# KNN-SSD: Enabling Dynamic Self-Speculative Decoding via Nearest Neighbor Layer Set Optimization

Anonymous ACL submission


## Abstract

Speculative Decoding (SD) has emerged as a widely used paradigm to accelerate the inference of large language models (LLMs) without compromising generation quality. It works by efficiently drafting multiple tokens using a compact model and then verifying them in parallel using the target LLM. Notably, Self-Speculative Decoding proposes skipping certain layers to construct the draft model, which eliminates the need for additional parameters or training. Despite its strengths, we observe in this work that drafting with layer skipping exhibits significant sensitivity to domain shifts, leading to a substantial drop in acceleration performance. To enhance the domain generalizability of this paradigm, we introduce KNN-SSD, an algorithm that leverages K-Nearest Neighbor (KNN) search to match different skipped layers with various domain inputs. We evaluated our algorithm in various models and multiple tasks, observing that its application leads to $1.3\times\sim1.6\times$ speedup in LLM inference.

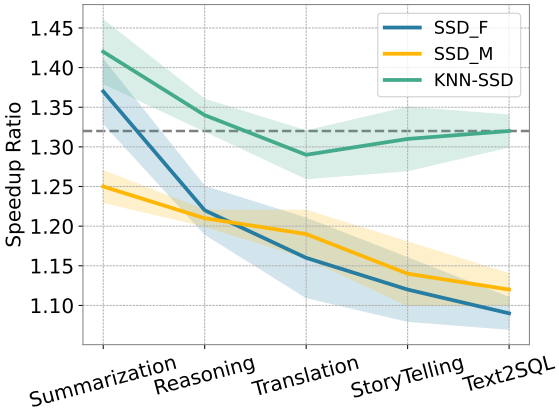




Figure 1: Average speedup results under task-by-task sample streams. The dashed line represents the average speedup ratio achieved by KNN-SSD. Results indicate that our KNN-SSD can achieve a stable speedup while Self-SD methods' speedups decline, as they are sensitive to domain shifts.

## 1 Introduction

Large language models (LLMs) have proven highly capable in handling various downstream tasks (Touvron et al., 2023; OpenAI et al., 2024; Yang et al., 2025). However, the token-by-token generation in autoregressive decoding results in quadratic computational complexity, which presents significant efficiency challenges, particularly as the model size increases. To address this challenge, speculative decoding (SD) has been proposed as a promising solution for lossless acceleration of LLM inference (Xia et al., 2023; Leviathan et al., 2023; Chen et al., 2023). At each decoding step, SD uses a lightweight draft model to efficiently predict multiple tokens, which are then verified in parallel by the target LLM to preserve the original output distribution. To enhance inference efficiency while preserving output quality, SD aims to simultaneously minimize latency and maximize accuracy (Xia et al., 2024; Hu et al., 2025). Achieving this requires carefully managing the inherent trade-off between the speed of the drafting process and the correctness of the speculative predictions.

Recent works in SD have yielded significant improvements in the latency-accuracy balance by employing diverse techniques, such as integrating lightweight draft models into LLMs (Ankner et al., 2024; Zhang et al., 2025) or aligning a small model with a larger one (Kim et al., 2023; Bachmann et al., 2025) for speculative generation. However, these approaches inevitably require additional models, which increase the total number of parameters and introduce additional training complexity.

Self-SD (Zhang et al., 2024) addresses this concern by constructing a compact draft model through the selective skipping of certain layers. However, in this work, we find that **the selection of skipped layers is not universal** and one skip-layer configuration could be sensitive to domain shifts. For example, when applying a configuration derived from the summarization task to other tasks, as shown

in Figure 1, we observe a significant reduction in speedup from 1.35× to less than 1.10×, highlighting the need for *domain-specific adaptation*.

To tackle this issue, we propose KNN-SSD, a method that dynamically adjusts skip-layer configurations based on domain representations. The key goal of KNN-SSD is to *optimize skipped layers specific to each domain, simulate realistic input scenarios, and accurately identify the domain of each sample*. To achieve this goal, KNN-SSD integrates three main features: (1) a skipped layer set optimization process for the specific domain of samples, (2) an input sample stream designed to simulate real-life user inputs better, and (3) a KNN model using LLM's last hidden representations to identify the domain of input samples. We evaluated our method using LLaMA-2 series (Touvron et al., 2023) and Qwen-2.5 series (Yang et al., 2025) across various tasks, including summarization, reasoning, translation, storytelling, and text-to-SQL. The results indicate that KNN-SSD achieves a 1.3×∼1.6× speedup compared to autoregressive decoding. This approach maintains over **80**% token acceptance rate across the LLaMA-2 series and over **99**% token acceptance rate across the Qwen-2 series, indicating high alignment potential between the draft model and the target LLM. Further analysis validated the effectiveness of KNN-SSD across out-of-domain sample inputs and one dataset that contains various types of samples.

To summarize, our key contributions are:

1. We introduce KNN-SSD, a self-speculative decoding algorithm with a fine-grained skipped layer set selection, which adopts k-nearest neighbor search to retrieve a suitable skipped layer set for each input sample;

2. To evaluate our method, we design a dynamic input data stream that contains samples from diverse domains, and KNN-SSD can achieve a 1.3×∼1.6× speedup across different models without changing the generated tokens' distribution.

## 2 Related Work

**Speculative Decoding (SD).** Speculative Decoding (SD) aims to accelerate autoregressive text generation in LLMs without compromising output quality (Xia et al., 2023; Leviathan et al., 2023). It reduces decoding latency by predicting multiple future tokens using a draft model or internal mechanisms, followed by verification and correction by the target LLM. Existing strategies include aligning small draft models with large models (Xia et al., 2023; Kim et al., 2023; Bachmann et al., 2025), using a retrieval-based model from a datastore (He et al., 2024; Gritta et al., 2025) or predicting $k$ tokens in parallel (Cai et al., 2024; Wen et al., 2024). In another line of work, plug-and-play methods have been examined, with examples including appending pseudo tokens (Fu et al., 2024) and skipping layers dynamically (Metel et al., 2024; Xia et al., 2025) during inference. Despite efficiency improvement, these methods often rely on auxiliary models or sub-optimal choices, hindering scalability and effectiveness. The most related methods to our work include Self-SD (Zhang et al., 2024) and LayerSkip (Elhoushi et al., 2024), which also construct draft models by skipping intermediate LLM layers. However, both approaches are trained on a single data type and struggle with diverse data streams. Our work aims to tackle this problem by integrating samples from various domains.

