arXiv:2508.15229v1 [cs.CL] 21 Aug 2025

# VocabTailor: Dynamic Vocabulary Selection for Downstream Tasks in Small Language Models

**Hanling Zhang**[1*], **Yayu Zhou**[4*], **Tongcheng Fang** [1], **Zhihang Yuan**[1†], **Guohao Dai** [2,3], **Yu Wang** [1†]

[1]Tsinghua University, [2]Infinigence AI, [3]Shanghai Jiao Tong University, [4]Independent Researcher

## Abstract

Small Language Models (SLMs) provide computational advantages in resource-constrained environments, yet memory limitations remain a critical bottleneck for edge device deployment. A substantial portion of SLMs' memory footprint stems from vocabulary-related components, particularly embeddings and language modeling (LM) heads, due to large vocabulary sizes. Existing static vocabulary pruning, while reducing memory usage, suffers from rigid, one-size-fits-all designs that cause information loss from the prefill stage and a lack of flexibility. In this work, we identify two key principles underlying the vocabulary reduction challenge: the *lexical locality* principle, the observation that only a small subset of tokens is required during any single inference, and the *asymmetry in computational characteristics* between vocabulary-related components of SLM. Based on these insights, we introduce **VocabTailor**, a novel decoupled dynamic vocabulary selection framework that addresses memory constraints through offloading embedding and implements a hybrid static-dynamic vocabulary selection strategy for LM Head, enabling on-demand loading of vocabulary components. Comprehensive experiments across diverse downstream tasks demonstrate that VocabTailor achieves a reduction of up to 99% in the memory usage of vocabulary-related components with minimal or no degradation in task performance, substantially outperforming existing static vocabulary pruning.

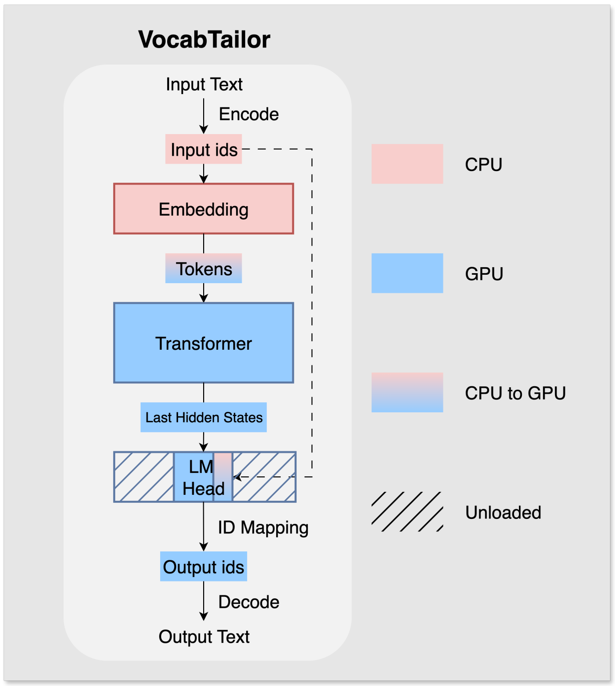


Figure 1: Overview of VocabTailor

## 1 Introduction

Large Language Models (LLMs) (Brown et al. 2020; Touvron et al. 2023a,b; Achiam et al. 2023; Team et al. 2024; Anthropic 2024, 2025; Bai et al. 2023; Guo et al. 2025) have rapidly become foundational to modern AI applications. Recently, increasing attention has turned towards small language models (SLMs), which are better suited for deployment on edge devices and in resource-constrained environments. Despite their compact size, memory still remains a bottleneck, particularly for edge devices with limited GPU memory. A key driver of this bottleneck is the model's vocabulary size, which directly impacts the memory footprint of both the embedding layer and the language modeling (LM) head. For example, in the Llama 3.2 1B model with a 128K-token vocabulary, the embedding and LM head account for over 20% of the total memory usage. As SLMs are scaled down and deployed under tighter memory constraints, vocabulary-related memory inefficiencies become increasingly unsustainable, posing a fundamental barrier to efficient SLM deployment.

To address this, prior work has explored static vocabulary pruning strategies (Ushio, Zhou, and Camacho-Collados 2023; Yang, Cui, and Chen 2022), which reduce vocabulary size by eliminating rare or irrelevant tokens based on curated corpora. While these approaches are well-motivated, they suffer from key limitations due to their static and coupled design, which assumes a single, globally pruned vocabulary is applied on all vocabulary-related components (i.e., tokenizer, embedding, and LM head). This design introduces two major issues:

*Equal contribution.
†Co-corresponding author.

- **Premature Information Loss**: Pruning the tokenizer, embedding, and LM head altogether alters the input representation passed to the transformer. These modified inputs may differ significantly from those seen during pretraining, causing distributional shifts and information loss from the prefill stage. Notably, each pruning step in the pipeline introduces information loss that accumulates, leading to cumulative performance degradation during inference.
- **Lack of flexibility and Adaptability**: The static strategy limits the model's adaptability across diverse tasks. Supporting different task configurations typically requires duplicating multiple copies of vocabulary-related components, resulting in substantial storage overhead and increased complexity in deployment.

