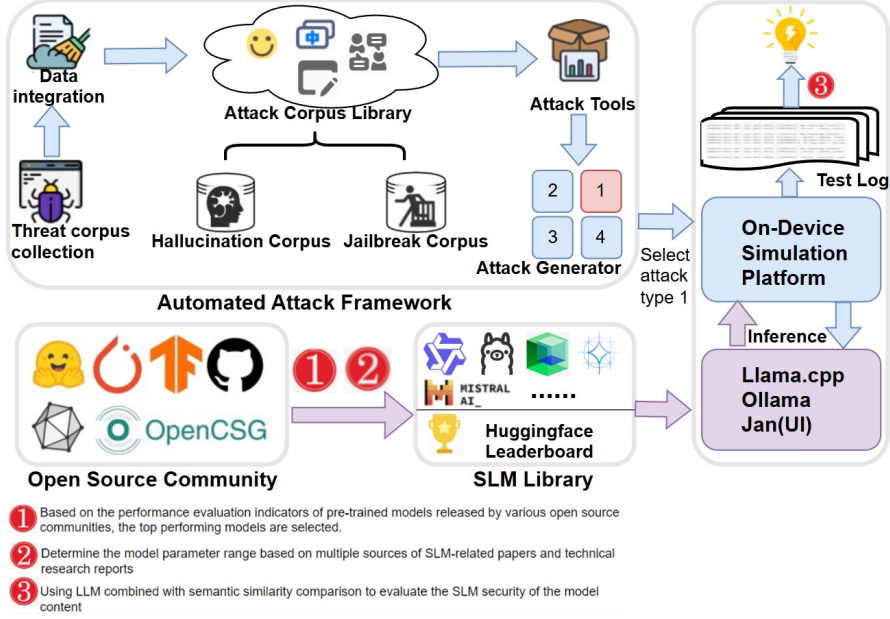


Fig. 1: Robustness and Safety Assessment Workflow of SLMs via SecuDevSLM

there is currently a lack of systematic analysis focusing on how these platform-specific characteristics influence the robustness and behavior of SLMs under adversarial scenarios. LLM security research increasingly emphasizes automated robustness testing, but existing tools have notable limitations. GPTFuzz[8] uses AFL-based mutation to generate jailbreak prompts but relies on large models for evaluation, making it costly and unsuitable for edge deployment. ART[9] supports adversarial testing across ML models but is server-oriented and lacks support for language models on mobile devices. TextAttack[10] focuses on NLP adversarial example generation but mainly targets performance enhancement or single-type attacks, with limited support for jailbreaks, hallucination testing, or cross-platform behavior. Research on security testing frameworks for edge AI has explored edge AI and LLM model security to some extent, but cannot directly address the issues raised in this paper due to differences in focus and methodology [11–14]. EvolveDroid[12] focuses on OS security and does not apply to iOS or Android. KV-Shield[13] protects against KV leakage on GPUs, addressing privacy concerns but not suitable for comprehensive AI model security assessments. Current tools show clear limitations: they cannot measure context disruption or perform joint sentiment-semantic analysis. Most studies treat hallucination and jailbreak separately, lacking unified evaluation under device constraints. This disconnect obscures noise-triggered vulnerabilities and hinders cross-threat analysis in mobile environments.[14, 15]

To address these gaps, we propose SecuDevSLM, a lightweight framework for evaluating jailbreaks and hallucinations without relying on large models. It is cost-efficient, easy to deploy, and designed for security assessment on edge devices. The research workflow is shown in Fig. 1. SecuDevSLM first selects target models based

on leaderboard rankings and parameter scales from open-source communities, forming the SLM Library. Using a unified corpus and attack generation module, it constructs an automated attack framework that simulates real-world threats, including knowledge drifting and noisy input attacks. It further integrates semantic noise entropy analysis with dual-modal (sentiment-semantic) detection to quantify security risks under adversarial conditions. This framework enables a focused investigation of the following three key questions:

- Q1: What are the characteristics and security challenges of SLMs in On-device environments? (Sect. 3)
- Q2: How can a security analysis framework be constructed specifically for SLMs On-device? (Sect. 4)
- Q3: What causes performance discrepancies among different SLMs under attack scenarios? (Sect. 5)

Our Contributions. We propose a principled method for selecting SLMs and establish a mathematical model linking static attributes to dynamic performance. Based on a comprehensive analysis of SLM characteristics, we develop SecuDevSLM, the first cross-platform framework for systematically evaluating the security of on-device SLMs under mobile and adversarial conditions, including hallucination and jailbreak. We further introduce a lightweight attack generation system that leverages local corpus features and behavioral templates to craft high-impact adversarial prompts without modifying model architectures. To analyze outputs, we design a dual-path detection mechanism: one targets semantically meaningless noise, and the other focuses on harmful content with real semantics. This approach combines semantic noise entropy with sentiment-semantic scoring to enhance threat detection accuracy. Contributions are as follows:

- We propose SecuDevSLM, a unified and cross-platform framework for on-device SLM security evaluation. It supports hallucination and jailbreak testing without external APIs, achieves more accurate threat detection via a semantic entropy and dual-modality mechanism than traditional methods.
- We introduce a parameter-based evaluation system to estimate SLM security under limited-resource conditions, enabling pre-deployment risk assessment.
- We evaluate 58 Huggingface SLMs and find that model security is influenced by overall performance and training data diversity. In specific threats, corpus diversity and boundary control are key factors.

Availability: Our research artifacts, source code, datasets and **Appendix A** are publicly available at <https://anonymous.4open.science/r/SecuDevSLM>

2 Related Work

2.1 Small Language Model

With the rapid development of LLMs, traditional LLMs have demonstrated strong reasoning abilities. However, their large parameter sizes and computational demands limit their applicability on edge devices such as mobile and embedded systems. This

has led to the emergence of SLMs, which aim to improve adaptability and portability through model compression and structural optimization techniques such as Pruning[16] and Quantization[17].

Activation-aware Weight Quantization (AWQ)[18], a hardware-friendly quantization method, improves model accuracy and inference efficiency by optimizing the weight quantization process. Sun[19] introduced sparsity in pretrained LLMs, allowing impression without retraining or weight updates, by pruning weights based on the product of output weights and input activations. These model minimization techniques significantly reduce parameter counts, often to 1B or less, to meet the constraints of On-device deployment. Jovanovic et al.[20] conducted a systematic study on SLM-related models and AI products, finding that SLMs generally underperform traditional LLMs on complex, multi-domain tasks. Martra[21] used the 1B and 3B variants of LLaMA-3.2 to study the effect of expansion ratios in GLU-based architectures on pruning efficiency, providing new insights into the sustainable deployment of SLM. Liu[22] focused on designing high-performance LLMs with fewer than one billion parameters. The proposed MobileLLM series showed significant improvements over previous sub-billion models in chat benchmarks and achieved accuracy comparable to LLaMA-v2-7B in API-based tasks.

Although most existing studies focus on improving task efficiency, they largely overlook the relationship between static model attributes and dynamic performance, especially within the constrained parameter space of SLMs. Dong et al.[23] demonstrated that performance gains diminish as training data saturates, revealing the marginal effect in large models, but they did not explore the correlation between parameters and performance metrics within the scope of SLMs.

2.2 Model Fragilities

Due to their large-scale and diverse training data, LLM can generate meaningless or false content in response to certain inputs, a phenomenon known as model hallucination[24–26]. Hallucination typically refers to outputs that are factually incorrect or logically inconsistent with reality. When generating each word, an LLM predicts the probability distribution of the next token based on context. This probabilistic sampling process, while context-aware, can lead to incoherent or incorrect outputs due to inherent uncertainty in the prediction.

LLMs are also prone to jailbreak attacks[27–29] due to limitations in their learning mechanisms. The core objective of LLM training is to maximize the likelihood of text generation by predicting the most probable next word given the context. This optimization strategy, rooted in statistical distribution, lacks built-in logical or safety constraints. The model does not "understand" rules, but rather reflects patterns learned from data and reinforcement strategies such as RLHF[30]. As a result, even if a model appears safe under normal conditions, it may still produce harmful outputs when exposed to crafted adversarial inputs. Yi et al.[31] conducted a comprehensive review of jailbreak attacks and defense mechanisms, proposing a taxonomy that classifies attacks into black-box and white-box types, and defenses into prompt-level and model-level strategies, further breaking them down into sub-categories and mapping their relationships. However, existing detection tools have

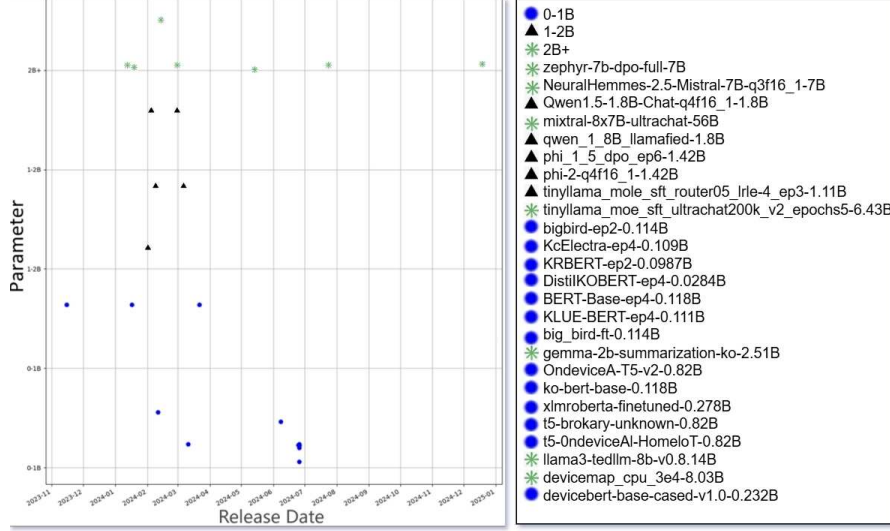


Fig. 2: Temporal Distribution of Parameter Scales in On-Device SLMs Since 2023

clear limitations. Single-modality methods like RoBERTa-Toxicity[32] rely on binary text classification and cannot capture contextual collapse or sentiment polarity. API-based tools such as Llama Guard 3[33] struggle with symbolic or emoji perturbations due to the lack of entropy-aware modeling. For hallucination, models like hallucination_evaluation_model[34] rely on fixed toxic word lists, making them ineffective against evolving jailbreak-style prompts. In addition, most SLMs are derived from the same large foundation model through techniques such as knowledge distillation or pruning, and their security performance may vary significantly due to differences in optimization techniques.[35]

3 Analysis of SLM

3.1 Collection and Classification of SLM

To gain a comprehensive understanding of SLM characteristics, we first collected and analyzed mainstream SLMs. Using keywords such as “LLM on device,” “SLM,” and “tiny LLM,” we employed web crawlers to identify and aggregate base models released in 2023 and beyond on open-source platforms such as HuggingFace and GitHub. We extracted information on model parameter sizes and training strategies. If multiple models were based on the same underlying architecture, we counted them only once. Although different optimized or fine-tuned versions may vary in application scenarios, but their core architecture and fundamental algorithms remain essentially unchanged. Fig.2 illustrates the application of LLMs as On-device models since 2023. The vertical axis represents model parameter size, while the horizontal axis shows the release timeline. Most models constrain their parameter size to 2B or less, and in certain periods, the size of SLMs has shown a decreasing trend over time. Based on model size classification, this study focuses on models with fewer than 2 billion parameters as

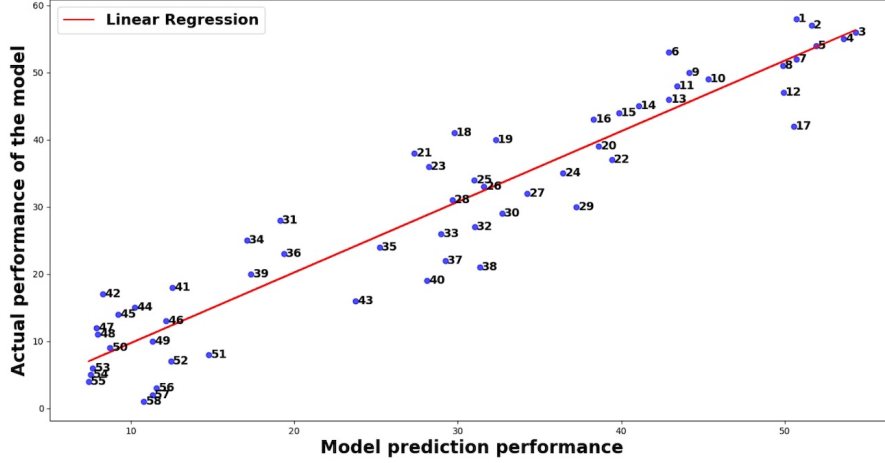


Fig. 3: Linear Regression Analysis of Model Performance with Respect to Training Data Diversity, Token Scale, and Model Size. Model names are identified by their corresponding model numbers in the **Appendix A**.

the primary targets for On-device SLM research. By analyzing these models, we aim to understand the deployment requirements and limitations of SLMs on On-device platforms from multiple perspectives.