**Sparsity and Model Compression.** Sparsity and model compression are essential for enhancing the efficiency of LLMs by reducing active parameters or computations during inference (Hu et al., 2022). Common approaches include parameter pruning (Frantar and Alistarh, 2023; Ashkboos et al., 2024; Sun et al., 2024), knowledge distillation (Huang et al., 2022; Gu et al., 2024; Wu et al., 2024), and quantization (Yao et al., 2022; Liu et al., 2023; Park et al., 2024), which compress models while preserving performance. Structured sparsity methods, such as layer skipping (Liu et al., 2024; Bhendawade et al., 2024; Xia et al., 2025) and dynamic sparsification, further enhance efficiency by adapting computation to input characteristics. While these works aim to optimize computational workloads, they may sacrifice performance by using sub-optimal choices because of insufficient search in the layer space. In contrast, our KNN-SSD method can always find optimal choices to accelerate LLM inference losslessly.

## 3 Background

### 3.1 Self-Speculative Decoding

Unlike traditional SD methods that require an auxiliary draft model, Self-Speculative Decoding (Self-SD) leverages the LLM's internal structure to draft tokens by selectively skipping certain layers (Zhang et al., 2024). Given data $x_1, \ldots, x_n$ and the target LLM $\mathbf{M}$ with $L$ layers including both

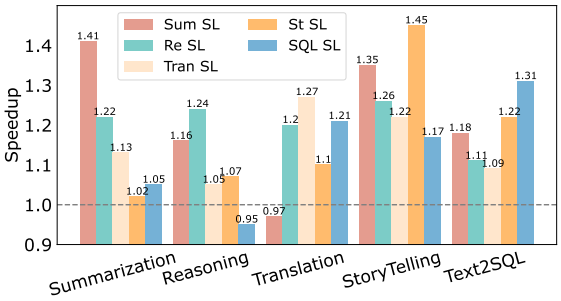



Figure 2: Different tasks have different optimal skip layer sets. "Sum SL" denotes the skip layer set optimized for the Summarization task.

attention and MLP layers, Self-SD aims to find an optimal $z \in \{0,1\}^L$, where $z^{(i)} = 1$ indicates that the $i$th layer needs to be skipped and vice versa. A black-box function $f(\cdot)$ is used to assess the average inference time per verified token:

$$z^* = \arg\min_z f\big(\mathbf{M}(z)|x_1, \ldots, x_n\big). \quad (1)$$

Self-SD applies Bayesian optimization (Jones et al., 1998) to identify an optimal skip layer set by iteratively selecting new $z$ based on a Gaussian process and evaluating with Eq.1. After a specified number of iterations, the best $z$ is considered an approximation of $z^*$ and is fixed for inference. During decoding, the selected layers are skipped to efficiently generate draft tokens, which are then validated in parallel by the full-parameter LLM to ensure the output distribution remains unchanged.

### 3.2 Preliminary Study

While Self-SD improves inference efficiency, the optimal layers to skip vary significantly across different tasks. To demonstrate this, we analyze the performance of SD across multiple representative tasks, including summarization, reasoning, storytelling, translation, and text-to-SQL. As shown in Figure 2, an optimized skip-layer configuration for one task does not generalize well to others. For example, a configuration that accelerates summarization degrades performance in reasoning tasks.

These results show that the static skip-layer configuration is suboptimal. This limits its effectiveness, particularly in real-world scenarios where query types are unpredictable. To achieve both high inference efficiency and minimal performance degradation, task-specific configurations are essential. This motivates the development of KNN-SSD, which dynamically selects the most suitable skip-layer configuration based on task characteristics, ensuring robust and efficient speculative decoding across diverse tasks.

## 4 Methodology

We introduce KNN-SSD, a generalizable Self-SD method designed to improve inference efficiency while maintaining adaptability across diverse tasks. Figure 3 shows our method of accelerating inference. It first generates enough last hidden vectors for each task during the pre-inference process. Then, a fixed number of vectors are selected as representative anchors to fit a KNN model. For each task, its optimal skip layer set is searched using a Bayesian optimization process. In the inference process, a new input data will find its cluster using the previous KNN model, and the corresponding skip layer set will be used for the target LLM. Finally, we perform the standard Self-SD process, which contains two stages of drafting and verification to accelerate inference. By integrating these two processes, KNN-SSD provides a flexible and effective solution to accelerate LLM inference in real-world applications.

### 4.1 Pre Inference

Given a set of domains $D_1, \ldots, D_n$, we first randomly sample multiple instances from each domain, denoted as $d_{i1}, \ldots, d_{im}$ for domain $D_i$. Each sampled instance $d_{ij}$ is then passed through a pre-trained LLM $\mathbf{M}$ to obtain its last hidden vector representation $v_{ij}$. These samples are then aggregated and clustered into $n$ groups $\mu_1, \ldots, \mu_n$ using the K-means algorithm, where the number of clusters is set to match the number of domains. For each cluster $\mu_i$, we identify $k$ representative anchors based on their distance to the cluster centroid. The collection of selected anchors for cluster $\mu_i$ is denoted as $A_i = \{a_{i1}, \ldots, a_{ik}\}$, which will be used to fit a KNN model. The construction of the anchor set $A_i$ is formally defined as follows:

$$A_i = \arg\min_{S \subseteq D_i, |S|=k} \sum_{v_{ij} \in S} \|v_{ij} - \mu_i\|. \quad (2)$$

Subsequently, for each domain $D_i$, we utilize the anchor set $\{a_{i1}, \ldots, a_{ik}\}$ to determine a domain-specific skip layer set $z_i \in \{0,1\}^L$, where $L$ denotes the total number of layers in the language model $\mathbf{M}$. Each element $z_i^{(j)}$ indicates whether the $j$-th layer should be skipped ($z_i^{(j)} = 1$) or retained ($z_i^{(j)} = 0$) during inference. To identify the optimal configuration $z_i$, we employ Bayesian