Based on empirical observations and theoretical analysis, we derive two key principles for efficient design:

- **Lexical Locality**: In common downstream tasks, generation relies on a highly localized vocabulary—each output depends on a small subset of input tokens and a limited set of task-specific tokens.
- **Computation Asymmetry**: There is computation asymmetry between embedding and LM head. The embedding layer mainly leverages the lookup operation, which is computationally cheap but limited by memory bandwidth. In contrast, the LM head is compute intensive with massive matrix multiplication requiring immense floating point power, making it an ideal task for a GPU. Existing pruning methods ignore such asymmetry, missing opportunities for system-level optimization.

Guided by these principles, we propose **VocabTailor**, a flexible and efficient framework for dynamic vocabulary selection. VocabTailor is based on two main pillars:

- **Decoupled Design**: We adopt a decoupled design for vocabulary-related components. We retain the full tokenizer and offload the embedding layer to CPU memory. Since the embedding lookup is a memory-intensive operation with $\mathcal{O}(\infty)$ computational complexity, we can strategically offload it to free up valuable GPU memory, with minimal overhead to the overall system performance.
- **Hybrid Static-Dynamic Vocabulary Selection**: At runtime, we dynamically select and load input-relevant tokens while maintaining a small, static set of task-specific tokens to ensure stable and efficient computation in the LM head.

This design enables substantial memory savings without compromising input fidelity and model generality. Compared to static pruning, which retains the union of all input and output tokens $(\bigcup \mathcal{I}_i) \bigcup (\bigcup \mathcal{O}_i)$, VocabTailor only needs $\mathcal{I}_i \bigcup \mathcal{T}$ at inference time, where $\mathcal{I}_i$, $\mathcal{O}_i$ are input and output tokens for example $i$, and $\mathcal{T}$ is a small, fixed task-specific token set. Since $\bigcup \mathcal{I}_i \gg \mathcal{I}_i$, this leads to substantial memory savings and improved task adaptability.

In summary, this paper makes the following contributions:

1. We present the first systematic analysis of vocabulary management in LLMs through the lens of *lexical locality* and *computation asymmetry*.
2. We propose **VocabTailor**, a flexible, memory-efficient, and task-adaptive framework that supports a hybrid static-dynamic vocabulary selection strategy along with an enhanced profiling strategy.
3. Across five representative downstream tasks—machine translation, summarization, code completion, information extraction, and math problem solving—VocabTailor reduces memory usage of vocabulary-related components by up to 99%, significantly lowers memory usage with minimal or no performance degradation.

## 2 Related Work

### 2.1 Small Language Model

Small language models (SLMs) are compact alternatives to large language models (LLMs), which are designed for efficiency, lower computational costs, and deployment on resource-constrained devices. While LLMs like GPT-4 (Achiam et al. 2023), LLaMA (Touvron et al. 2023a,b), Claude (Anthropic 2024, 2025), Gemini (Team et al. 2024), Qwen (Bai et al. 2023), and DeepSeek (Guo et al. 2025) have achieved widespread success across real-life applications, SLMs are gaining attention due to their suitability in GPU memory-constrained environments, personal devices, and task-specific scenarios. They offer a scalable, efficient, and sustainable solution tailored for real-time and on-device applications (Lamaakal et al. 2025). With careful selection, small open models can rival and even outperform LLMs while offering improved speed and memory efficiency (Sinha, Jain, and Chadha 2024).

Despite advances in architectures, training techniques, and model compression techniques, SLMs still face challenges, including trade-offs between model size and accuracy, generalization limitations, and concerns over bias and privacy (Van Nguyen et al. 2024; Lamaakal et al. 2025). A common approach to building SLMs is distilling them from LLMs while retaining the same tokenizer and vocabulary. This typically causes vocabulary-related components (i.e., embedding and LM Head) to account for a large proportion of the model's total parameters. This makes vocabulary pruning an effective optimization strategy for SLMs.

### 2.2 Tokenization

Tokenization is a fundamental preprocessing step in Natural Language Processing (NLP) that splits text into smaller units called tokens (e.g., words or characters), which form the input to downstream tasks. Over the years, various tokenization techniques have been developed (Sennrich, Haddow, and Birch 2016; Kudo and Richardson 2018; Devlin et al. 2019). Among these, Byte-Pair Encoding (BPE) (Sennrich, Haddow, and Birch 2016) has become one of the most widely used. Originally developed for data compression, BPE was adapted to tokenize text by iteratively merging the most frequent adjacent symbol pairs starting from individual characters until a target vocabulary size is reached. This approach enables efficient representation of frequent words with fewer tokens while breaking rare or unseen words into informative subword units. BPE's widespread adoption across transformer architectures has established it as a core component of LLM infrastructure.

State-of-the-art LLMs such as GPT-4 (Achiam et al. 2023), LLaMA (Touvron et al. 2023a,b), Gemini (Team et al. 2024), Claude (Anthropic 2024, 2025), and DeepSeek (Guo et al. 2025) rely on BPE-based tokenizers to balance vocabulary efficiency and expressiveness. However, the resulting vocabulary sizes are often large, leading to large embedding matrices and LM heads, which increase computational and memory overhead during inference. This scalability bottleneck has motivated research into vocabulary reduction and adaptive tokenization strategies, especially for resource-constrained deployments.