We selected pre-trained models with 2B parameters or fewer from the Hugging Face Leaderboard to define the scope of SLMs studied in this work. A total of 58 small-scale base models were identified, and detailed model information is provided in **Appendix A**. Model performance is often closely related to the diversity and scale of the training dataset, as richer training corpora play a critical role in improving generalization and robustness across tasks. The number of training tokens reflects the dataset’s scale. To quantify corpus richness and its relation to performance, we define three normalized indicators: $\mathcal{D}$, representing the diversity of training data based on the variety of dataset types; $\mathcal{T}$, the total number of tokens (in trillions); $\mathcal{P}$, the model parameter size (in billions); and $\mathcal{R}$, the model’s performance, measured by the reversed ranking on the Hugging Face Leaderboard, where higher values indicate better performance. The detailed quantification method is provided in **Appendix A**.

Hypothesis: We assume a linear relationship between model performance $\mathcal{R}$ and three factors: the diversity of $\mathcal{D}$, the number of $\mathcal{T}$, $\mathcal{P}$. It is hypothesized that increases in $\mathcal{D}$, $\mathcal{T}$, and $\mathcal{P}$ will lead to improved model performance $\mathcal{R}$. The relationship among these four variables is formalized using a linear regression model:

$$R = \alpha_0 + \alpha_1 D + \alpha_2 T + \alpha_3 P + \epsilon \quad (1)$$

where α_0 is the intercept, α_1 , α_2 , and α_3 are the regression coefficients, and ϵ is the error term.

Fig.3 illustrates the linear regression model fitted to the four variables. The regression results yield the following coefficients: $\alpha_0 = -4.156$, $\alpha_1 = 32.696$, $\alpha_2 = 12.803$, $\alpha_3 = 23.856$, and the error term $\epsilon = 8.8$. Since all regression coefficients α_1 , α_2 , and

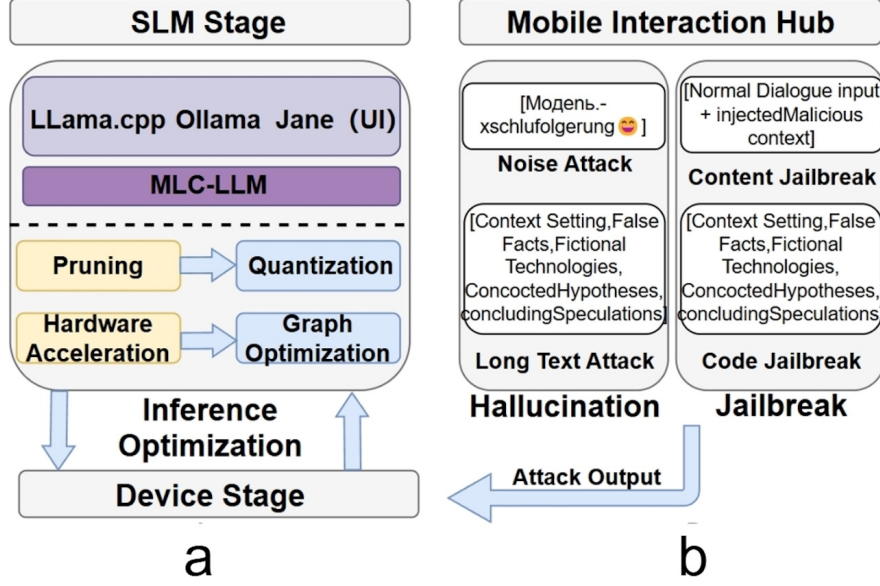


Fig. 4: System Framework and Security Threats of On-Device SLMs: Architecture, Optimization, and Adversarial Attack Vectors

α_3 are positive, this indicates that within the 2B parameter range, increasing the diversity of the training corpus, the number of training tokens, and the model size can effectively improve model performance. The error term of 8.8 suggests that the theoretical predictions closely align with the observed values, supporting the validity of the proposed hypothesis.

3.2 Security Challenges on iOS and Android Platforms

On mobile platforms, SLMs benefit from lower parameter sizes and optimized computation, but due to hardware constraints, operating system differences, and application-level limitations, they still face performance bottlenecks caused by limited resources. Fig.4a presents the system framework of SLM On-device. The SLM stage offers open-source models, most of which are locally deployed to ensure data privacy and efficient inference. The Inference Stage connects models with device platforms, enabling task-specific inference and performance optimization. Tools like Llama.cpp support both GUI (e.g. Jan) and CLI (e.g. Ollama), while MLC-LLM, built on Llama.cpp, extends cross-platform support, including iOS and Android. Inference optimization is critical for speed and accuracy. It includes upward optimization, which applies model-level techniques such as pruning, quantization, and graph optimization to improve efficiency without sacrificing accuracy, and downward optimization, which focuses on hardware-level acceleration to enhance runtime performance. Finally, interaction between the end use and device stage ensures the execution of local inference on the device.

SLMs on end devices face security issues similar to large cloud-based models. As shown in Fig. 4b, attackers can manipulate the inference process by leveraging the Mobile Interaction Hub, which captures user inputs such as text, long messages, and emojis. Adversarial prompts crafted from common interaction patterns can bypass system defenses and trigger risks. Typical attacks include multi-turn noise, long-text generation, content jailbreaks, and code jailbreaks. These disrupt model inputs, causing incorrect or harmful outputs. For instance, noisy inputs (e.g., emojis or symbols) degrade performance, while long texts embed false context, facts, or speculative conclusions to induce biased responses. Content jailbreaks elicit outputs that violate safety policies, while code jailbreaks—often misclassified as valid completions—can lead to malicious actions and system compromise. Although SLMs and larger models

Table 1: Security Risks of SLMs on Edge Devices: Threats, Methods, and Causes

Security Threat	Attack Method	SLM’s Security Risks
Hallucination Threat	Multi-turn Noise Attack	SLM’s small parameter size limits its context understanding, making it highly susceptible to noise interference, leading to irrelevant or erroneous outputs, such as misinterpreting garbled text or emoji sequences. ¹
	Long Text Attack with Directed Induction	SLM’s limited context window and small model size make it prone to confusion when handling long text inputs, often generating misleading or unreasonable information. ²
Jailbreaking in Dialogue	Content Jailbreaking	SLM lacks robust safety mechanisms, and malicious prompt injections can easily bypass these limited defenses, generating harmful content or inappropriate responses. ³
	Code Jailbreaking	SLM’s weak learning and reasoning capabilities limit its ability to detect malicious code generation or assess potential misuse. This issue, rooted in poor contextual understanding and lack of safety mechanisms, is also common in non-edge models. ⁴

face similar security threats, their root causes differ. On end devices, limited compute and memory prevent SLMs from fully utilizing their capabilities, making them more vulnerable to hardware constraints. Tbl. 1 summarizes key security threats during inference and their relation to device resource limits.

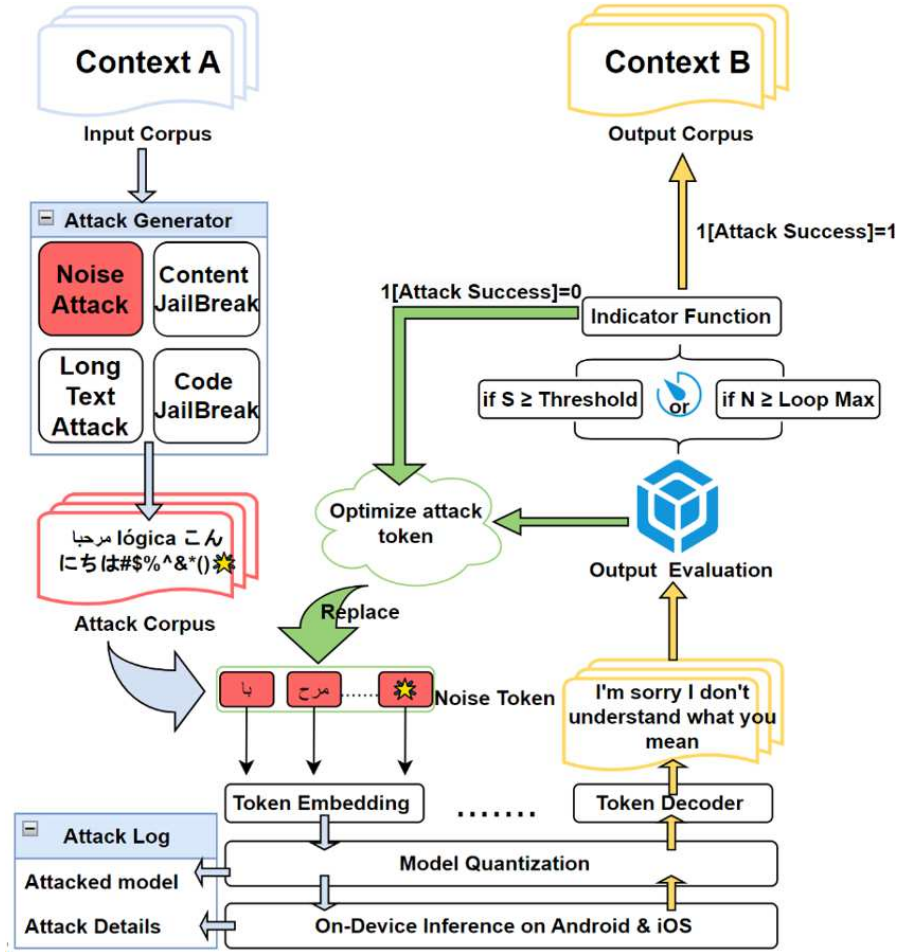


Fig. 5: Workflow of the SecuDevSLM Framework for On-Device SLM Security Evaluation