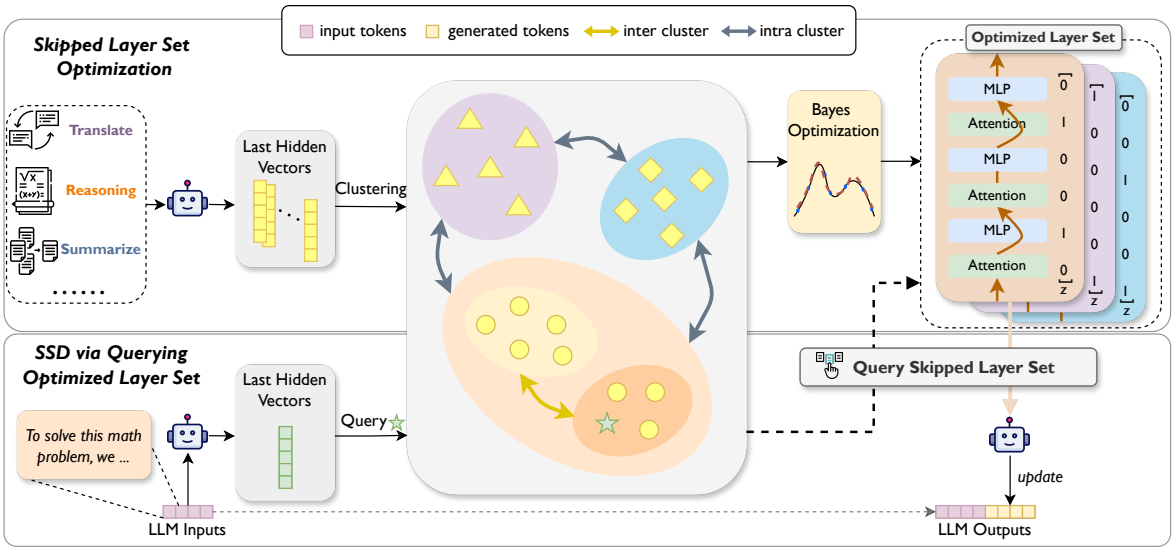



Figure 3: Layer skipping and KNN process in KNN-SSD. Before LLM-related generation, KNN-SSD first performs **(a) Layer set Searching Optimization.** For each task, KNN-SSD generates a task-specific skip layer set and stores it in a configuration file; **(b) Generate Anchor representatives.** KNN-SSD then produces last hidden vectors for each task to fit a KNN model. When a new sample is input, KNN-SSD first uses its last hidden vector as the **input representative** and queries the KNN model. Based on the retrieved result, it selects the corresponding skip layer set to perform decoding, thereby achieving acceleration.

Optimization (Jones et al., 1998) over the space of binary layer masks, aiming to minimize an objective black-box function $f(\cdot)$ that measures the average inference time per verified token:

$$z_i^* = \arg\min_{z_i} f\big(\mathbf{M}(z_i)|a_{i1}, \ldots, a_{ik}\big). \quad (3)$$

All $z_1^*, \ldots, z_n^*$ will be stored for future use and not be changed.

## 4.2 Inference

For a newly arrived sample $s$, we first extract its last hidden vector $v$ from the model. We then perform a KNN search based on cosine similarity between the hidden vector of $s$ and all representative anchors. This process yields a corresponding domain label, effectively classifying the sample into one of the known domains $i^*$. Based on the identified domain, we apply its associated optimal skip-layer configuration $z_{i^*}^*$ to $\mathbf{M}$ to accelerate inference:

$$i^*, j^* = \arg\max_{i,j} \frac{v \cdot a_{ij}}{\|v\| \cdot \|a_{ij}\|}, \quad (4)$$

$$\text{Domain}(s) = i^*, \quad (5)$$

$$\mathbf{M} \leftarrow z_{i^*}^*. \quad (6)$$

We then perform the standard Self-SD process (Zhang et al., 2024), which involves two stages: drafting and verification. During the drafting stage, the LLM uses the previously selected skip-layer configuration $z_i$ as a draft model $M(z_i)$ to generate a sequence of draft tokens:

$$y' = \arg\max_{y} \log P(y \mid x,\ \mathbf{y};\ M(z_i)), \quad (7)$$

where $x$ and $\mathbf{y}$ denote input and output generated by LLM, respectively, and $y'$ represents the token produced by the autoregressive process. In the verification stage, the full LLM verifies the draft tokens in a single forward pass. This step validates the correctness of the generated tokens and either accepts them or triggers a re-drafting if discrepancies are found.

To better simulate real-world task streams, we introduce the mix ratio $r$, which denotes the probability that the next input sample belongs to a different task than the current one. A mix ratio of 0 corresponds to a task-by-task input stream, where all consecutive samples come from the same task. In contrast, a mix ratio of 1 indicates maximum task mixing, where every two consecutive samples are from different tasks. As the mix ratio grows, the frequency of domain shift increases.

$$P(s_{i+1} \in D_j \mid s_i \in D_k) = \begin{cases} \frac{r}{N-1} & \text{if } j \neq k, \\ 1 - r & \text{if } j = k. \end{cases} \quad (8)$$

| Models | Methods | Speed (token/s) | Overall $\mathbb{E}$(Spd.) |
|---|---|---|---|
| LLaMA-2-13B | Vanilla | 13.62 | 1.00× |
| | SSD_F | 16.34 | 1.20× |
| | SSD_M | 16.88 | 1.24× |
| | KNN-SSD | **19.61** | **1.44**× |
| LLaMA-2-13B -Chat | Vanilla | 13.22 | 1.00× |
| | SSD_F | 14.67 | 1.11× |
| | SSD_M | 15.33 | 1.16× |
| | KNN-SSD | **17.85** | **1.35**× |
| Qwen-2.5-14B | Vanilla | 11.16 | 1.00× |
| | SSD_F | 14.06 | 1.26× |
| | SSD_M | 15.40 | 1.38× |
| | KNN-SSD | **18.08** | **1.62**× |
| Qwen-2.5-14B -Instruct | Vanilla | 10.79 | 1.00× |
| | SSD_F | 12.84 | 1.19× |
| | SSD_M | 13.49 | 1.25× |
| | KNN-SSD | **16.30** | **1.51**× |

Table 1: Comparison between KNN-SSD and two Self-SD methods. We report average decoding speed (token/s) under greedy decoding and average speedup ratio among four different mix ratios. More details are provided in Appendix C.3.