Importantly, pruning a BPE-based tokenizer can interfere with its learned merge operations governing subword segmentation, resulting suboptimal tokenizations. This disruption reduces the granularity of lexical representations, ultimately impairing model performance on downstream tasks.

### 2.3 Vocabulary Pruning

Vocabulary pruning has emerged as a key area of research in NLP, particularly for scaling and deploying efficient language models. During the BERT era, interest in pruning surged as researchers explored ways to streamline BERT and other transformer-based models. More recently, with the rise of small language models (SLMs), efficient vocabulary selection has once again become a pressing concern.

Vocabulary-trimming (VT) (Ushio, Zhou, and Camacho-Collados 2023) and TextPruner (Yang, Cui, and Chen 2022) were proposed to improve efficiency and reduce model size by removing tokens irrelevant to the target language or rarely seen in downstream tasks. Both methods follow a static pruning strategy: they identify language-specific or task-relevant tokens from a curated corpus and prune the vocabulary accordingly. While this process simultaneously reduces the size of the tokenizer, embedding, and LM head, it introduces premature information loss, leading to cumulative performance degradation, and limited flexibility and adaptability, resulting in substantial memory overhead and increased deployment complexity. These two major limitations underscore the need for more flexible and dynamic vocabulary pruning strategies that are task-specific, adaptable, and generalizable across deployment settings. In response, we propose our framework in Section 3.

## 3 Method

### 3.1 VocabTailor Framework

As illustrated in Fig. 1, VocabTailor decouples vocabulary management across vocabulary-related components. Unlike static vocabulary pruning that uniformly reduces the tokenizer, embedding layer, and LM head, VocabTailor treats each component based on its unique computational and storage properties.

For the **tokenizer**, we retain the full vocabulary to preserve input fidelity and prevent information loss during the prefill stage. This avoids the cumulative performance degradation caused by pruned tokenizers, which often fail to fully capture the input expressiveness.

For the embedding layer and LM head, despite their similar roles(tokenize and detokenize), a fundamental computation asymmetry exists between them.

For the **embedding layer**, which relies on simple lookup operations, is naturally CPU-friendly and incurs negligible runtime overhead when offloaded to CPU memory from high-cost accelerator memory (Yu et al. 2025). Thus, we retain it fully and offload it to CPU.

In contrast, for the **LM head**, which performs compute-intensive matrix multiplications, must remain on GPU for efficient inference. To optimize its memory footprint, we introduce a hybrid static-dynamic vocabulary selection strategy. This approach leverages the lexical locality of tokens and significantly reduces memory usage while preserving downstream task performance and enabling flexible, task-specific adaptation without model duplication.

This decoupled architecture addresses the key limitations of existing approaches while providing a theoretically grounded and practically efficient solution for large-scale vocabulary management. The detailed design and implementation of vocabulary selection strategy are presented in Sections 3.2–3.4.

### 3.2 Motivation: Analysis of Lexical Locality

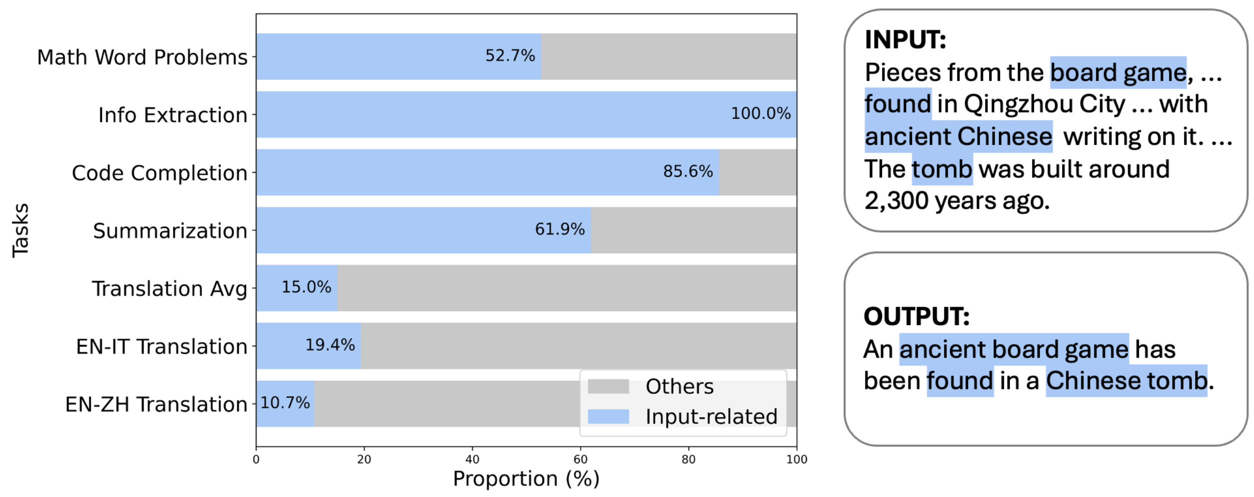
To investigate the essence of efficient vocabulary selection, we analyze input-output pairs across diverse datasets aligned with various downstream tasks. Our empirical analysis reveals two fundamental properties of lexical locality that motivate VocabTailor's dynamic vocabulary selection strategy.