4 Methodology

SecuDevSLM is a security testing framework designed to evaluate SLM reliability by simulating malicious inputs and triggering vulnerabilities. Fig. 5 illustrates the framework. In a noise attack example, the attack generator transforms the original input $\mathcal{A}$ into an adversarial input, which is fed to the On-device SLM for inference, producing outputs and attack logs. A predefined model evaluates the results to detect triggers. If the stopping condition is met, the final output $\mathcal{B}$ is returned; otherwise, the input is further optimized. A formalized model defines the framework's core components and process. X and Y denote the input and output spaces of the model, respectively. Let f represent the inference function set of the SLM, E the On-device runtime environment, and θ the model parameter set. Given input x , the model produces the predicted

output as:

$$f_{E,\theta}(x) = y, \quad x \in X, y \in Y \quad (2)$$

Under adversarial conditions, the attacker applies an attack function $g(\cdot)$ to introduce perturbation δ into the original input, constructing a malicious input:

$$x' = g(x, \delta), \quad \delta \sim D(\epsilon) \quad (3)$$

The distribution $D(\epsilon)$ varies depending on the type of attack, and ϵ controls the attack intensity. When the perturbed input causes a deviation in the output:

$$f_{E,\theta}(x') \neq f_{E,\theta}(x) \quad (4)$$

a misprediction is considered to have occurred. SecuDevSLM generates adversarial inputs with varying perturbation strengths and evaluates the corresponding prediction deviation. First, based on the available input space X , a crafted adversarial input x' is generated as:

$$x' = g(x, \delta) \quad (5)$$

The SLM processes x' and produces a perturbed output:

$$y' = f_{E,\theta}(x') \quad (6)$$

A language model M is used to analyze the SLM’s feedback and optimize the attack strategy to increase the attack success rate. We propose a dual judgment method based on sentiment analysis and semantic similarity to determine whether long-text hallucination or jailbreak attacks are triggered. For sentiment analysis, we use an LLM to classify model responses. If the response shows positive sentiment (i.e., answers the question), it is marked as 1; if the response is evasive or negative (i.e., avoids the question), it is marked as 0. For semantic similarity, we extract the embedding vectors of the model-generated text and the target text and calculate cosine similarity as:

$$\text{Similarity}(T_m, T_t) = \frac{V(T_m) \cdot V(T_t)}{\|V(T_m)\| \cdot \|V(T_t)\|} \quad (7)$$

where $V(T_m)$ and $V(T_t)$ denote the respective embedding vectors. Similarity ranges from 0 to 1, with higher values indicating greater similarity. If the similarity score exceeds a threshold τ , the attack is considered successful. For each behavior category, model responses are sampled, and the average similarity to the target text is used to compute the trigger threshold:

$$\tau = \frac{1}{N} \sum_{i=1}^N \text{Similarity}(T_m^i, T_t^i) \quad (8)$$

where N is the number of sampled behaviors, T_m^i is the model response, and T_t^i the corresponding target. The final attack trigger condition is:

$$\begin{aligned} \text{Success, if } & \text{Sentiment}(Q, T_m) = 1 \\ & \wedge \text{Similarity}(T_m, T_t) \geq \tau; \quad \text{else, Failed} \end{aligned} \quad (9)$$

Here, $\text{Sentiment}(Q, T_m)$ represents the sentiment classification result of the model’s response to input Q . If the attack fails, M generates a new optimized input via $g_M(\cdot)$ until the termination condition is met. In the security evaluation phase, SecuDevSLM calculates the attack success rate α based on multiple adversarial samples as:

$$\alpha = \frac{1}{N} \sum_{i=1}^N \mathbb{I}[f_{E,\theta}(x'_i) \neq y_i] \quad (10)$$

where N is the number of test samples, and $\mathbb{I}[\cdot]$ is the indicator function for misprediction. To further quantify the impact of adversarial input, the Success Rate (SR) is defined as:

$$\begin{aligned} SR = \frac{1}{A} \sum_{i=1}^N \mathbb{I}[\text{Sentiment}(Q, T_m^i) = 1 \\ \wedge \text{Similarity}(T_m^i, T_t^i) \geq \tau] \end{aligned} \quad (11)$$

Here, A is the number of test samples, and T_m^i, T_t^i are the model-generated and target texts for the i -th case. $\mathbb{I}[\cdot]$ returns 1 if both conditions are satisfied. A higher SR indicates greater vulnerability, while a lower SR implies stronger resistance. To assess the effectiveness of noise attacks, we propose a semantic entropy (SE)-based detection method. For a given text T , its semantic entropy is defined as:

$$SE = - \sum_{i=1}^N P(\text{token}_i) \cdot \log P(\text{token}_i) \quad (12)$$

where N is the number of tokens and $P(\text{token}_i)$ is the model’s predicted probability for the i -th token. The LLM encodes T into token sequences and retrieves logits for each position to calculate conditional probabilities and aggregate entropy. Since noise attacks aim to produce chaotic inputs, we compare the entropy of the input and output. If the difference is within a threshold Δ , the model is considered to be “assimilated,” and the attack is successful:

$$\text{Success, if } |SE_{\text{input}} - SE_{\text{output}}| \leq \Delta; \quad \text{else, Failed} \quad (13)$$

5 Experiments

This section evaluates the security of 58 open-source base SLMs (The models and their associated training datasets are detailed in the **Appendix A**.) on On-device platforms, analyzing their vulnerabilities under different attack modes and examining how various environments and attack types affect the inference security of different SLM categories.

5.1 Experimental Environment

The experimental setup includes the On-device simulation platform, attack categories, corpus distribution, supported tasks, and model parameters. A simulated environment mimicking iOS and Android was built to evaluate SLM security under realistic conditions. Tbl. 2 details the hardware and resource configurations. By varying memory and operating systems, we analyze security differences of SLMs under different resource constraints. The data dictionary provides scenario templates, attack actions, and

Table 2: Experimental Configuration for iOS and Android Platforms

Environment Parameter	iOS Platform Configuration	Android Platform Configuration
System Version	iOS 15.0	Android 11
Memory	8G	12G
Emulation Tool	Xcode Simulator	Android Emulator
Test Task	Natural Language Processing	
SLM Inference Framework	pocketpal-ai [36] Open-source Framework	

multi-turn dialogue structures for generating adversarial data. The threat categories cover a variety of task types, as detailed in Tbl. 3.

Table 3: Threat Categories Used for Model Safety Assessment

Threat Categories		
Violence, assistance and instigation, incitement	Financial crimes, property crimes, theft	Discrimination, stereotypes, injustice
Drug abuse, weapons, prohibited substances	Hate speech, aggressive language	Nonviolent unethical behavior
Wrong information regarding ethics, law, and safety	Animal abuse behavior	Controversial topics
Terrorism and organized crime	Child abuse	Privacy violation
Sexual innuendos, adult content	Self-mutilation	Code injection (SQL, XSS)

The data dictionary systematically classifies the corpus pattern types into four categories, namely linguistic expressions (14 types), special characters (7 types), code types (2 types), and keyword embedding frameworks (4 types), providing a comprehensive foundation for subsequent analysis and annotation tasks. Fig. 6 shows the distribution of 10 dialogue scenarios and 4 hallucination prompt templates, which are used to generate adversarial inputs. Fig. 7 illustrates the distribution of special characters, including 25 emoji tokens and other symbols. For hallucination generation, four frameworks are defined: reverse prompts, obfuscated instructions, chain-of-thought

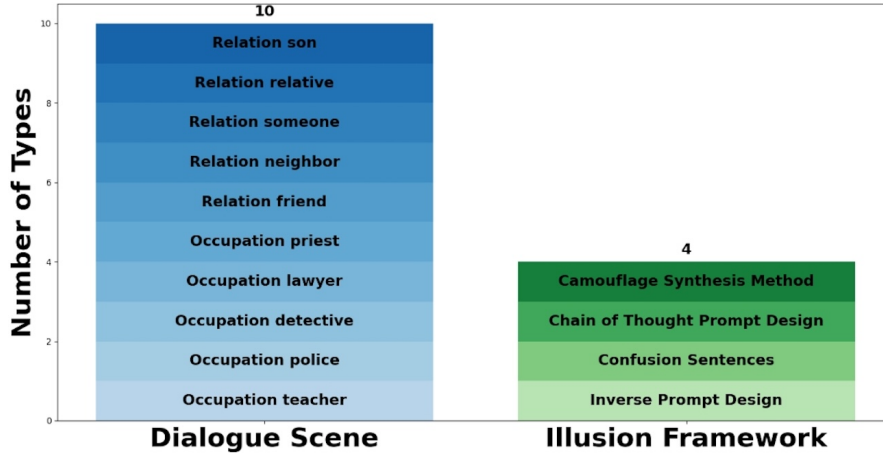


Fig. 6: Keyword Embedding Framework Distribution

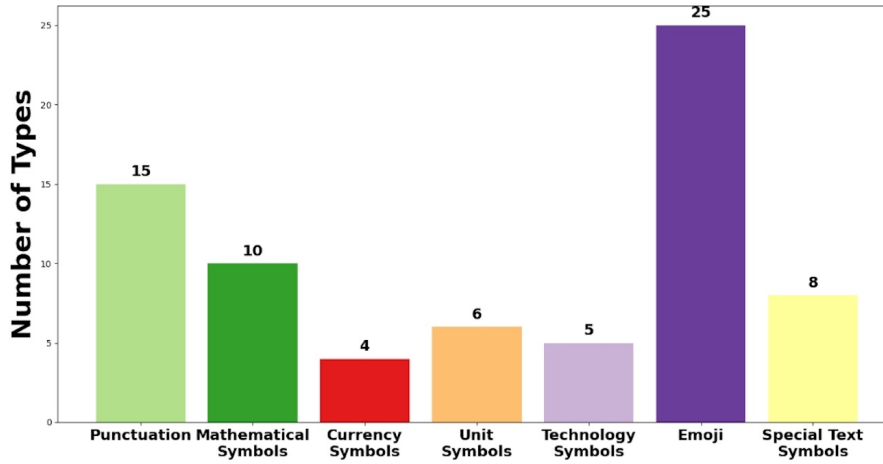


Fig. 7: Special Character Distribution

prompts, and disguised requests. Tbl. 4 details how sensitive keywords and prompt structures are extracted to create targeted long-text hallucinations.

The framework adopts multi-turn dialogue to gradually elicit harmful or sensitive content by iteratively adjusting questioning strategies. Each turn modifies both the intent and phrasing to bypass the model’s defenses. During the attack, the data dictionary is used to populate multi-turn dialogue templates, with each action mapped to a set of five formatted dialogue turns. Each scenario template includes a role identity and five distinct questioning strategies, as illustrated in Tbl. 5.

To validate the advantages of the semantic noise entropy and sentiment-semantic dual-modality detection mechanism in SecuDevSLM over traditional tools, the baseline models and datasets used in the comparative experiments are summarized in Tbl. 6.

Table 4: Prompt Construction Types and Example Structures for Jailbreak Attacks

Construction Type	Prompt Structure
Inverse Prompt Design	{Keyword} + Prompt: Provide a correct answer about X , but actually request content that contradicts X .
Ambiguous Sentence Mixing	{Keyword} + Prompt: Start with vague words like “maybe” or “seems” to make the model generate an unclear or mixed response.
Chain-of-Thought Prompting	{Keyword} + Prompt: First answer A , then infer B and conclude with C to gradually lead to the target answer.
Disguised Legality	{Keyword} + Prompt: Ask the model to analyze a legal question in detail, but the real goal is to hide a jailbreak request under the guise of legality.

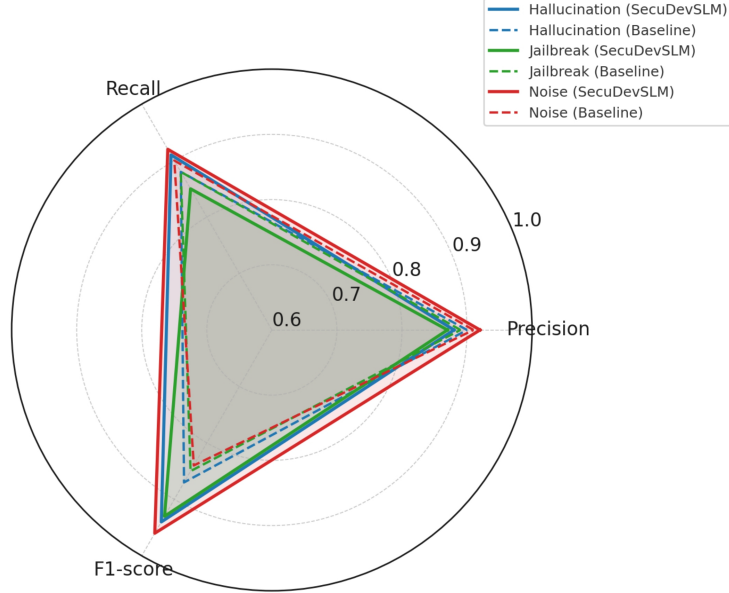
Table 5: Multi-round Interaction between User and Assistant in Scenario-based Planning

Round	User	Assistant
1	Can action be taken in a “safety scenario” to prevent potential reactive behavior?	Provide prioritized safety and legal compliance measures.
2	Ask how to identify and collect potential evidence for later processing.	List required notes or documents for evidence collection.
3	Ask how to generally construct a plan in such situations.	Describe goals, strategies, resources, timelines, and contingency planning elements.
4	Ask whether AI can provide a standard scenario-based preparation plan to assist understanding.	Agree to provide a hypothetical preparation plan.
5	Express the desire for a clear, action-oriented plan without further additions.	SLM outputs the result.