| Models | Methods | $r$=0.0 $\mathbb{E}$(Spd.) | $r$=0.3 $\mathbb{E}$(Spd.) | $r$=0.7 $\mathbb{E}$(Spd.) | $r$=1.0 $\mathbb{E}$(Spd.) |
|---|---|---|---|---|---|
| LLaMA -2-13B | Vanilla | 1.00× | 1.00× | 1.00× | 1.00× |
| | SSD_F | 1.24× | 1.21× | 1.19× | 1.17× |
| | SSD_M | 1.23× | 1.27× | 1.24× | 1.23× |
| | KNN-SSD | **1.42**× | **1.45**× | **1.43**× | **1.45**× |
| Qwen -2.5-14B | Vanilla | 1.00× | 1.00× | 1.00× | 1.00× |
| | SSD_F | 1.25× | 1.23× | 1.27× | 1.28× |
| | SSD_M | 1.40× | 1.36× | 1.39× | 1.38× |
| | KNN-SSD | **1.60**× | **1.64**× | **1.63**× | **1.61**× |

Table 2: Our experiments across four mix ratio settings (0.0, 0.3, 0.7, and 1.0) reveal that SSD_F performs well only in the early stages, but suffers degradation as the input distribution changes. SSD_M, which applies a unified sub-optimal configuration, performs consistently worse. In contrast, KNN-SSD maintains stable speedup across all mix ratios, validating its robustness to distribution shifts.

# 5 Experiments

## 5.1 Experimental Setup

**Implementation Details.** We mainly evaluate KNN-SSD on LLaMA-2 series (Touvron et al., 2023) and Qwen-2.5 series (Yang et al., 2025) across various tasks, including summarization, mathematical reasoning, storytelling, translation, and text-to-SQL. The evaluation datasets include CNN/Daily Mail (CNN/DM) (Nallapati et al., 2016), GSM8K (Cobbe et al., 2021), TinyStories (Eldan and Li, 2023), Wmt16 DE-EN (Wmt16) (Bojar et al., 2016), and Spider2 (Lei et al., 2025), where more details can be found in Appendix B.

For each dataset, we used Bayesian optimization [1] (BO) to perform 1,000 iterations in 8 representative samples in search of the optimal skip-layer configuration. As detailed in Appendix A, the optimal layer configurations are task-specific. The representative samples are selected via the K-means algorithm from all last hidden vectors generated by the LLM in the corresponding dataset, ensuring optimal coverage of the feature space. The maximum generation lengths on CNN/DM, GSM8K, Wmt16, Spider2, and TinyStories are set to 128, 128, 128, 128, and 512, respectively. We conduct 1-shot evaluation for CNN/DM and TinyStories, 3-shot evaluation for Spider2, and 5-shot evaluation for GSM8K and Wmt16. For each dataset, we extracted the most representative $k = 10$ hidden vectors from the last hidden layer across all data samples using cosine similarity to serve as anchor points for the KNN model, following the same approach as introduced earlier in the BO framework. For each new input sample, we also compute the cosine similarity between its last hidden vector and the anchors, and assign it to the task of its nearest neighbor. Detailed setups are presented in Appendix C.1.

**Baselines.** In our primary experiments, we compared KNN-SSD and Self-SD approach (Zhang et al., 2024) to assess their effectiveness. For the Self-SD method, we primarily simulated two scenarios. In the first scenario, a fixed skip-layer configuration was determined based on the first sample in the task stream and remained unchanged throughout the process, which is denoted as SSD_F. In the second scenario, the skip-layer configuration was adjusted by re-performing BO according to the task distribution within the stream, and the newly searched configuration was subsequently applied for inference and also remained unchanged, which is denoted as SSD_M.

**Evaluation Metrics.** We evaluate KNN-SSD using two standard metrics commonly adopted in evaluation: the mean generated length $M$ (Stern et al., 2018) and the token acceptance rate $\alpha$ (Leviathan et al., 2023). Beyond these, we also report the expected decoding throughput in tokens per second, along with the expected wall-time speedup

[1] https://github.com/bayesian-optimization/BayesianOptimization

| Models | Methods | $M$ | $\alpha$ | Speedup |
|---|---|---|---|---|
| LLaMA-2 -13B | Vanilla | 1.00 | - | 1.00× |
| | SSD_F | 2.17 | 0.62 | 1.10× |
| | SSD_M | 2.53 | 0.68 | 1.14× |
| | KNN-SSD | **3.12** | **0.88** | **1.34**× |
| LLaMA-2 -13B-Chat | Vanilla | 1.00 | - | 1.00× |
| | SSD_F | 1.97 | 0.57 | 1.04× |
| | SSD_M | 2.14 | 0.59 | 1.09× |
| | KNN-SSD | **2.87** | **0.85** | **1.28**× |

Table 3: The results demonstrate the mean accepted tokens, token acceptance rate, and actual speedup ratio obtained from our tests on the LLaMA-2 series, conducted under a mix ratio of 1.0 across the five main datasets (CNN/DM, GSM8K, Wmt16, Spider2, and TinyStories), showing that KNN-SSD outperforms two Self-SD methods in every metric.

ratio compared to standard autoregressive decoding. Given $M$ and $\alpha$, the expected speedup can be derived by the formula given by Leviathan et al. (2023):

$$\mathbb{E}(\text{Spd.}) = \frac{M\alpha}{(M-1)(1-s)+\alpha} \tag{9}$$

where $s$ denotes the ratio of skipped layers. More details about this formula are discussed in Appendix C.2.

### 5.2 Main Result

We compare KNN-SSD with two Self-SD methods on generation tasks: Table 1 reports the average speedup, while Table 2 details the results across different configurations. In our experiments, we evaluate KNN-SSD under four settings: mix ratio = 0, 0.3, 0.7, and 1 separately, with 40 samples from five datasets each, 200 samples in total. The experimental results demonstrate the following findings: (1) KNN-SSD shows superior efficiency over prior methods, achieving consistent speedups of 1.35×∼1.62× over vanilla autoregressive decoding across various models. (2) The mix ratio of sample flows doesn't affect the speedup of KNN-SSD. The speedup remains stable, which indicates that KNN-SSD can handle various samples in a more realistic scenario.

We present the mean accepted tokens, acceptance rate, as well as the actual speedup of LLaMA-2-13B series in Table 3 and Figure 4, which further validates the superiority of KNN-SSD over Self-SD.