**Observation 1** *Input-Driven Locality: Common downstream tasks exhibit strong input-output lexical overlap—each output contains a small subset of input tokens.*

In various downstream tasks, the model output reuses tokens from the input, as shown in Fig. 2. This phenomenon is particularly pronounced in text extraction tasks (e.g., span-based QA or named entity recognition), where output tokens are typically a subset of the input tokens. In code completion, generated code often replicates variable names, function names, and other identifiers from the input context. Similarly, in text summarization, the generated summary contains large spans from the source document. For instance, in summarization, on average 61.9% of tokens in generated summaries are copied from the input document. From an information-theoretic perspective, the input context dramatically reduces the entropy of the output token distribution, constraining generation to a much smaller effective vocabulary. Thus, preserving input vocabulary is critical for maintaining performance during vocabulary reduction.

**Observation 2** *Task-Driven Locality: Vocabulary required for generation is highly localized—each output depends on a limited set of task-specific tokens.*

Due to input diversity across datasets, the union of all input tokens is significantly larger than the tokens required for any single generation instance. Each input introduces unique



**INPUT:**
Pieces from the board game, … found in Qingzhou City … with ancient Chinese writing on it. … The tomb was built around 2,300 years ago.

**OUTPUT:**
An ancient board game has been found in a Chinese tomb.

Figure 2: Left: Input-output lexical overlap ratio. Right: Example of input-output lexical overlap in summarization task.

tokens, making the aggregate input vocabulary much larger than individual requirements. Crucially, the remaining output tokens—those not found in the corresponding input—form a relatively small, task-specific set $\mathcal{T}$ that captures the essential generation patterns for the task.

Formally, let $\mathcal{I}$ denote the set of all input tokens, $\mathcal{I}_i$ the set of input tokens for instance $i$ in a dataset, $\mathcal{O}$ the set of all output tokens, and $\mathcal{T} \subset \mathcal{O}$ the set of all task-specific tokens in outputs. We observe that:

$$\mathcal{I} = \bigcup_i \mathcal{I}_i \gg \mathcal{I}_i \text{ and } \mathcal{O} > \mathcal{T}$$

These observations expose the fundamental inefficiency of static vocabulary pruning that operates on the union $\mathcal{I} \cup \mathcal{O}$ for every example. VocabTailor exploits this lexical locality by dynamically constructing active per-example vocabularies using only per-example input tokens $\mathcal{I}_i$ and a compact task-specific set $\mathcal{T}$:

$$\mathcal{I} \bigcup \mathcal{O} \gg \mathcal{I}_i \bigcup \mathcal{T}$$

This dynamic targeting of the much smaller $\mathcal{I}_i \cup \mathcal{T}$ enables substantial memory savings without compromising generation quality, as it only retains the tokens necessary for each inference instance.

## 3.3 The Hybrid Vocabulary Selection Strategy

VocabTailor utilizes lexical locality through a hybrid architecture that combines dynamic runtime adaptation with static offline optimization, enabling efficient vocabulary management without sacrificing generation flexibility.

**Dynamic Selection (Runtime Behavior)** At the start of each inference instance, VocabTailor identifies the unique input token set $\mathcal{I}_i$ from the input text and selectively loads the corresponding LM head weight vectors from CPU main memory to GPU. This selective loading exploits the input-driven locality principle, ensuring that only input-relevant vocabulary components are active during generation. Crucially, we overlap this memory transfer with the model's forward propagation, leveraging GPU-CPU parallelism to mitigate runtime overhead.

**Static Selection (Offline Construction)** The static component maintains a compact, task-specific core vocabulary $\mathcal{T}$ that captures essential output tokens independent of input context. While simple frequency-based filtering proves inadequate due to noise from typos and multilingual interference in broad corpora, we introduce a theoretically grounded filtering pipeline (Algorithm 1) that constructs $\mathcal{T}$ through fine-grained analysis.

## 3.4 Fine-grained Construction of the Static Task Vocabulary

Our static vocabulary construction algorithm addresses the challenge of isolating truly task-essential tokens through a three-stage filtering process:

**Input-Aware Filtering** We first exclude all input tokens from the candidate vocabulary, isolating tokens the model must generate without input cues (e.g., function keywords in code generation, discourse markers in summarization). This step directly implements task-driven locality by identifying the irreducible core $\mathcal{T}$ that cannot be derived from the input context.

**Language-Specific Filtering** To suppress noise in mixed-language scenarios (e.g., code datasets with multilingual comments), we apply Unicode block analysis to retain only tokens belonging to the target language family. This heuristic-based approach effectively handles cross-lingual interference while preserving task-relevant vocabulary.