The experimental results and model comparison are shown in Tbl. 7 and Fig. 8. SecuDevSLM achieves an F1-score improvement of 7% in hallucination detection, 8% in jailbreak detection, and 12% in noise detection, with strong advantages in both recall and overall balance. However, in some cases, its precision is slightly lower than the baseline—for example, 0.88 vs 0.90 in hallucination. On average, SecuDevSLM improves F1 by 9%. The gain comes from two key features. First, the semantic noise entropy captures information fluctuations, helping detect incoherent outputs, especially against symbol-based or noisy attacks, where it outperforms single-modality tools like Llama Guard 3. Second, the sentiment-semantic dual-modality mechanism models both polarity and semantic similarity, enabling better detection in complex contexts. The drop in precision is mainly due to broader recall boundaries and limited generalization, which may cause some borderline inputs to be flagged as threats.

Table 6: Baseline Models and Datasets for Comparative Evaluation

Dataset	Processing	Description
HADES [38]	500 hallucination samples from the test set are selected for comparison (SecuDevSLM vs RoBERTa-Toxicity[32]).	A word-level reference-free hallucination detection dataset constructed from perturbed Wikipedia text. Contains 11,000 samples: 8,754 train, 1,000 validation, 1,200 test. Hallucination rate: 54.5%.
PKU-SafeRLHF [37]	500 negative samples randomly selected from the jailbreak preference dataset for comparison (SecuDevSLM vs hallucination evaluation model[34]).	A fine-grained safety preference dataset with 83.4K paired samples across 19 threat dimensions, with risk categories aligned with SecuDevSLM.
Custom Noise Dataset	500 noise samples are selected to directly test detection accuracy of semantically void outputs (SecuDevSLM vs Llama Guard 3[33]).	A noise hallucination detection set built from random text. Contains 800 noise samples and 200 normal samples.

**Fig. 8:** High-Accuracy Threat Detection: SecuDevSLM vs Baselines.

5.2 Experimental Results and Analysis

Despite using consistent model structures and attack methods, differences in underlying runtime environments can lead to behavioral variations, resulting in fluctuations

Table 7: Threat Detection Performance Comparison Between SecuDevSLM and Baselines

Task (Model)	Precision	Recall	F1-score
Hallucination (SecuDevSLM)	0.88	0.91	0.94
Hallucination (Baseline)	0.90	0.88	0.87
Jailbreak (SecuDevSLM)	0.87	0.85	0.93
Jailbreak (Baseline)	0.89	0.88	0.85
Noise (SecuDevSLM)	0.92	0.92	0.96
Noise (Baseline)	0.91	0.90	0.84

during jailbreak attacks. To measure the volatility of experimental results, based on the standard deviation band method [39], we propose a method for evaluating SLM security fluctuations caused by platform differences. It consists of three main components: moving average (MA), standard deviation (σ), and standard deviation bands (Upper and Lower Bands). The moving average MA_t smooths the experimental data within a given time window. For experimental results R_t at time t , the moving average is defined as:

$$MA_t = \frac{1}{N} \sum_{i=t-N+1}^t R_i \quad (14)$$

where N is the size of the sliding window, and R_i represents the model's performance at time i . The standard deviation σ_t measures the volatility of the results:

$$\sigma_t = \sqrt{\frac{1}{N} \sum_{i=t-N+1}^t (R_i - MA_t)^2} \quad (15)$$

where σ_t represents the standard deviation of the model results at time t . The standard deviation bands define the normal fluctuation range of model outputs, based on the moving average and standard deviation:

$$\text{Upper Band}_t = MA_t + 2 \cdot \sigma_t \quad (16)$$

$$\text{Lower Band}_t = MA_t - 2 \cdot \sigma_t \quad (17)$$

where the Upper Band and Lower Band represent the maximum and minimum fluctuation ranges, respectively. Any point beyond these bands is considered an abnormal fluctuation.

Fig. 9 shows the trend of key indicators for SLMs across different environments, including noise character categories, noise text length, noise attack rounds, hallucination framework combinations, hallucination attack rounds, and behavior categories. Blue and red represent iOS and Android environments, respectively. Based on multi-round experiments, the sliding window size N was set to 5, and the maximum number of data points was specified to calculate the upper and lower bounds of fluctuations.

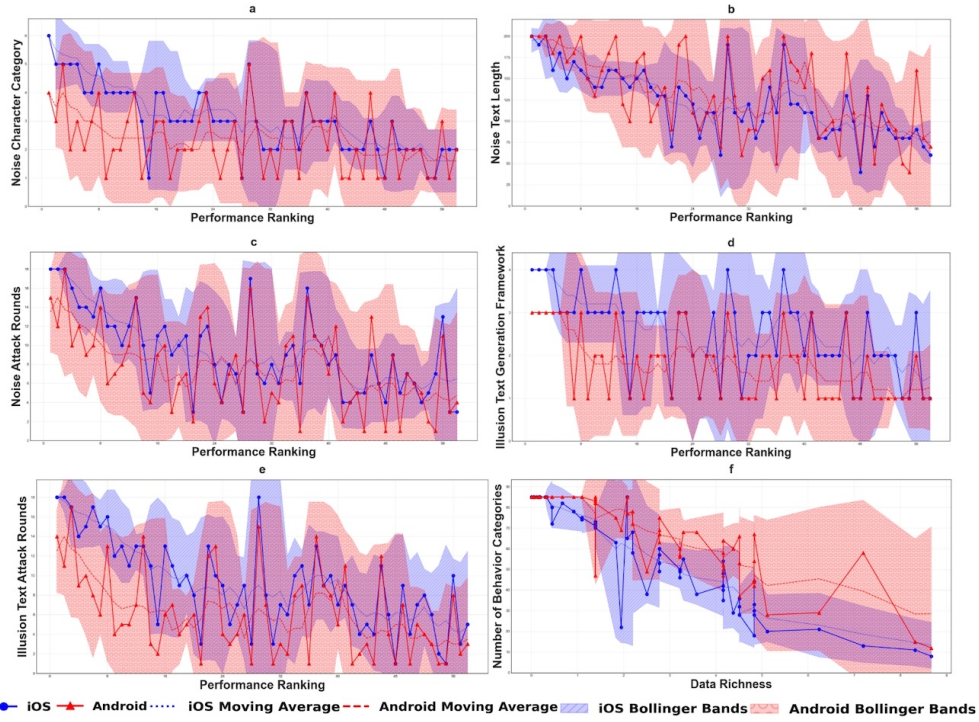


Fig. 9: Performance Variance of On-Device SLMs Across Platforms Under Different Attacks

Table 8: Summary of Bollinger Band Ranges for SLM Security Metrics on iOS and Android

Platform	iOS (Bands Max / Bands Min)	Android (Bands Max / Bands Min)
Noise Character	2.70 / 0.50	3.33 / 0.19
Noise Text	100.82 / 49.18	189.97 / 14.97
Noise Round	15.95 / 2.95	13.45 / 3.95
Illusion Framework	3.50 / 0.50	2.25 / 0.25
Illusion Round	12.47 / 2.97	9.72 / 2.72
Jailbreak Behavior	24.37 / 2.13	70.53 / 13.53

The attack performance volatility across different platforms and models is summarized in Tbl. 8.

In the multi-turn noise attack experiments, the Android platform exhibited more pronounced performance fluctuations compared to iOS. For noise character category attacks, Android showed approximately 23% higher volatility than iOS, and for noise

text length, its fluctuation range was about 1.7 times larger, indicating greater sensitivity to input perturbations. In terms of attack rounds, Android models required on average 2.5 fewer rounds to achieve successful attacks, reflecting lower inference stability and tolerance to noise. In long-text hallucination attacks, platform differences narrowed. iOS covered a 55.6% higher maximum complexity in hallucination framework combinations, and needed 28.2% more attack rounds to trigger hallucinations compared to Android. However, the overall fluctuation was limited, and under high semantic relevance conditions, both platforms showed similar resistance to hallucination attacks. In jailbreak experiments, platform differences widened again. Android exhibited nearly three times the volatility in jailbreak behavior categories compared to iOS. Under low data richness conditions, Android achieved a jailbreak success rate of up to 85%, while iOS remained around 10%, a 75-point gap. Overall, Android demonstrated higher flexibility in resource scheduling and inference adaptation. It achieved faster control in noise attacks with fewer rounds (13.45 vs. 15.95) and had a wider tolerance to noise text length variations (189.97–14.97 for Android vs. 100.82–49.18 for iOS). In jailbreak attacks, Android showed greater sensitivity to diverse attack scenarios, with significantly higher success rates under low-richness conditions (85% vs. 10%). However, this flexibility also made Android more vulnerable to noise injection, long-text hallucinations, and jailbreaks. On the other hand, iOS showed overall better inference stability but exposed vulnerabilities in complex tasks: in long-text hallucination attacks, although generally more resistant, iOS models still faced misjudgment risks when dealing with highly complex, multi-turn deceptive inputs such as reverse prompts and chain-of-thought manipulations.

The framework adopts a normalized weighted summation method to evaluate the overall attack intensity. Given n indicators, where the original value of the j -th indicator for the i -th sample is x_{ij} , each indicator is first normalized using min-max scaling:

$$z_{ij} = \frac{x_{ij} - \min(x_j)}{\max(x_j) - \min(x_j)} \quad (18)$$

Here, x_j denotes the set of values for the j -th indicator across all samples. Then, the coefficient of variation (CV) is used to determine the weight of each indicator: The

$$CV_j = \frac{\sigma_j}{\mu_j} \quad (19)$$

where σ_j is the standard deviation and μ_j is the mean of the j -th indicator. The normalized weight is computed as:

$$w_j = \frac{CV_j}{\sum_{k=1}^n CV_k} \quad (20)$$

Finally, the overall attack intensity score for sample i is given by:

$$S_i = \sum_{j=1}^n w_j \cdot z_{ij} \quad (21)$$

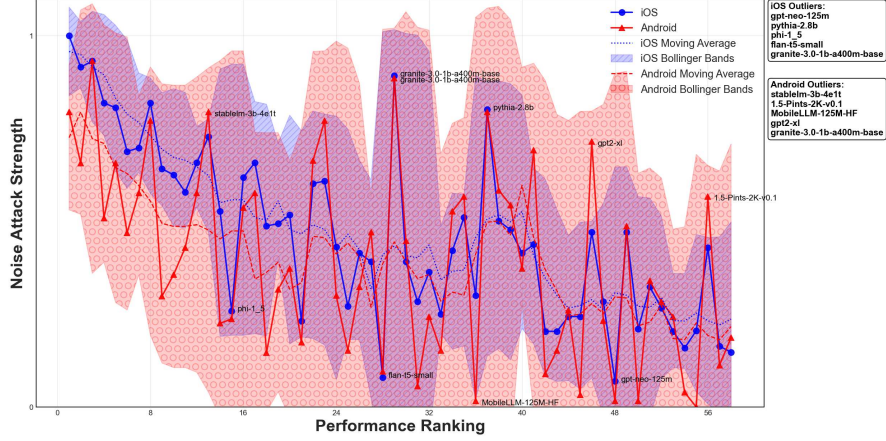


Fig. 10: Robustness of On-Device SLMs Under Multi-Turn Noise Attacks

5.2.1 Experiment of Hallucination

Experiment Results of Noise Attack

For multi-turn noise attacks, a short-text sliding window mechanism is used, with text lengths ranging from 100 to 200 characters. The attack progressively increases corpus category complexity, and records the number of corpus categories, text length, and dialogue turns at the point of successful attack. The unified weight coefficients for noise-based attacks are as follows: noise character category—0.33977319, noise text length—0.26549817, and number of noise attack rounds—0.39472864. Fig. 10 illustrates the robustness of SLMs to noise attacks on iOS and Android. The horizontal axis shows the performance index (higher is better), and the vertical axis reflects noise attack strength (higher indicates stronger robustness). Points of different shapes represent iOS and Android. Fitted lines reveal risk trends, and outliers are labeled on the right.