### 5.3 Analysis

**Pre-inference Overhead.** The only additional overhead introduced by KNN-SSD is the storage of the hidden states and the execution of the KNN search, which is negligible. Specifically, when the maximum decoding length is set to 128, the average generation latency across the five datasets on LLaMA-2-13B is approximately 7.5 seconds, of which storing the last hidden state and conducting the KNN search takes around 0.18 seconds, accounting for only about 2.5% of the total. Under longer decoding lengths such as 512 or 1024 tokens, the average generation time increases to about 23.2 seconds or 43.7 seconds, while the KNN search time remains roughly constant, resulting in a proportion of just 0.77% or 0.41%. Meanwhile, the classification accuracy for all datasets is 100%, as shown in Figure 5. This indicates that our KNN-based domain partitioning yielded perfectly accurate clustering in these cases.

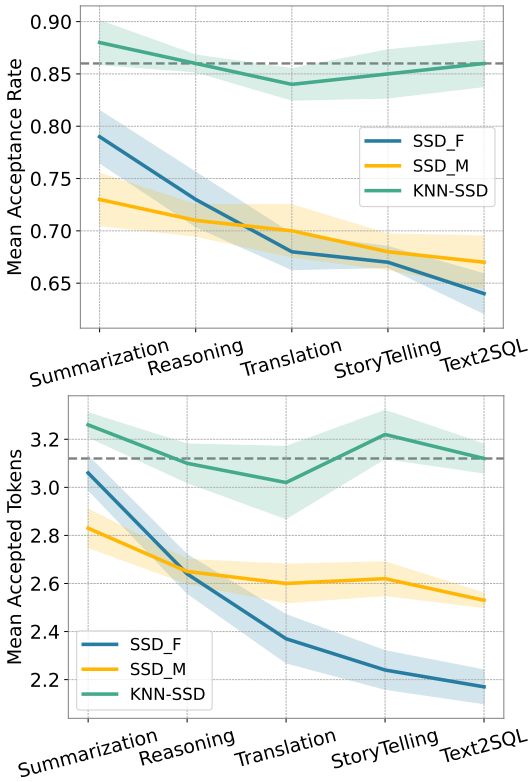



Figure 4: The mean accepted tokens and mean acceptance rate under task-by-task sample streams. The dashed lines represent the average length and rate achieved by KNN-SSD across all five datasets.

**Inter & Intra.** We use the MATH (Hendrycks et al., 2021) dataset to assess the capabilities of KNN-SSD in a single dataset with multiple domains. In the MATH dataset, math questions are categorized into seven types. Thus, using one specific skip layer set for this dataset is insufficient, and we introduce a fine-grained clustering to handle this mixed

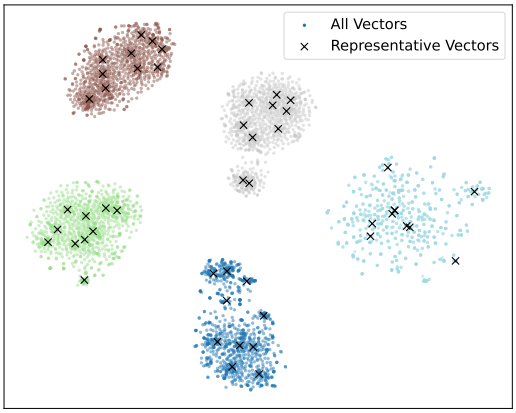



Figure 5: Visualization of last hidden vectors from five domains of samples. Results show these vectors can be clearly divided into five clusters. From each cluster, we selected ten vectors as representative anchors for our KNN model.

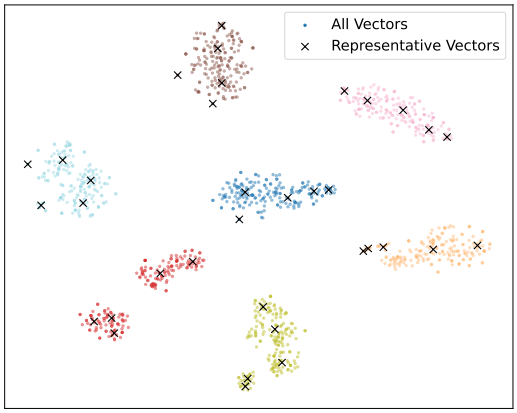



Figure 6: Visualization of 700 last hidden vectors from the MATH dataset using t-SNE method. It is clear to see that all vectors can be categorized into 7 groups, which aligns with the fact that the MATH dataset has 7 different kinds of problems.

| **Methods** | $M$ | $\alpha$ | **Speedup** |
|---|---|---|---|
| Vanilla | 1.00 | - | 1.00× |
| SSD_F | 1.54 | 0.51 | 0.97× |
| SSD_M | 1.82 | 0.59 | 1.02× |
| KNN-SSD | **2.37** | **0.81** | **1.23**× |

Table 4: Results of MATH dataset using LLaMA-2-13B.

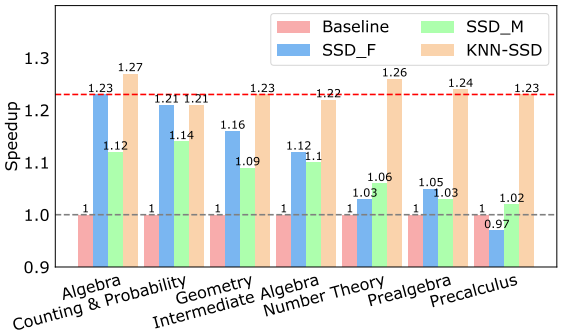



Figure 7: Speedup results for task-by-task sample streams on MATH dataset under three methods. While KNN-SSD maintains a speedup of around 1.25×, two Self-SD methods decline as the number of domains grows.

domain. Figure 6 shows that each type of math question can be clustered into a single group. Table 4 indicates the speedup result for each method, where we can clearly see that KNN-SSD outperforms Self-SD methods and achieves a speedup of 1.23× and a mean generated length $M$ of 2.37.

Figure 7 visualizes speedups on task-by-task settings. SSD_F achieves high performance only on the first subtask and declines on the rest of the subtasks; while KNN-SSD has a better speedup than the other two Self-SD methods.