**Tolerance Filtering** When additional vocabulary reduction is required, we formulate vocabulary selection similar to a pruning problem. We define the impact metric $df(v)$ as the document frequency—the fraction of instances where

Algorithm 1: Profiling Strategy

**Definition**:
$\mathcal{V}$: Corpus-profiling vocabulary
$M$: Total number of documents in the corpus
$\mathcal{I}_i$: Set of input tokens corresponding to a single example $i$
$\mathcal{U}$: Set of language-specific tokens obtained from Unicode blocks
$df \in \mathbb{R}$: Dictionary mapping token $v$ to its document frequency $df(v)$
$\tau \in [0, 1]$: Tolerance threshold (fraction of documents allowed to be impacted)
$\mathcal{T}$: Final calibrated task-specific vocabulary
**Input**: $\mathcal{V}$, $\tau$
**Output**: $\mathcal{T}$

```
 1: **Input-Aware Filtering**
 2: $\mathcal{V}_1 \leftarrow \{v \in \mathcal{V} \mid v \notin \mathcal{I}_i\}$
 3: **Language-specific Filtering**
 4: $\mathcal{V}_2 \leftarrow \{v \in \mathcal{V}_1 \mid v \in \mathcal{U}\}$
 5: **Tolerance Filtering**
 6: $N \leftarrow |\mathcal{V}_2|$
 7: $F \leftarrow$ list of $(v, df(v))$ for $v \in \mathcal{V}_2$
 8: Sort $F$ ascending by $df(v)$
 9: cumfreq $\leftarrow 0$, index $\leftarrow 0$
10: for $j \leftarrow 1$ to $N + 1$ do
11:     if sum($F$[: j]) $> \tau M$ then
12:         index $\leftarrow N - j + 1$
13:         break
14:     end if
15: end for
16:
17: return $\mathcal{T} \leftarrow \{v \in F[-\text{index} :]\}$
```

token $v$ appears in the ground truth. Tokens are sorted by ascending $df(v)$ and iteratively removed until the cumulative document frequency of pruned tokens reaches a user-defined tolerance threshold $\tau$. This threshold bounds the worst-case performance drop, where $\tau = 0.01$ ensures that at most 1% of profiling samples lose a critical token.

VocabTailor's hybrid approach bridges theoretical rigor (exploiting lexical locality principles) with practical efficiency (optimizing GPU-CPU memory hierarchy). The static-dynamic decomposition enables substantial memory savings while preserving the model's generational capabilities, providing a scalable solution for deploying SLMs across diverse applications with controllable efficiency-accuracy trade-offs.

## 4 Experiments

### 4.1 Settings

**Tasks and datasets** We evaluate VocabTailor on five representative SLM downstream tasks across diverse domains: machine translation, summarization, code completion, information extraction, and math word problem solving. For machine translation, we involve English-to-Italian and English-to-Chinese translation, as Chinese is logographic with minimal morphology and a large character set, while Italian is alphabetic and morphologically rich. We use the Opus-100 corpus (Zhang et al. 2020) as a profiling dataset and WMT24++ (Deutsch et al. 2025) for the evaluation. For summarization, we use the XSum training set (Narayan, Cohen, and Lapata 2018) to profile and evaluate on its test set. For code completion, the CodeContests$^+$ corpus (Wang et al. 2025) is used for profiling, while evaluation is conducted on SAFIM (Gong et al. 2024). For information extraction, we use the SQuAD (Rajpurkar et al. 2016) training set for profiling and its test set for evaluation. In math problem solving, GSM8K (Cobbe et al. 2021) served as the profiling corpora, with evaluation performed on MAWPS (Koncel-Kedziorski et al. 2016). Each dataset is selected for its strong alignment with the target task.

**Evaluation metrics** We include sacreBLEU (Post 2018), METEOR (Banerjee and Lavie 2005), and COMET (Rei et al. 2020) for machine translation. For summarization, we use Rouge-1, Rouge-2, and Rouge-L scores (Lin 2004). Pass@1 (Chen et al. 2021) is used for code completion. We use F1 score for information extraction, and accuracy for math problem solving. These metrics are standard in the field and provide a robust measure of model performance across the target task.

**Models** For machine translation, we use Qwen3-1.7B (Yang et al. 2025). For summarization and information extraction, we employ Llama 3.2 1B (Dubey et al. 2024). For summarization, the base model is fine-tuned for a better base performance. For code completion, we choose deepseek-coder-1.3b-base (Guo et al. 2024), and for math problem solving, we apply rho-math-1b-interpreter-v0.1 (Lin et al. 2024).

**Baselines and Other Settings** We compare our method (VocabTailor) with the original model (Original) and static vocabulary pruning (VP). Static vocabulary pruning follows the common routines of corpus-based filtering. For a fair comparison, VP and VocabTailor use the same profiling corpra. We set the tolerance threshold $\tau = 0.01$ on all tasks for VocabTailor. As a hybrid dynamic-static framework, the vocabulary size of VocabTailor model varies in each single inference. Here we report the average vocabulary size for each downstream task. For all models, we set the temperature to 0 to avoid randomness.

### 4.2 Results

As shown in Table 1, VocabTailor consistently achieves substantial vocabulary reduction while maintaining or even improving task performance compared with the original model, outperforming static pruning in nearly all cases.