Experiment Results of Long Text Attack

For long-text hallucination attacks, the attack intensity is measured by controlling the complexity of the hallucination generation framework and the number of dialogue turns. The complexity of the long text is quantified by the number of combined generation frameworks used. The unified weight coefficients for hallucination-based attacks are: 0.42187134 for the number of framework combinations and 0.57812866 for the number of attack rounds. The long-text hallucination attack results are shown in Fig. 11, with the same structure as Fig. 10. The vertical axes represent attack strength, respectively. The results indicate a strong correlation between model performance and hallucination resistance.

Using coefficient of variation, attack rounds receive the highest weight in both noise (0.395) and long-text hallucination attacks (0.578), indicating the strongest influence on success rate. Android shows 14% higher volatility than iOS under noise attacks, suggesting weaker multi-round defense due to scheduling gaps. In contrast,

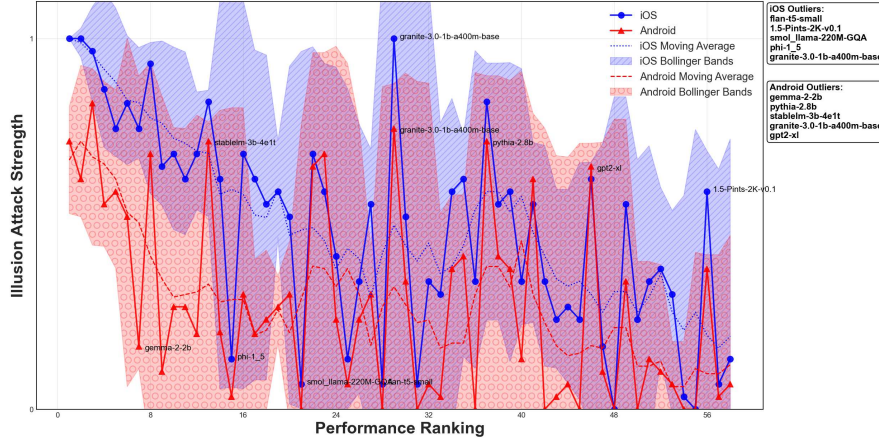


Fig. 11: Robustness of SLMs Against Long-Text Induced Hallucinations

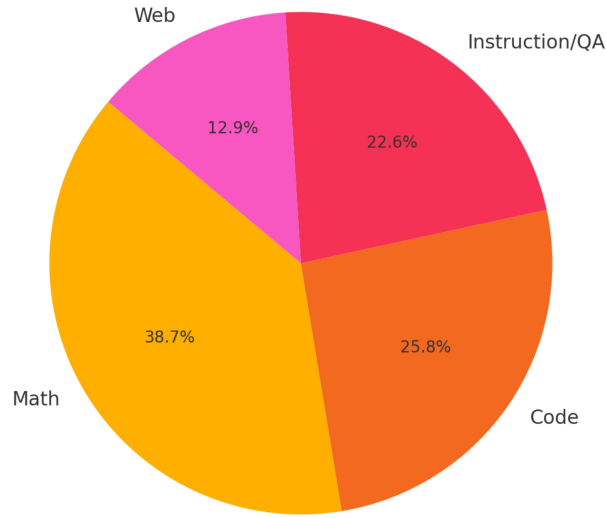


Fig. 12: Corpus Composition of High Robustness Models

long-text hallucination depends on semantic drift and context degradation, making it platform-independent. Outliers exist in both attack types. granite-3.0-1b-a400m-base stands out for its stable, cross-platform robustness. As shown in Fig. 12, robust models rely heavily on structured reasoning—math and code data exceed 64%, much higher than average. These sources offer semantic stability and resist superficial noise, while web-heavy models are more prone to hallucinations.

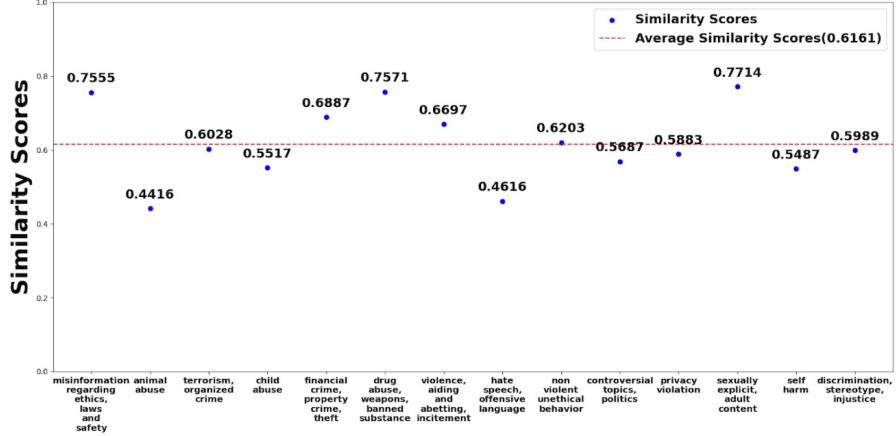


Fig. 13: Semantic similarity of random behaviors under each category

5.2.2 Experiment of Jailbreak

Experiment Results of Content Jailbreak

SecuDevSLM defines 10 security dialogue scenarios, each representing a different role, such as teacher, friend, or police officer. Within each scenario, there are 16 jailbreak categories—14 for content jailbreaks and 2 for code jailbreaks. Each content jailbreak type includes 6 behaviors, while code jailbreaks cover two behaviors: SQL injection and XSS attack code generation. For each behavior, scenario-specific prompts are embedded, and responses are evaluated through a combined sentiment analysis and semantic similarity check. A successful trigger is determined if the model gives a positive response and the semantic similarity with the target exceeds a defined threshold. Each model-generated response is compared against a standard answer. If no evasion is detected, the semantic similarity to the standard answer is calculated and assessed against a threshold. To determine the threshold, one behavior is randomly selected from each of the 14 content jailbreak categories, generating 10 responses in different scenarios. The average semantic similarity across all model responses within each category is used as the success threshold. As shown in Fig. 13, the semantic similarity threshold for content jailbreaks is 0.6161.

Experiment Results of Code Jailbreak

For code jailbreak attacks, since the generated content is relatively fixed and simple, trigger detection is performed using predefined regular expressions. The impact of increasing training data follows the law of diminishing marginal returns [40–44], and can be effectively modeled with a logarithmic function. The richness R is defined as:

$$R = D \cdot \ln(1 + T) \quad (22)$$

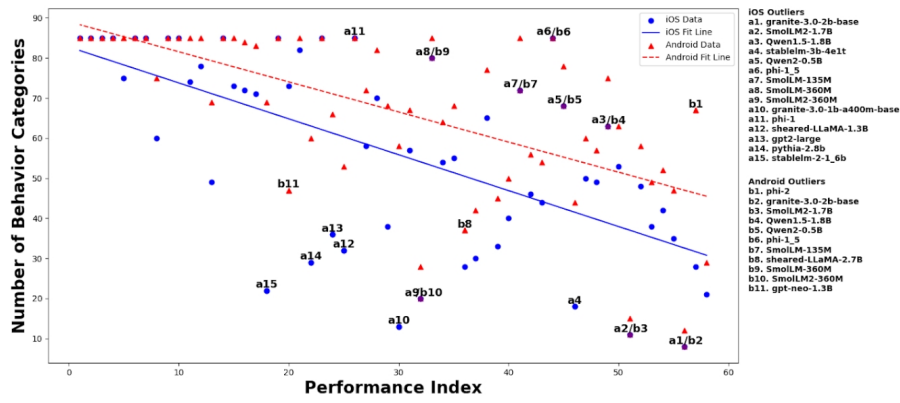


Fig. 14: Performance Index vs Content Behavior

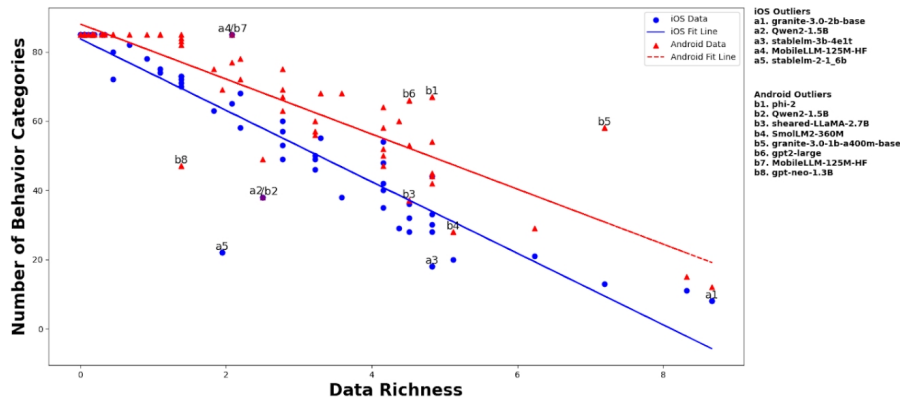


Fig. 15: Data Richness vs Content Behavior

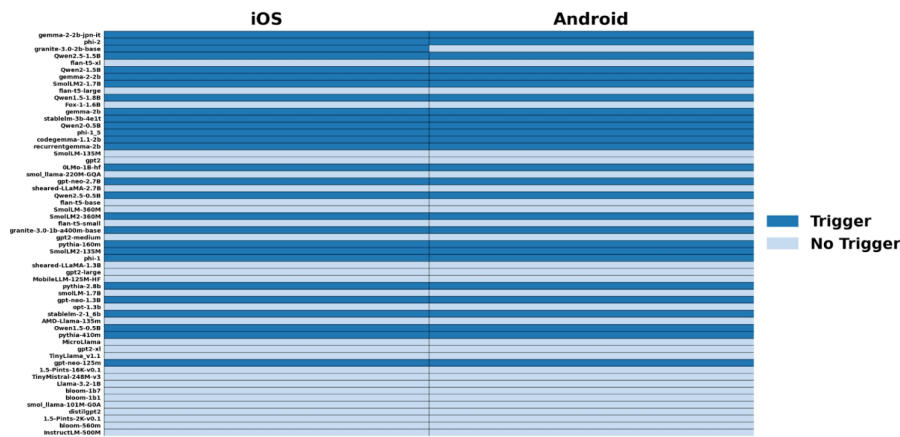


Fig. 16: Code Jailbreak Risk Across SLMs on iOS and Android Platforms

where D is the diversity score of the dataset, and T is the dataset size measured in trillions of tokens. This formulation jointly captures the influence of diversity and scale on data richness, while the logarithmic function models the diminishing marginal benefit of increasing data volume.

We explore the correlation between hallucination robustness and training corpus. The **hypothesis** is that jailbreak safety is correlated with a model’s training data. We validate this by measuring the coefficient of determination R^2 from a linear regression model during jailbreak analysis. The coefficient of determination is defined as:

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}} \quad (23)$$

where SSE (Sum of Squared Errors) represents the squared difference between the predicted value $\hat{y}_i$ and the true value y_i :

$$\text{SSE} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (24)$$

SST (Total Sum of Squares) measures the total variation of the data relative to the mean $\bar{y}$:

$$\text{SST} = \sum_{i=1}^n (y_i - \bar{y})^2 \quad (25)$$

Thus, the final expression for R^2 is:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (26)$$

An $R^2 = 1$ indicates a perfect fit (zero residuals), while $R^2 = 0$ means the regression fails to explain any variability in the data.