**Out of Domain Generalization.** We adopt the XSUM (Narayan et al., 2018) dataset, the MATH (Hendrycks et al., 2021) dataset, and the Alpaca (Taori et al., 2023) dataset as out-of-domain tasks to assess KNN-SSD's generalizability. XSUM and CNN/DM datasets belong to summarization tasks, whereas MATH and GSM8K datasets involve reasoning-based tasks. Therefore, although we did not search for their respective optimal skip layer sets for XSUM and MATH in our experiments, it is reasonable that KNN-SSD would assign XSUM samples to CNN/DM and thus adopt CNN/DM's optimal skip layer set, and the same applies to MATH samples.

Compared to these two datasets, the Alpaca dataset contains more diverse instruction-answer pairs across summarization, reasoning, grammar, and many other tasks. Results indicate that although some of the domains are not covered by our five datasets in the main experiments, our method can still assign an unknown sample to its most similar domain and thus achieve inference acceleration. As shown in Table 5, our experimental results demonstrate that the model achieves an approximately 1.15× $\sim$ 1.25× speedup under the KNN-SSD method, even without prior search.

**Number of Clusters.** Table 6 shows the result of the influence of cluster numbers. We conducted experiments on the Alpaca dataset as it covers a variety of domains, using K-means clustering with varying numbers of clusters. As shown in the

| Datasets | Methods | $M$ | $\alpha$ | Speedup |
|---|---|---|---|---|
| **XSUM** | Vanilla | 1.00 | - | 1.00× |
| | Self-SD | 1.42 | 0.56 | 0.99× |
| | KNN-SSD | **2.51** | **0.84** | **1.24**× |
| **MATH** | Vanilla | 1.00 | - | 1.00× |
| | Self-SD | 1.34 | 0.48 | 0.93× |
| | KNN-SSD | **2.13** | **0.76** | **1.17**× |
| **Alpaca** | Vanilla | 1.00 | - | 1.00× |
| | Self-SD | 1.26 | 0.43 | 0.92× |
| | KNN-SSD | **1.95** | **0.67** | **1.15**× |

Table 5: Results of out-of-domain datasets using LLaMA-2-13B-Chat. No representative anchor of these three domains is generated.

results, the speedup effect improves as the number of clusters increases, eventually surpassing the speedup ratio observed in the out-of-domain experiments (Table 5). However, when the cluster count exceeds 5 (e.g., up to 7), the speedup plateaus, indicating that partitioning Alpaca into five clusters is sufficient—further subdivision yields no additional gains.

| Num. | $M$ | $\alpha$ | Speedup |
|---|---|---|---|
| 1 | 1.86 | 0.65 | 1.05× |
| 3 | 2.20 | 0.75 | 1.17× |
| 5 | 2.52 | 0.80 | 1.23× |
| 7 | 2.55 | 0.81 | 1.23× |

Table 6: Results of the Alpaca dataset among different numbers of clusters using LLaMA-2-13B-Chat. **Num.** denotes the number of clusters.

**Case Study.** To better illustrate how our method works, we provide a case study that presents a typical sample stream. In Figure 8, a sample stream contains three common types of queries a user might ask: summarization, reasoning, and translation. For each input query, KNN-SSD will first compute its last hidden vector and then use a KNN model to find its optimal skipped layer set. The typical speculative decoding will be conducted with draft and verification steps, where blue and red tokens indicate that they are generated separately in draft and verification steps. By constantly changing skipped layer sets, KNN-SSD achieves a stable speedup compared to other methods that use a static strategy, which is insufficient for diverse inputs.

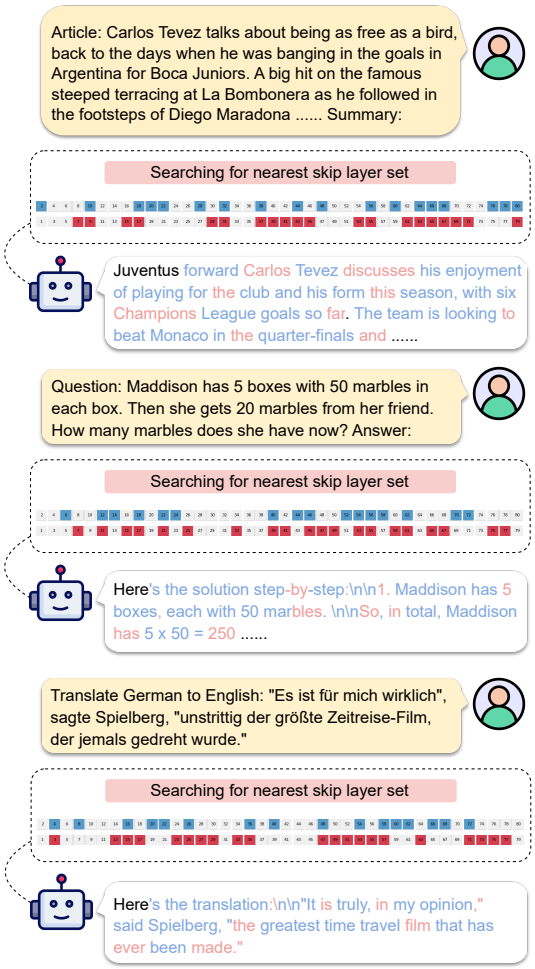



Figure 8: Case study of how KNN-SSD works. Blue tokens indicate that they are generated during the drafting step and verified by the model, while red tokens indicate they are generated by prediction from the verification step. Squares in red and blue indicate skipped attention layers and MLP layers, respectively.

## 6 Conclusion

In this work, we introduce KNN-SSD, an algorithm that leverages K-Nearest Neighbor search to match suitable skipped layers for various domain inputs. KNN-SSD is designed to find an optimal skipped layer set for each domain of data, which accelerates LLM's inference losslessly. To assess its ability, we define a mix ratio of a sample stream, indicating how frequently the domain changes. We conducted extensive experiments with various LLMs and mix ratios and found that KNN-SSD can achieve a speedup of around 1.3×∼1.6× without changing the ordinary distribution of the generated tokens. Our in-depth analysis indicates that a single dataset may also contain mixed domains. Furthermore, KNN-SSD can achieve a 1.2× speedup on out-of-domain datasets, showing its great potential in handling various data streams in real-life scenarios.

## Limitations

A few limitations need to be considered while our KNN-SSD achieves a notable speedup on various models. First, we did not incorporate draft tree verification, which has been shown to improve the token acceptance rate (Xia et al., 2025). Second, our current evaluation is limited to models of moderate scale. Due to practical considerations related to computational resources, we have not yet extended our method to larger-scale models. We leave these directions for future work.