**Machine Translation** In the EN$\rightarrow$ZH task, VocabTailor achieves the best scores across all metrics with SacreBLEU = 15.393, METEOR = 12.692, and COMET = 81.440, while using only 12% of the full vocabulary. These improvements are especially notable given the inherent difficulty of vocabulary pruning in high-character-set languages like Chinese. The results suggest that VocabTailor effectively retains the tokens most essential for both surface-level fluency and deeper semantic adequacy.

| Task | Model | Vocabulary | | Metric |
|---|---|---|---|---|
| Machine Translation: EN→ZH | Original | 151,643 | | 13.475/11.996/81.160 |
| | VP | 64,117 | (42.28%) | 14.840/11.842/81.191 |
| | VocabTailor (Ours) | 18,874 + [40] | (12.47%) | **15.393/12.692/81.440** |
| Machine Translation: EN→IT | Original | 151,643 | | **24.332/48.954**/73.869 |
| | VP | 65,518 | (43.21%) | 22.585/46.735/74.933 |
| | VocabTailor (Ours) | 24,185 + [14] | (15.96%) | 21.127/46.970/**75.493** |
| Summarization | Original | 128,000 | | 0.364/0.147/0.292 |
| | VP | 59,613 | (90.99%) | 0.364/0.147/0.292 |
| | VocabTailor (Ours) | 36,332 + [15] | (55.48%) | 0.364/0.147/0.292 |
| Code Completion | Original | 32,000 | | **54.097%** |
| | VP | 24,888 | (77.78%) | 8.124% |
| | VocabTailor (Ours) | 3,521 + [58] | (11.18%) | 53.865% |
| Information Extraction | Original | 128,000 | | 38.400 |
| | VP | 49,106 | (38.36%) | 2.580 |
| | VocabTailor (Ours) | [105] | (0.08%) | **62.730** |
| Math Problem Solving | Original | 32,000 | | 88.40 |
| | VP | 10,300 | (32.19%) | 87.80 |
| | VocabTailor (Ours) | 5,135 + [14] | (16.09%) | **88.40** |

Table 1: Results on machine translation (sacreBLEU/METEOR/COMET), summarization (Rouge-1/Rouge-2/Rouge-L), code completion (Pass@1), information extraction (F-1), and math problem solving (Accuracy), including the vocabulary size and the ratio to the original model (%). For VocabTailor, vocabulary size consists of task-specific tokens and the average number of dynamic tokens highlighted in brackets. The best results are in bold characters.

In the EN→IT setting, VocabTailor attains the highest COMET score (75.493), indicating strong preservation of semantic and contextual meaning. It retains only 16% of the original vocabulary, yet performs competitively on Sacre-BLEU (21.127) and METEOR (46.970), with modest drops compared to the unpruned model. In contrast, VP removes less vocabulary (43%) but still causes moderate degradation across all metrics, suggesting that static pruning may eliminate rare but task-relevant tokens. VocabTailor's ability to adapt to Italian's rich inflectional morphology further reinforces the general applicability of the method.

**Summarization** For summarization, all three models yield identical ROUGE scores, indicating no difference in summary quality. However, VocabTailor achieves this with just 55% of the original vocabulary, 35% fewer tokens than VP. These results demonstrate that aggressive vocabulary reduction is possible without compromising output quality when apply a hybrid dynamic-static vocabulary selection.

**Code Completion** On the SAFIM benchmark with deepseek-coder-1.3b-base, VP leads to a dramatic drop in performance: Pass@1 falls from 54.097% to 8.124%, despite a relatively modest vocabulary reduction (down to 77.78% of the original size). This highlights a key limitation of corpus-based pruning for code: essential elements such as variable names and identifiers may be discarded if they appear infrequently in the profiling corpus. In contrast, VocabTailor retains only 11.18% of the vocabulary and yet achieves a high Pass@1 of 53.865%. These findings underscore the robustness of our input- and task-specific vocabulary selection strategy for generation-intensive tasks such as code synthesis.

**Information Extraction** For information extraction tasks, we evaluate on the SQuAD dataset using LLaMA 3.2 1B. VocabTailor achieves the most striking result: with just 0.08% of the original vocabulary retained (and no static tokens at all), it attains an F1 score of 62.73, outperforming both the Original (38.4) and VP (2.58) by large margins. This result stems from the nature of extractive tasks, where the output vocabulary is typically a subset of the input. Because VocabTailor preserves input tokens dynamically, it retains all the necessary vocabulary for accurate extraction. The poor performance of VP suggests that static, corpus-based pruning is poorly suited for tasks where input-output overlap is high, whereas VocabTailor is especially effective.

**Math Word Problem Solving** For symbolic reasoning, we evaluate on a math problem-solving task. The unpruned model achieves a score of 88.40, which is fully preserved under VocabTailor, even though it reduces the vocabulary to just 16% of the original. VP, while also reducing vocabulary size to 32%, causes a slight performance drop to 87.80. This shows that even in tasks requiring high-precision symbolic handling, our approach remains effective.