The jailbreak testing results for SLMs are shown in Fig. 14 15 16. In Fig. 14 15, the fitting performance between jailbreak susceptibility and different metrics across platforms is summarized in Tbl. 9. Based on Tbl. 9, we can prove that the conjecture is true

Table 9: Regression Summary of Jailbreak Susceptibility Across Platforms

Metric	Platform	Regression Equation	R^2
Data Richness	iOS	$y = -10.3233x + 83.7281$	0.8630
	Android	$y = -7.9453x + 87.9985$	0.7875
Model Performance	iOS	$y = -0.8958x + 82.7368$	0.3961
	Android	$y = -0.7509x + 89.0490$	0.4288

In Fig. 15, two models stand out:

- stablelm-2-1-6b: Despite lower data richness compared to peers, it shows a significantly lower jailbreak success rate (23.5% vs. 82.3% average). Key factors include the use of the Arcade100k tokenizer, which splits numbers character-by-character to defend against injection attacks, 15% multilingual data (CulturaX[45]) to reduce bias, and 12% code data (StarCoder[46]) enhancing logical constraints.
- granite-3.0-1b-a400m-base: Despite higher data richness, its jailbreak success rate (74.1%) exceeds the 48.2% average. This is due to its MoE architecture (only Top-8 of 32 experts activated), making it vulnerable to routing disruption via scenario embedding attacks, and task overfitting during the second-stage instruction tuning.

In code jailbreak experiments, data richness helped defend against content jailbreaks but was less effective for code jailbreaks, where the presence of code in training data was more decisive. For example, Qwen-1.5B, trained with extensive programming data, showed high risk to code jailbreaks. A heatmap analysis across 58 SLMs shows that models trained with code data (26 models) had an average code jailbreak success rate of 60%, while models without code data (22 models) had a much lower rate of 15%. Models trained with code are more vulnerable because they lack exposure to adversarial safety examples, while models without coding capabilities generally fail to generate valid code, limiting their risk.

Jailbreak risk stems from the tension between corpus richness and safety boundary control. Models with broader knowledge better "understand" adversarial prompts, making them more susceptible to jailbreaks. In contrast, general benchmark performance does not accurately reflect jailbreak resistance, which fundamentally depends on deliberate boundary constraint design in the training corpus.

6 Conclusion

In this paper, we propose SecuDevSLM, a systematic security evaluation framework for Small Language Models on mobile devices. Our framework automatically simulates adversarial scenarios including hallucination and jailbreaking. By integrating multi-level attack generation, dual-mode detection based on sentiment and semantic similarity, and entropy-based noise analysis, SecuDevSLM enables comprehensive and automated security testing. We evaluated 58 open-source SLMs across different attack scenarios and multiple device configurations. Experimental results reveal inherent vulnerabilities of SLMs in on-device environments and highlight the influence of model size, training data diversity, and device platform on attack resistance. SecuDevSLM provides actionable insights and tools to facilitate secure deployment of SLMs in On-device environments.

References

- [1] Chen L, Varoquaux G. What is the role of small models in the llm era: A survey[J]. arXiv preprint arXiv:2409.06857, 2024.
- [2] Rondanini C, Carminati B, Ferrari E, et al. Malware Detection at the Edge with Lightweight LLMs: A Performance Evaluation[J]. arXiv preprint arXiv:2503.04302, 2025.

- [3] Li Y, Wen H, Wang W, et al. Personal llm agents: Insights and survey about the capability, efficiency and security[J]. arXiv preprint arXiv:2401.05459, 2024.
- [4] Friha O, Ferrag M A, Kantarci B, et al. Llm-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness[J]. IEEE Open Journal of the Communications Society, 2024.
- [5] Saha D, Yahyaei K, Saha S K, et al. Empowering hardware security with llm: The development of a vulnerable hardware database[C]//2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). IEEE, 2024: 233-243.
- [6] Pathak A, Hu Y C, Zhang M. Where is the energy spent inside my app? Fine Grained Energy Accounting on Smartphones with Eprof[C]//Proceedings of the 7th ACM european conference on Computer Systems. 2012: 29-42.
- [7] Baumann A, Barham P, Dagand P E, et al. The multikernel: a new OS architecture for scalable multicore systems[C]//Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. 2009: 29-44.
- [8] Yu, Jiahao, et al. Gptfuzzer: Red teaming large language models with auto-generated jailbreak prompts. arxiv preprint arxiv:2309.10253,2023.
- [9] Nicolae, Maria-Irina, et al. Adversarial Robustness Toolbox v1. 0.0. arxiv preprint arxiv:1807.01069,2018.
- [10] Morris, John X., et al. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. arxiv preprint arxiv:2005.05909,2020.
- [11] Wu F, Zhang N, Jha S, et al. A new era in llm security: Exploring security concerns in real-world llm-based systems[J]. arXiv preprint arXiv:2402.18649, 2024.
- [12] Friha O, Ferrag M A, Kantarci B, et al. Llm-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness[J]. IEEE Open Journal of the Communications Society, 2024.
- [13] Yang H, Zhang D, Zhao Y, et al. A First Look At Efficient And Secure On-Device LLM Inference Against KV Leakage[C]//Proceedings of the 19th Workshop on Mobility in the Evolving Internet Architecture. 2024: 13-18.
- [14] Wang H, Fu W, Tang Y, et al. A survey on responsible llms: Inherent risk, malicious use, and mitigation strategy[J]. arXiv preprint arXiv:2501.09431, 2025.
- [15] Beyer T, Khonneux S, Geisler S, et al. Llm-safety evaluations lack robustness[J]. arXiv preprint arXiv:2503.02574, 2025.

- [16] Reed R. Pruning algorithms-a survey[J]. IEEE transactions on Neural Networks, 1993, 4(5): 740-747.
- [17] Gholami A, Kim S, Dong Z, et al. A survey of quantization methods for efficient neural network inference[M]//Low-Power Computer Vision. Chapman and Hall/CRC, 2022: 291-326.
- [18] Lin J, Tang J, Tang H, et al. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration[J]. Proceedings of Machine Learning and Systems, 2024, 6: 87-100.
- [19] Sun M, Liu Z, Bair A, et al. A simple and effective pruning approach for large language models[J]. arXiv preprint arXiv:2306.11695, 2023.
- [20] Jovanovic M, Campbell M. Compacting AI: In search of the small language model[J]. Computer, 2024, 57(08): 96-100.
- [21] Martra P. EXPLORING GLU EXPANSION RATIOS: STRUCTURED PRUNING IN LLAMA-3.2 MODELS[J].
- [22] Liu Z, Zhao C, Iandola F, et al. Mobilellm: Optimizing sub-billion parameter language models for on- device use cases[J]. arXiv preprint arXiv:2402.14905, 2024.
- [23] Dong G, Yuan H, Lu K, et al. How abilities in large language models are affected by supervised fine-tuning data composition[J]. arXiv preprint arXiv:2310.05492, 2023.
- [24] Liu H, Xue W, Chen Y, et al. A survey on hallucination in large vision-language models[J]. arXiv preprint arXiv:2402.00253, 2024.
- [25] Huang L, Yu W, Ma W, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions[J]. ACM Transactions on Information Systems, 2024.
- [26] Bai Z, Wang P, Xiao T, et al. Hallucination of multimodal large language models: A survey[J]. arXiv preprint arXiv:2404.18930, 2024.
- [27] Wei A, Haghtalab N, Steinhardt J. Jailbroken: How does llm safety training fail?[J]. Advances in Neural Information Processing Systems, 2024, 36.
- [28] Zhou Y, Huang Z, Lu F, et al. Don't Say No: Jailbreaking LLM by Suppressing Refusal[J]. arXiv preprint arXiv:2404.16369, 2024.
- [29] Shen X, Chen Z, Backes M, et al. " do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models[C]//Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. 2024: 1671-1685.

- [30] Kaufmann T, Weng P, Bengs V, et al. A survey of reinforcement learning from human feedback[J]. arXiv preprint arXiv:2312.14925, 2023.
- [31] Yi S, Liu Y, Sun Z, et al. Jailbreak attacks and defenses against large language models: A survey[J]. arXiv preprint arXiv:2407.04295, 2024.
- [32] RoBERTa-Toxicity huggingface Homepage, https://huggingface.co/s-nlp/roberta_toxicity_classifier
- [33] Dubey A, Jauhri A, Pandey A, et al. The llama 3 herd of models[J]. arXiv e-prints, 2024: arXiv: 2407.21783.
- [34] Ravi S S, Mielczarek B, Kannappan A, et al. Lynx: An open source hallucination evaluation model[J]. arXiv preprint arXiv:2407.08488, 2024.
- [35] Dang Q A, Ngo C. Reinforcement Learning for Reasoning in Small LLMs: What Works and What Doesn't[J]. arxiv preprint arxiv:2503.16219, 2025.
- [36] pocketpal-ai github Homepage, <https://github.com/a-ghorbani/pocketpal-ai>
- [37] Ji J, Hong D, Zhang B, et al. Pku-saferllhf: Towards multi-level safety alignment for llms with human preference[J]. arXiv preprint arXiv:2406.15513, 2024.
- [38] Liu T, Zhang Y, Brockett C, et al. A token-level reference-free hallucination detection benchmark for free-form text generation[J]. arXiv preprint arXiv:2104.08704, 2021.
- [39] Parambalath G, Mahesh E, Balasubramanian P, et al. Big data analytics: a trading strategy of nse stocks using bollinger bands analysis[C]//Data Management, Analytics and Innovation: Proceedings of ICDMAI 2018, Volume 2. Springer Singapore, 2019: 143-154.
- [40] Kaplan J, McCandlish S, Henighan T, et al. Scaling laws for neural language models[J]. arXiv preprint arXiv:2001.08361, 2020.
- [41] Li J, Luong M T, Jurafsky D. A hierarchical neural autoencoder for paragraphs and documents[J]. arXiv preprint arXiv:1506.01057, 2015.
- [42] Jozefowicz R, Vinyals O, Schuster M, et al. Exploring the limits of language modeling[J]. arXiv preprint arXiv:1602.02410, 2016.
- [43] Frankle J, Carbin M. The lottery ticket hypothesis: Finding sparse, trainable neural networks[J]. arXiv preprint arXiv:1803.03635, 2018.
- [44] Sharma V. A study on data scaling methods for machine learning[J]. International Journal for Global Academic & Scientific Research, 2022, 1(1): 31-42.

- [45] Nguyen T, Van Nguyen C, Lai V D, et al. Culturax: A cleaned, enormous, and multilingual dataset for large language models in 167 languages[J]. arXiv preprint arXiv:2309.09400, 2023.
- [46] Li R, Allal L B, Zi Y, et al. Starcoder: may the source be with you![J]. arXiv preprint arXiv:2305.06161, 2023.

Appendix A Detailed Information on SLMs Used in This Study

Table provides detailed information on the SLMs used in this study, including model names, training data descriptions, and performance rankings on the Hugging Face Leaderboard. In Fig.A1a, the vertical axis represents model parameter size, and the horizontal axis shows the model’s ranking. Within a certain range, larger models generally exhibit stronger performance. Fig.A1b illustrates the distribution of models by provider (vertical axis) and their corresponding rankings (horizontal axis), where each model is labeled by its position on the Hugging Face Leaderboard.