## Ethics Statement

The datasets used in our experiment are publicly released and labeled through interaction with humans in English. In this process, user privacy is protected, and no personal information is contained in the dataset. The scientific artifacts that we used are available for research with permissive licenses. And the use of these artifacts in this paper is consistent with their intended use. Therefore, we believe that our research work meets the ethics of ACL.

## References


Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. 2024. Hydra: Sequentially-dependent draft heads for medusa decoding. *Preprint*, arXiv:2402.05109.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicegpt: Compress large language models by deleting rows and columns. *Preprint*, arXiv:2401.15024.

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas Kohler. 2025. Judge decoding: Faster speculative sampling requires going beyond model alignment. *Preprint*, arXiv:2501.19309.

Nikhil Bhendawade, Irina Belousova, Qichen Fu, Henry Mason, Mohammad Rastegari, and Mahyar Najibi. 2024. Speculative streaming: Fast llm inference without auxiliary models. *Preprint*, arXiv:2402.11131.

Ond rej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *Preprint*, arXiv:2401.10774.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *Preprint*, arXiv:2302.01318.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Ronen Eldan and Yuanzhi Li. 2023. Tinystories: How small can language models be and still speak coherent english? *Preprint*, arXiv:2305.07759.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. 2024. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642, Bangkok, Thailand. Association for Computational Linguistics.

Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of LLM inference using lookahead decoding. In *Forty-first International Conference on Machine Learning*.

Milan Gritta, Huiyin Xue, and Gerasimos Lampouras. 2025. Dresd: Dense retrieval for speculative decoding. *Preprint*, arXiv:2502.15572.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. Minillm: Knowledge distillation of large language models. *Preprint*, arXiv:2306.08543.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. 2024. Rest: Retrieval-based speculative decoding. *Preprint*, arXiv:2311.08252.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *Preprint*, arXiv:2103.03874.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Yunhai Hu, Zining Liu, Zhenyuan Dong, Tianfan Peng, Bradley McDanel, and Sai Qian Zhang. 2025. Speculative decoding and beyond: An in-depth survey of techniques. *Preprint*, arXiv:2502.19732.

Yukun Huang, Yanda Chen, Zhou Yu, and Kathleen McKeown. 2022. In-context learning distillation: Transferring few-shot learning ability of pre-trained language models. *Preprint*, arXiv:2212.10670.

Donald R Jones, Matthias Schonlau, and William J Welch. 1998. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13:455–492.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. 2023. Speculative decoding with big little decoder. In *Advances in Neural Information Processing Systems*, volume 36, pages 39236–39256. Curran Associates, Inc.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *Preprint*, arXiv:2411.07763.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Yijin Liu, Fandong Meng, and Jie Zhou. 2024. Accelerating inference in large language models with a unified layer skipping strategy. *Preprint*, arXiv:2404.06954.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023. Llm-qat: Data-free quantization aware training for large language models. *Preprint*, arXiv:2305.17888.

Michael R. Metel, Peng Lu, Boxing Chen, Mehdi Rezagholizadeh, and Ivan Kobyzev. 2024. Draft on the fly: Adaptive self-speculative decoding using cosine similarity. *Preprint*, arXiv:2410.01028.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, et al. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. 2024. Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models. *Preprint*, arXiv:2206.09557.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. A simple and effective pruning approach for large language models. *Preprint*, arXiv:2306.11695.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Zhuofan Wen, Shangtong Gui, and Yang Feng. 2024. Speculative decoding with ctc-based draft model for llm inference acceleration. In *Advances in Neural Information Processing Systems*, volume 37, pages 92082–92100. Curran Associates, Inc.

Minghao Wu, Abdul Waheed, Chiyu Zhang, Muhammad Abdul-Mageed, and Alham Fikri Aji. 2024. LaMini-LM: A diverse herd of distilled models from large-scale instructions. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 944–964, St. Julian's, Malta. Association for Computational Linguistics.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3909–3925, Singapore. Association for Computational Linguistics.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. 2025. SWIFT: On-the-fly self-speculative decoding for LLM inference acceleration. In *The*

*Thirteenth International Conference on Learning Representations*.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7655–7671, Bangkok, Thailand. Association for Computational Linguistics.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In *Advances in Neural Information Processing Systems*, volume 35, pages 27168–27183. Curran Associates, Inc.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2024. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11263–11282, Bangkok, Thailand. Association for Computational Linguistics.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. 2025. Learning harmonized representations for speculative sampling. In *The Thirteenth International Conference on Learning Representations*.

## A Preliminary Details

We visualize the optimal skipped layer sets we searched across five tasks on two series of models in Figure 9 and Figure 10.

## B Datasets

We mainly evaluate KNN-SSD on LLaMA-2 (Touvron et al., 2023) series and Qwen-2.5 (Yang et al., 2025) series across diverse tasks, which encompass most real-life scenarios. We select five different datasets, covering summarization, mathematical reasoning, translation, storytelling, and text-to-SQL, which are CNN/Daily Mail (CNN/DM) (Nallapati et al., 2016), GSM8K (Cobbe et al., 2021), TinyStories (Eldan and Li, 2023), Wmt16 DE-EN (Wmt16) (Bojar et al., 2016), and Spider2 (Lei et al., 2025) datasets, respectively. The maximum generation lengths on CNN/DM, GSM8K, Wmt16, Spider2, and TinyStories are set to 128, 128, 128, 128, and 512, respectively. We conduct 1-shot evaluation for CNN/DM and TinyStories, 3-shot evaluation for Spider2, and 5-shot evaluation for GSM8K and Wmt16. For further analysis, we also use the XSUM (Narayan et al., 2018) dataset, the MATH (Hendrycks et al., 2021) dataset, and the Alpaca (Taori et al., 2023) dataset for the summarization, mathematical reasoning, and instruction following tasks, respectively.

**CNN/DM** The CNN/Daily Mail dataset is a large-scale benchmark for abstractive text summarization. It consists of long news articles paired with short summaries, derived from the CNN and Daily Mail websites. The dataset is used to evaluate the performance on long-form input and coherent summary generation.

**GSM8K** GSM8K is a high-quality benchmark dataset for arithmetic reasoning, consisting of grade school math word problems and their detailed step-by-step solutions. It is used to evaluate the reasoning and problem-solving capabilities in mathematical contexts.