Across five distinct tasks, VocabTailor consistently demonstrates strong performance while substantially reducing vocabulary size. In many cases, it matches or even exceeds the performance of the unpruned model. Compared to static pruning, which often compromises accuracy, VocabTailor reduces vocabulary more aggressively (up to 99%)

without sacrificing model quality. These findings support that the dynamic vocabulary selection is a practical and efficient approach to deploying SLMs in resource-constrained environments.

## 4.3 Ablation Study

To understand the contributions of each component in our framework, we conduct a series of ablation experiments on the SAFIM dataset using the deepseek-coder-1.3b-base model for the code completion task. We primarily focus on evaluating the impact of different vocabulary configurations, including the static and dynamic components, as well as our proposed three-stage filtering process.

**Impact of Dynamic vs. Static Vocabulary Components** We first evaluate the impact of the dynamic and static vocabulary components, both individually and in combination. As shown in Table 2, using only the dynamic part that contains tokens profiled from the specific input examples, results in a significant performance drop (Pass@1 of 36.064%). This demonstrates that input tokens alone lack the broader coverage needed for robust code completion. Using only the static part (task-specific tokens) achieves a Pass@1 of 52.3%. However, this still underperforms the full static-dynamic configuration, which achieves the best result at 53.865% with nearly the same vocabulary size. This indicates that while the static tokens carry most of the task-relevant capacity, including the dynamic tokens adds critical input-specific nuances, and their combination is essential for optimal performance.

We also compare our static-only approach with VP: VP retains 78% of the original vocabulary—more than our approach—it results in a drastically lower Pass@1 of 8.124%. This stark contrast underscores a key insight: while both VP and our static-only setup are static, VP's direct modification of the tokenizer and embeddings damages input representations, leading to severe degradation. In contrast, our method retains the full tokenizer and embedding, thereby preserving representational integrity and maintaining high performance even with significantly fewer tokens.

| Model | Vocabulary | Pass@1 |
|---|---|---|
| Original | 100% | 54.097% |
| VP | 77.80% | 8.124% |
| Dynamic + Static ($\tau = 0.01$) | 11.18% | 53.865% |
| Dynamic only | 0.81% | 36.064% |
| Static only ($\tau = 0.01$) | 11.00% | 52.300% |

Table 2: Comparison of the dynamic and static components in VocabTailor.

**Impact of Input-aware and Language-specific Filtering** As discussed earlier, VocabTailor calibrates the static vocabulary through three-stage filtering. As shown in Table 3, starting from the unfiltered static vocabulary (78% of the original), applying input-aware filtering (IA) reduces the size to 53% with virtually no performance loss (54.074% vs. 54.062%). Adding language-specific filtering (LS) further reduces the vocabulary size to 46%, while performance slightly improves to 54.085%. This improvement likely stems from the removal of noisy or irrelevant tokens from multilingual corpora, allowing the model to focus on task-relevant representations. These results demonstrate that IA and LS can significantly compress the vocabulary without degrading accuracy, validating the effectiveness of our static token selection process.

| Model | Vocabulary | Pass@1 |
|---|---|---|
| Original | 100.00% | 54.097% |
| Dynamic + Unfiltered static | 77.78% | 54.074% |
| Dynamic + IA | 52.85% | 54.062% |
| Dynamic + IA + LS | 45.61% | 54.085% |

Table 3: Comparison of input-aware filtering (IA) and language-specific filtering (LS) in VocabTailor.

**Impact of Tolerance Filtering** Lastly, we analyze the effect of tolerance filtering, which allows for further reduction of rarely activated tokens based on cumulative document frequency. In Table 4, we vary the tolerance threshold ($\tau$) to observe the trade-off between vocabulary size and accuracy. At $\tau = 0$, we preserve all profiled tokens, achieving a Pass@1 of 54.085%. As tolerance increases, more tokens are filtered out, reducing the vocabulary to as low as 2.5% of the original size ($\tau = 0.10$), with a gradual decline in performance. Importantly, even with $\tau = 0.01$, the vocabulary shrinks to 11.18% with only 1.8% drop in Pass@1, suggesting that our method is robust to aggressive reduction. This highlights the flexibility of tolerance as a tuning knob to balance compression vs. accuracy.

| Model | Vocabulary | Pass@1 |
|---|---|---|
| $\tau = 0$ | 45.61% | 54.085% |
| $\tau = 0.01$ | 11.18% | 53.865% |
| $\tau = 0.02$ | 7.28% | 53.714% |
| $\tau = 0.10$ | 2.50% | 52.277% |

Table 4: Comparison of different tolerance thresholds in VocabTailor.

## 5 Conclusion

In this paper, we propose a flexible and efficient vocabulary selection framework effectively reduce memory usage during SLM inference. By identifying and leveraging lexical locality together with the computation asymmetry, our method reduce up to 99% in the memory usage of vocabulary-related components of SLM while maintain the performance on representative downstream tasks.

## 6 Limitations & Future Work

The proposed framework presented in this paper is only explored on language downstream tasks of SLMs. This method may be extended and applied to VLMs, ALMs, and MLLMs for memory-efficient inference on image/video understanding and audio generation tasks. Custom kernel implementations and hardware-specific optimizations can be integrated into the framework for further inference optimization.