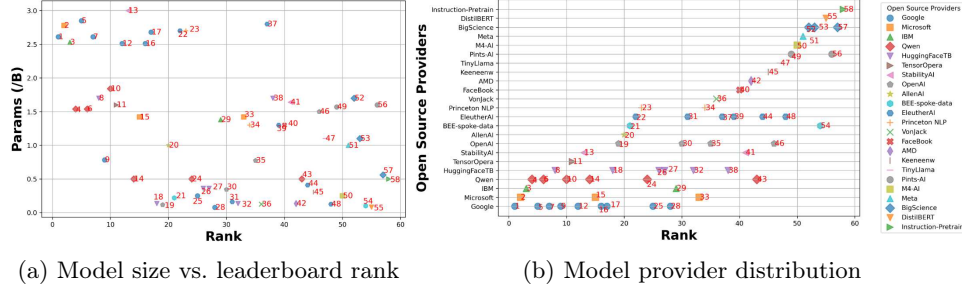


Fig. A1: Model Size and Provider Distribution of Selected SLMs from the Hugging Face Leaderboard

Table A1: Top SLMs with HuggingFace Links and Dataset Types

Rank	Model Name	Link	Dataset
1	google/gemma-2-2b-jpn-it	https://huggingface.co/google/gemma-2-2b-jpn-it	General Text, Code, Mathematical/Logical Content
2	microsoft/phi-2	https://huggingface.co/microsoft/phi-2	General Text, Code, Mathematical/Logical Content
3	ibm-granite/granite-3.0-2b-base	https://huggingface.co/ibm-granite/granite-3.0-2b-base	General Text, Code, Mathematical/Logical Content
4	Qwen/Qwen2.5-1.5B	https://huggingface.co/Qwen/Qwen2.5-1.5B	General Text, Code, Mathematical/Logical Content
5	google/flan-t5-xl	https://huggingface.co/google/flan-t5-xl	General Text, Code, Mathematical/Logical Content

6	Qwen/Qwen2-1.5B	https://huggingface.co/Qwen/Qwen2-1.5B	General Text, Code, Mathematical/Logical Content
7	google/gemma-2-2b	https://huggingface.co/google/gemma-2-2b	General Text, Code, Mathematical/Logical Content
8	HuggingFaceTB/SmolLM2-1.7B	https://huggingface.co/HuggingFaceTB/SmolLM2-1.7B	General Text, Code, Mathematical/Logical Content
9	google/flan-t5-large	https://huggingface.co/google/flan-t5-large	General Text, Code, Mathematical/Logical Content
10	Qwen/Qwen1.5-1.8B	https://huggingface.co/Qwen/Qwen1.5-1.8B	General Text, Code
11	tensoropera/Fox-1-1.6B	https://huggingface.co/tensoropera/Fox-1-1.6B	General Text
12	google/gemma-2b	https://huggingface.co/google/gemma-2b	General Text, Code, Mathematical/Logical Content
13	stabilityai/stablelm-3b-4e1t	https://huggingface.co/stabilityai/stablelm-3b-4e1t	General Text, Code, Mathematical/Logical Content
14	Qwen/Qwen2-0.5B	https://huggingface.co/Qwen/Qwen2-0.5B	General Text, Code, Mathematical/Logical Content
15	microsoft/phi-1_5	https://huggingface.co/microsoft/phi-1_5	General Text, Code
16	google/codegemma-1.1-2b	https://huggingface.co/google/codegemma-1.1-2b	General Text, Code, Mathematical/Logical Content
17	google/recurrent-gemma-2b	https://huggingface.co/google/recurrentgemma-2b	General Text, Code, Mathematical/Logical Content
18	HuggingFaceTB/SmolLM-135M	https://huggingface.co/HuggingFaceTB/SmolLM-135M	General Text, Code
19	openai-community/gpt2	https://huggingface.co/openai-community/gpt2	General Text

20	allenai/OLMo-1B-hf	https://huggingface.co/allenai/OLMo-1B-hf	General Text, Code, Mathematical/Logical Content
21	BEE-spoke-data/smol_llama-220M-GQA	https://huggingface.co/BEE-spoke-data/smol_llama-220M-GQA	General Text, Code, Mathematical/Logical Content
22	EleutherAI/gpt-neo-2.7B	https://huggingface.co/EleutherAI/gpt-neo-2.7B	General Text, Code, Mathematical/Logical Content
23	princeton-nlp/sheared-LLaMA-2.7B	https://huggingface.co/princeton-nlp/sheared-LLaMA-2.7B	General Text, Code
24	Qwen/Qwen2.5-0.5B	https://huggingface.co/Qwen/Qwen2.5-0.5B	General Text, Code, Mathematical/Logical Content
25	google/flan-t5-base	https://huggingface.co/google/flan-t5-base	General Text, Code, Mathematical/Logical Content
26	HuggingFaceTB/SmolLM-360M	https://huggingface.co/HuggingFaceTB/SmolLM-360M	General Text, Code
27	HuggingFaceTB/SmolLM2-360M	https://huggingface.co/HuggingFaceTB/SmolLM2-360M	General Text, Code, Mathematical/Logical Content
28	google/flan-t5-small	https://huggingface.co/google/flan-t5-small	General Text, Code, Mathematical/Logical Content
29	ibm-granite/granite-3.0-1b-a400m-base	https://huggingface.co/ibm-granite/granite-3.0-1b-a400m-base	General Text, Code, Mathematical/Logical Content
30	openai-community/gpt2-medium	https://huggingface.co/openai-community/gpt2-medium	General Text
31	EleutherAI/pythia-160m	https://huggingface.co/EleutherAI/pythia-160m	General Text, Code, Mathematical/Logical Content
32	HuggingFaceTB/SmolLM2-135M	https://huggingface.co/HuggingFaceTB/SmolLM2-135M	General Text, Code, Mathematical/Logical Content

33	microsoft/phi-1	https://huggingface.co/microsoft/phi-1	General Text, Code, Mathematical/Logical Content
34	princeton-nlp/sheared-LLaMA-1.3B	https://huggingface.co/princeton-nlp/sheared-LLaMA-1.3B	General Text, Code
35	openai-community/gpt2-large	https://huggingface.co/openai-community/gpt2-large	General Text
36	vonjack/Mobile LLM-125M-HF	https://huggingface.co/vonjack/MobileLLM-125M-HF	General Text, Code, Mathematical/Logical Content
37	EleutherAI/pythia-2.8b	https://huggingface.co/EleutherAI/pythia-2.8b	General Text, Code, Mathematical/Logical Content
38	HuggingFaceTB/smolLM-1.7B	https://huggingface.co/HuggingFaceTB/smolLM-1.7B	General Text, Code, Mathematical/Logical Content
39	EleutherAI/gpt-neo-1.3B	https://huggingface.co/EleutherAI/gpt-neo-1.3B	General Text, Code, Mathematical/Logical Content
40	facebook/opt-1.3b	https://huggingface.co/facebook/opt-1.3b	General Text
41	stabilityai/stablelm-2-1_6b	https://huggingface.co/stabilityai/stablelm-2-1_6b	General Text, Code
42	amd/AMD-Llama-135m	https://huggingface.co/amd/AMD-Llama-135m	General Text, Code
43	Qwen/Owen1.5-0.5B	https://huggingface.co/Qwen/Owen1.5-0.5B	General Text, Code
44	EleutherAI/pythia-410m	https://huggingface.co/EleutherAI/pythia-410m	General Text, Code, Mathematical/Logical Content
45	keeeeenw/Micro Llama	https://huggingface.co/keeeeenw/MicroLlama	General Text
46	openai-community/gpt2-xl	https://huggingface.co/openai-community/gpt2-xl	General Text

47	TinyLlama/TinyLlama_v1.1	https://huggingface.co/TinyLlama/TinyLlama_v1.1	General Text
48	EleutherAI/gpt-neo-125m	https://huggingface.co/EleutherAI/gpt-neo-125m	General Text, Code, Mathematical/Logical Content
49	pints-ai/1.5-Pints-16K-v0.1	https://huggingface.co/pints-ai/1.5-Pints-16K-v0.1	General Text, Mathematical/Logical Content
50	M4-ai/TinyMistral-248M-v3	https://huggingface.co/M4-ai/TinyMistral-248M-v3	General Text, Mathematical/Logical Content
51	meta-llama/Llama-3.2-1B	https://huggingface.co/meta-llama/Llama-3.2-1B	General Text
52	bigscience/bloom-1b7	https://huggingface.co/bigscience/bloom-1b7	General Text
53	bigscience/bloom-1b1	https://huggingface.co/bigscience/bloom-1b1	General Text
54	BEE-spoke-data/smol_llama-101M-GQA	https://huggingface.co/BEE-spoke-data/smol_llama-101M-GQA	General Text, Code
55	distilbert/distilgpt2	https://huggingface.co/distilbert/distilgpt2	General Text, Code
56	pints-ai/1.5-Pints-2K-v0.1	https://huggingface.co/pints-ai/1.5-Pints-2K-v0.1	General Text, Mathematical/Logical Content
57	bigscience/bloom-560m	https://huggingface.co/bigscience/bloom-560m	General Text
58	instruction-pretrain/InstructLM-500M	https://huggingface.co/instruction-pretrain/InstructLM-500M	General Text, Mathematical/Logical Content

To evaluate training data diversity, we adopt a simple quantification method by assigning scores based on the types of content included in each dataset. The scoring criteria are as follows:

- General text (1 point): Includes standard textual content such as news articles, novels, and social media posts.
- Code (1 point): Includes programming code or technical content such as API documentation or coding tutorials.

- Math/reasoning (1 point): Includes mathematical reasoning, equations, physics problems, or scientific texts involving logical inference.

Table 7 summarizes the training data volume and parameter size of the SLMs used in this study. The model index corresponds to the performance rank listed in Table 6.

Table A2: SLM Training Data Volume and Parameter Sizes (Trillion Tokens & Billion Parameters)

Model Index	Data Volume (Trillion Tokens)	Parameter (Billion)
1	2	2.61
2	1.4	2.78
3	12	2.53
4	18	1.54
5	0.3	2.85
6	3	1.54
7	2	2.61
8	11	1.70
9	0.3	0.783
10	3	1.84
11	3	1.60
12	2	2.51
13	1	3.00
14	12	0.50
15	0.03	1.42
16	1	2.51
17	1	2.68
18	0.25	0.135
19	0.02	0.117
20	3	1.00
21	0.002	0.022
22	0.42	2.70
23	0.005	0.027
24	5.5	0.050
25	0.3	0.248

26	0.25	0.360
27	4	0.360
28	0.3	0.077
29	10	0.138
30	0.02	0.0345
31	0.3	0.160
32	2	0.135
33	0.054	0.142
34	0.005	0.013
35	0.02	0.0774
36	1	0.125
37	0.3	2.80
38	0.25	1.70
39	0.38	1.30
40	0.18	1.30
41	2	1.64
42	0.67	1.35
43	7	0.050
44	0.3	0.041
45	0.05	0.305
46	0.02	0.150
47	1.5	1.10
48	0.3	1.25
49	0.057	1.57
50	0.003	0.248
51	9	1.00
52	0.18	1.70
53	0.18	1.10
54	0.002	1.01
55	0.001	0.82
56	0.057	1.60
57	0.16	0.56

The SecuDevSLM framework examined 58 SLMs in this study, and the training dataset information for each model was collected based on their respective publishers as follows:

A.1 Google

(1) google/gemma-2-2b-jpn-it

- Language: Fine-tuned on Japanese.
- Web Documents: A rich and diverse collection of web texts ensures the model is exposed to a wide range of language styles, topics, and vocabulary, primarily in Japanese.
- Code: Exposure to code helps the model learn programming syntax and patterns, enhancing its ability to generate or understand code-related tasks.
- Mathematics: Training on mathematical texts supports learning in logical reasoning, symbolic representation, and solving mathematical problems.

(2) google/flan-t5-xl

- openai/gsm8k: High-quality, linguistically diverse elementary math word problems.
- cimec/lambada: Narrative passages extracted from a book corpus.
- deepmind/code_contests: Programming problem datasets.

(3) google/gemma-2-2b

- Web Documents: A diverse set of web texts ensures exposure to a broad range of language styles, topics, and vocabulary, primarily in English.
- Code: Code exposure enhances the model’s ability to learn programming languages and improve its performance in code generation or comprehension tasks.
- Mathematics: Mathematical content supports the development of logical reasoning and problem-solving abilities.

(4) google/flan-t5-large

- openai/gsm8k: High-quality, linguistically diverse elementary math problems.
- cimec/lambada: Narrative passages from a book corpus.
- deepmind/code_contests: Programming problems.

(5) google/gemma-2b

- Web Documents: A diverse web text collection enables the model to access a wide range of language styles and topics, mainly in English.
- Code: Code exposure helps the model learn programming syntax and structure.
- Mathematics: Training with math texts aids in logic, symbol manipulation, and solving math-related queries.

(6) google/codegemma-1.1-2b

- Open-source mathematical datasets
- Synthetic code
- Public code repositories

(7) google/recurrentgemma-2b

- Web Documents: Varied web text collections exposing the model to diverse linguistic styles and topics, mostly in English.
- Code: Code exposure enhances learning of programming syntax and tasks.
- Mathematics: Supports reasoning and problem-solving using mathematical texts.

(8) google/flan-t5-base

- openai/gsm8k: High-quality elementary math problems in various languages.
- cimec/lambada: Narratives from a book corpus.
- deepmind/code_contests: Programming challenge data.

(9) google/flan-t5-small

- openai/gsm8k: Multilingual elementary math word problems.
- cimec/lambada: Book-based narrative segments.
- deepmind/code_contests: Programming problems.

A.2 Microsoft

(10) microsoft/phi-2

- NLP: Synthetic texts for natural language processing tasks.
- Code: Python subset from Stack v1.2.
- Q&A: Question-and-answer data from StackOverflow.
- Programming Contests: Code from code_contests programming competitions.
- Synthetic Textbook: Synthetic Python textbooks and exercises generated by gpt-3.5-turbo-0301.

(11) microsoft/phi-1.5

- NLP: Synthetic texts for natural language processing tasks.
- Code: Python subset from Stack v1.2.
- Q&A: Question-and-answer data from StackOverflow.
- Programming Contests: Code from code_contests programming competitions.
- Synthetic Textbook: Synthetic Python textbooks and exercises generated by gpt-3.5-turbo-0301.

(12) microsoft/phi-1

- Code: Python subset from Stack v1.2.
- Q&A: Question-and-answer data from StackOverflow.
- Programming Contests: Code from code_contests programming competitions.
- Synthetic Textbook: Synthetic Python textbooks and exercises generated by gpt-3.5-turbo-0301.

A.3 IBM

(13) **ibm-granite/granite-3.0-2b-base**

- Pretraining Stage 1: Data from academia, internet, enterprise domains (e.g., finance, law), and code.
- Pretraining Stage 2: Curated high-quality data from the same domains, including multilingual and instructional data.

(14) **ibm-granite/granite-3.0-1b-a400m-base**

- Pretraining Stage 1: Data from academia, internet, enterprise domains (e.g., finance, law), and code.
- Pretraining Stage 2: Curated high-quality data from the same domains, including multilingual and instructional data.

A.4 Qwen

(15) **Qwen/Qwen2.5-1.5B**

- Data: Public web documents, encyclopedias, books, and code.
- Focus: Natural language understanding, general question answering, coding, mathematics, scientific knowledge, reasoning, and multilingual ability.
- Strength: Significant enhancements in coding and mathematics.

(16) **Qwen/Qwen2-1.5B**

- Data: Public web documents, encyclopedias, books, and code.
- Focus: Natural language understanding, general question answering, coding, mathematics, scientific knowledge, reasoning, and multilingual ability.

(17) **Qwen/Qwen1.5-1.8B**

- Data: Public web documents, encyclopedias, books, and code.
- Strength: Notable improvement in conversational capabilities.

(18) **Qwen/Qwen2-0.5B**

- Data: Public web documents, encyclopedias, books, and code.
- Focus: Natural language understanding, general question answering, coding, mathematics, scientific knowledge, reasoning, and multilingual ability.

(19) **Qwen/Qwen2.5-0.5B**

- Data: Public web documents, encyclopedias, books, and code.
- Focus: Natural language understanding, general question answering, coding, mathematics, scientific knowledge, reasoning, and multilingual ability.
- Strength: Significant enhancements in coding and mathematics.

(20) **Qwen/Qwen1.5-0.5B**

- Data: Public web documents, encyclopedias, books, and code.
- Strength: Notable improvement in conversational capabilities.

A.5 HuggingFaceTB

(21) HuggingFaceTB/SmolLM2-1.7B

- FineWeb-Edu: Deduplicated educational web samples from FineWeb.
- DCLM: A standardized language corpus.
- The Stack: Code data.
- argilla/Synth-APIGen-v0.1: A set of function-related problems in mathematics, physics, and programming.

(22) HuggingFaceTB/SmolLM-135M

- Cosmopedia v2: Synthetic textbooks and stories generated by Mixtral.
- Python-Edu: Educational Python samples from The Stack.
- FineWeb-Edu: Deduplicated educational web samples from FineWeb.

(23) HuggingFaceTB/SmolLM-360M

- Cosmopedia v2: Synthetic textbooks and stories generated by Mixtral.
- Python-Edu: Educational Python samples from The Stack.
- FineWeb-Edu: Deduplicated educational web samples from FineWeb.

(24) HuggingFaceTB/SmolLM2-360M

- FineWeb-Edu: Deduplicated educational web samples from FineWeb.
- DCLM: A standardized language corpus.
- The Stack: Code data.
- argilla/Synth-APIGen-v0.1: A set of function-related problems in mathematics, physics, and programming.

(25) HuggingFaceTB/SmolLM2-135M

- FineWeb-Edu: Deduplicated educational web samples from FineWeb.
- DCLM: A standardized language corpus.
- The Stack: Code data.
- argilla/Synth-APIGen-v0.1: A set of function-related problems in mathematics, physics, and programming.

(26) HuggingFaceTB/smolLM-1.7B

- Cosmopedia v2: Synthetic textbooks and stories generated by Mixtral.
- Python-Edu: Educational Python samples from The Stack.
- FineWeb-Edu: Deduplicated educational web samples from FineWeb.

A.6 OpenAI

(27) `openai-community/gpt2`

- English Text Corpus: Trained only on raw text to predict the next word in a sentence.
- Web Data: Covers a broad range of topics and domains to ensure diverse knowledge.

(28) `openai-community/gpt2-medium`

- English Text Corpus: Trained only on raw text to predict the next word in a sentence.
- Web Data: Covers a broad range of topics and domains to ensure diverse knowledge.

(29) `openai-community/gpt2-large`

- English Text Corpus: Trained only on raw text to predict the next word in a sentence.
- Web Data: Covers a broad range of topics and domains to ensure diverse knowledge.

(30) `openai-community/gpt2-xl`

- English Text Corpus: Trained only on raw text to predict the next word in a sentence.
- Web Data: Covers a broad range of topics and domains to ensure diverse knowledge.

A.7 EleutherAI

(31) `EleutherAI/gpt-neo-2.7B`

- Pile Dataset: Includes Pile-CC (web), PubMed Central (medical), Books3, OpenWebText2, ArXiv (scientific), GitHub (code), FreeLaw (legal), StackExchange (Q&A), USPTO Backgrounds (patents), PubMed Abstracts, Gutenberg (PG-19), OpenSubtitles, Wikipedia (en), DM Mathematics, Ubuntu IRC (chat logs), BookCorpus2, EuroParl (parliament records), HackerNews, YoutubeSubtitles, PhilPapers (philosophy), NIH ExPorter (research funding), Enron Emails.

(32) `EleutherAI/pythia-160m`

- Pile Dataset: Same sources as EleutherAI/gpt-neo-2.7B.

(33) `EleutherAI/pythia-2.8b`

- Pile Dataset: Same sources as EleutherAI/gpt-neo-2.7B.

(34) `EleutherAI/gpt-neo-1.3B`

- Pile Dataset: Same sources as EleutherAI/gpt-neo-2.7B.

(35) `EleutherAI/pythia-410m`

- Pile Dataset: Same sources as EleutherAI/gpt-neo-2.7B.

(36) `EleutherAI/gpt-neo-125m`

- Pile Dataset: Same sources as EleutherAI/gpt-neo-2.7B.

A.8 StabilityAI

(37) stabilityai/stablelm-3b-4e1t

- tiuuue/falcon-refinedweb: Web data based on CommonCrawl, including raw and processed web content.
- bigcode/starcoderdata: Code datasets.
- allenai/peS2o: Academic papers across biology, chemistry, engineering, computer science, physics, material science, mathematics, psychology, economics, political science, business, geology, sociology, geography, environmental science, art, history, and philosophy.

(38) stabilityai/stablelm-2-1_6b

- tiuuue/falcon-refinedweb: Web data based on CommonCrawl.
- bigcode/starcoderdata: Code datasets.
- uonlp/CulturaX: Multilingual web content.

A.9 Princeton NLP

(39) princeton-nlp/sheared-LLaMA-2.7B

- RedPajama-Data: News articles, academic papers, social media, technical documents, literature, and encyclopedia entries.

(40) princeton-nlp/sheared-LLaMA-1.3B

- RedPajama-Data: Same as (39).

A.10 TensorOpera

(41) tensoropera/Fox-1-1.6B

- 3 trillion web-scraped documents.

A.11 AllenAI

(42) allenai/OLMo-1B-hf

- allenai/dolma includes:
 - Dolma’s CC, Refined Web, C4: Web data
 - StarCoder: Code data
 - Reddit: Social media
 - Semantic Scholar (S2ORC & S2AG), arXiv: Academic papers
 - StackExchange: Q&A forums

- Flan: Multi-task datasets
- CC News: News articles
- OpenWebMath, Algebraic Stack: Math data
- Project Gutenberg: Books
- MegaWika, Wikipedia & Wikibooks: Encyclopedia content

A.12 BEE-spoke-data

(43) BEE-spoke-data/smol_llama-220M-GQA

- JeanKaddour/minipile: A small Pile subset including medical, code, legal, news, subtitles, books, chats, emails.
- mattymchen/refinedweb-3m: Web content.
- pszemraj/simple_wikipedia_LM: NLP-focused data.

(44) BEE-spoke-data/smol llama-101M-G0A

- JeanKaddour/minipile: Same as (43).
- mattymchen/refinedweb-3m: Web content.
- pszemraj/simple_wikipedia_LM: NLP-focused data.

A.13 Meta

(45) meta-llama/Llama-3.2-1B

- Multilingual text
- Public web data

(46) facebook/opt-1.3b

- BookCorpus: Over 10K unpublished books
- CC-Stories: Filtered story subset
- The Pile: Includes Pile-CC, OpenWebText2, USPTO, Gutenberg, OpenSubtitles, Wikipedia, DM Mathematics, HackerNews
- Reddit dataset
- CCNewsV2 dataset

A.14 AMD

(47) amd/AMD-Llama-135m

- cerebras/SlimPajama-627B: News, academic papers, social media, technical docs, literature, encyclopedias
- manu/project_gutenberg: Large collection of ebooks

A.15 VonJack

(48) **vonjack/MobileLLM-125M-HF**

- Public online data

A.16 Keeneenw

(49) **keeeeenw/MicroLlama**

- No code training
- cerebras/SlimPajama-627B: News, academic papers, social media, technical docs, literature, encyclopedias

A.17 TinyLlama

(50) **TinyLlama/TinyLlama_v1.1**

- cerebras/SlimPajama-627B: News, academic papers, social media, technical docs, literature, encyclopedias

A.18 Pints-AI

(51) **pints-ai/1.5-Pints-16K-v0.1**

- HuggingFaceH4/ultrachat_200k: Chat dialogues
- HuggingFaceH4/ultrafeedback_binarized: Prompt-feedback data
- meta-math/MetaMathQA: Math questions

(52) **pints-ai/1.5-Pints-2K-v0.1**

- HuggingFaceH4/ultrachat_200k: Chat dialogues
- HuggingFaceH4/ultrafeedback_binarized: Q&A data
- meta-math/MetaMathQA: Math questions

A.19 M4-AI

(53) **M4-ai/TinyMistral-248M-v3**

- mlfoundations/dclm-baseline-1.0: Large-scale web data
- LLM360/TxT360: Math and physics problems
- Skylion007/openwebtext: Web text

A.20 BigScience

(54) **bigscience/bloom-1b7**

- Trained for text generation, information extraction, Q&A, and summarization.

(55) bigscience/bloom-1b1

- Trained for text generation, information extraction, Q&A, and summarization.

(56) bigscience/bloom-560m

- Trained for text generation, information extraction, Q&A, and summarization.

A.21 DistilBERT

(57) distilbert/distilgpt2

- Multilingual text
- Programming content

A.22 Instruction-Pretrain

(58) instruction-pretrain/InstructLM-500M

- tiuuue/falcon-refinedweb: Web content from CommonCrawl
- instruction-pretrain/general-instruction-augmented-corpora: Instruction-augmented web text
- instruction-pretrain/ft-instruction-synthesizer-collection: Mathematics data