## References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Anthropic. 2024. The Claude 3 Model Family: Opus, Sonnet, Haiku. Technical report.

Anthropic. 2025. Introducing Claude 4. https://www.anthropic.com/news/claude-4.

Bai, J.; Bai, S.; Chu, Y.; Cui, Z.; Dang, K.; Deng, X.; Fan, Y.; Ge, W.; Han, Y.; Huang, F.; et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Banerjee, S.; and Lavie, A. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 65–72. Ann Arbor, Michigan: Association for Computational Linguistics.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Deutsch, D.; Briakou, E.; Caswell, I.; Finkelstein, M.; Galor, R.; Juraska, J.; Kovacs, G.; Lui, A.; Rei, R.; Riesa, J.; et al. 2025. Wmt24++: Expanding the language coverage of wmt24 to 55 languages & dialects. *arXiv preprint arXiv:2502.12404*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Burstein, J.; Doran, C.; and Solorio, T., eds., *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186. Minneapolis, Minnesota: Association for Computational Linguistics.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv e-prints*, arXiv–2407.

Gong, L.; Wang, S.; Elhoushi, M.; and Cheung, A. 2024. Evaluation of LLMs on Syntax-Aware Code Fill-in-the-Middle Tasks. In *International Conference on Machine Learning*, 15907–15928. PMLR.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Guo, D.; Zhu, Q.; Yang, D.; Xie, Z.; Dong, K.; Zhang, W.; Chen, G.; Bi, X.; Li, Y.; et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196*.

Koncel-Kedziorski, R.; Roy, S.; Amini, A.; Kushman, N.; and Hajishirzi, H. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, 1152–1157.

Kudo, T.; and Richardson, J. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

Lamaakal, I.; Maleh, Y.; El Makkaoui, K.; Ouahbi, I.; Pławiak, P.; Alfarraj, O.; Almousa, M.; and Abd El-Latif, A. A. 2025. Tiny language models for automation and control: Overview, potential applications, and future research directions. *Sensors*, 25(5): 1318.

Lin, C.-Y. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, 74–81.

Lin, Z.; Gou, Z.; Gong, Y.; Liu, X.; Shen, Y.; Xu, R.; Lin, C.; Yang, Y.; Jiao, J.; Duan, N.; et al. 2024. Rho-1: Not all tokens are what you need. *arXiv preprint arXiv:2404.07965*.

Narayan, S.; Cohen, S. B.; and Lapata, M. 2018. Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. arXiv:1808.08745.

Post, M. 2018. A Call for Clarity in Reporting BLEU Scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, 186–191. Belgium, Brussels: Association for Computational Linguistics.

Rajpurkar, P.; Zhang, J.; Lopyrev, K.; and Liang, P. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. arXiv:1606.05250.

Rei, R.; Stewart, C.; Farinha, A. C.; and Lavie, A. 2020. COMET: A Neural Framework for MT Evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2685–2702.

Sennrich, R.; Haddow, B.; and Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. In Erk, K.; and Smith, N. A., eds., *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1715–1725. Berlin, Germany: Association for Computational Linguistics.

Sinha, N.; Jain, V.; and Chadha, A. 2024. Are Small Language Models Ready to Compete with Large Language Models for Practical Applications? *arXiv preprint arXiv:2406.11402*.

Team, G.; Georgiev, P.; Lei, V. I.; Burnell, R.; Bai, L.; Gulati, A.; Tanzer, G.; Vincent, D.; Pan, Z.; Wang, S.; et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ushio, A.; Zhou, Y.; and Camacho-Collados, J. 2023. Efficient Multilingual Language Model Compression through Vocabulary Trimming. In Bouamor, H.; Pino, J.; and Bali, K., eds., *Findings of the Association for Computational Linguistics: EMNLP 2023*, 14725–14739. Singapore: Association for Computational Linguistics.

Van Nguyen, C.; Shen, X.; Aponte, R.; Xia, Y.; Basu, S.; Hu, Z.; Chen, J.; Parmar, M.; Kunapuli, S.; Barrow, J.; et al. 2024. A survey of small language models. *arXiv preprint arXiv:2410.20011*.

Wang, Z.; Liu, S.; Sun, Y.; Li, H.; and Shen, K. 2025. CodeContests+: High-Quality Test Case Generation for Competitive Programming. *arXiv preprint arXiv:2506.05817*.

Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yang, Z.; Cui, Y.; and Chen, Z. 2022. TextPruner: A Model Pruning Toolkit for Pre-Trained Language Models. In Basile, V.; Kozareva, Z.; and Stajner, S., eds., *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 35–43. Dublin, Ireland: Association for Computational Linguistics.

Yu, D.; Cohen, E.; Ghazi, B.; Huang, Y.; Kamath, P.; Kumar, R.; Liu, D.; and Zhang, C. 2025. Scaling Embedding Layers in Language Models. arXiv:2502.01637.

Zhang, B.; Williams, P.; Titov, I.; and Sennrich, R. 2020. Improving massively multilingual neural machine translation and zero-shot translation. *arXiv preprint arXiv:2004.11867*.