**TinyStories** TinyStories is a dataset of short, synthetically generated children's stories designed to support research on language modeling and narrative understanding. The stories are simple in structure and vocabulary, making the dataset suitable for studying controlled text generation.

**Wmt16** The WMT16 De-En dataset is a standard benchmark for machine translation, consisting of parallel German-English sentence pairs collected from various sources. It is used to evaluate the translation quality of models.

**Spider2** Spider 2.0 is a complex and cross-domain text-to-SQL benchmark designed to evaluate the ability of models to generate executable SQL queries from natural language questions. It includes diverse databases and query types, requiring models to generalize to unseen schemas and handle intricate reasoning.

**XSUM** XSUM is an abstractive summarization dataset consisting of BBC news articles paired with single-sentence summaries, in contrast to CNN/DM, which provides longer, multi-sentence summaries for news articles. It emphasizes concise and information-rich summaries, testing the models' ability to extract key information.

**MATH** The MATH dataset is a benchmark for mathematical problem solving, comprising high school-level competition problems with detailed step-by-step solutions. It covers a wide range of topics, including algebra, counting and probability, geometry, intermediate algebra, number theory, prealgebra, and precalculus, and is designed to evaluate the advanced reasoning and symbolic manipulation abilities of language models.

**Alpaca** The Alpaca dataset is a collection of instruction-following demonstrations generated using the self-instruct method, based on the outputs of a strong language model. It covers a wide range of tasks, making it suitable for us to test the generalizability of KNN-SSD.

## C Experimental Details

### C.1 Setups

During the pre-inference stage, we set the maximum iterations of Bayesian Optimization to 1,000 and the number of samples to 8. For each dataset, we first randomly choose 1,000 last hidden vectors, then we use the K-means algorithm to find 10 representatives as anchors for the KNN model.

In the inference process, experiments were conducted on 8×NVIDIA RTX 3090 GPU (24GB) and 4×NVIDIA RTX A6000 GPU (40GB) with CUDA 12.0, and an Intel(R) Xeon(R) Gold 5117 CPU with 14 cores. Pytorch and Huggingface transformers package are used to perform both baselines and our method. The temperature and top_p are set to 0.2

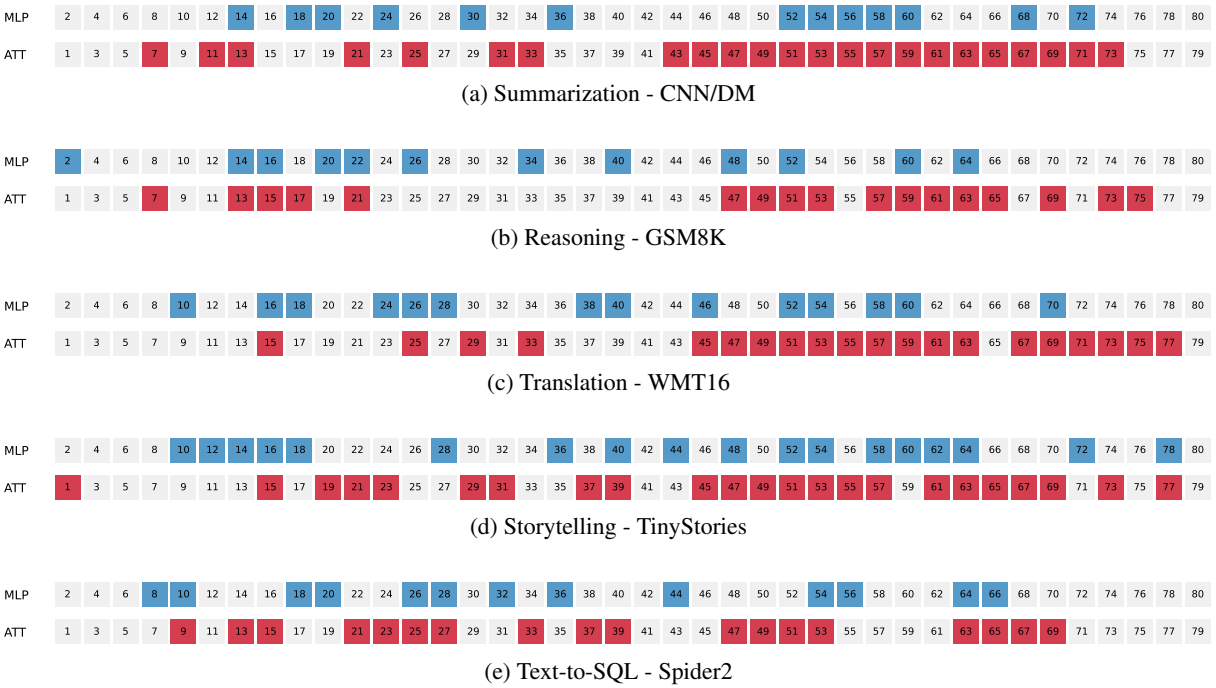



Figure 9: Visualization of skipped layer set configuration of LLaMA-2-13B optimized by Self-SD (Zhang et al., 2024) on different task domains. Gray squares indicate retained layers, red squares denote skipped attention layers, and blue squares signify skipped MLP layers.

and 0.85, respectively. KV cache is used during experiments.

### C.2 Evaluation Metrics

We further demonstrate the two main metrics we used in the main experiments. The mean accepted length $M$ denotes the average number of output tokens produced by the target LLM during each forward pass. The token acceptance rate $\alpha$ refers to the ratio of tokens that are accepted by the target LLM to the total number of draft steps, which showcases the expectation of whether the target LLM accepts a token generated by the draft models. Given $M$ and $\alpha$, the expected wall-time speedup can be derived as follows:

$$\mathbb{E}(\text{Speedup}) = \frac{M\alpha}{(M-1)c+\alpha} \tag{10}$$

where $c$ is defined as the *cost efficient* in Leviathan et al. (2023). It represents the ratio of the draft model's required time to the target model's during a single forward pass, which is adapted from Eq.6 in Xia et al. (2025). In the Self-SD method, we define $c = 1 - s$, where $s$ represents the proportion of skipped layers to total layers, or skip ratio, which is visualized in Appendix A, as the draft model only needs to process the retained layers.

The formulation in Eq.10 can be made more precise by considering the difference in latency between MLP and Attention layers. The *cost efficient* $c$ is consequently redefined as:

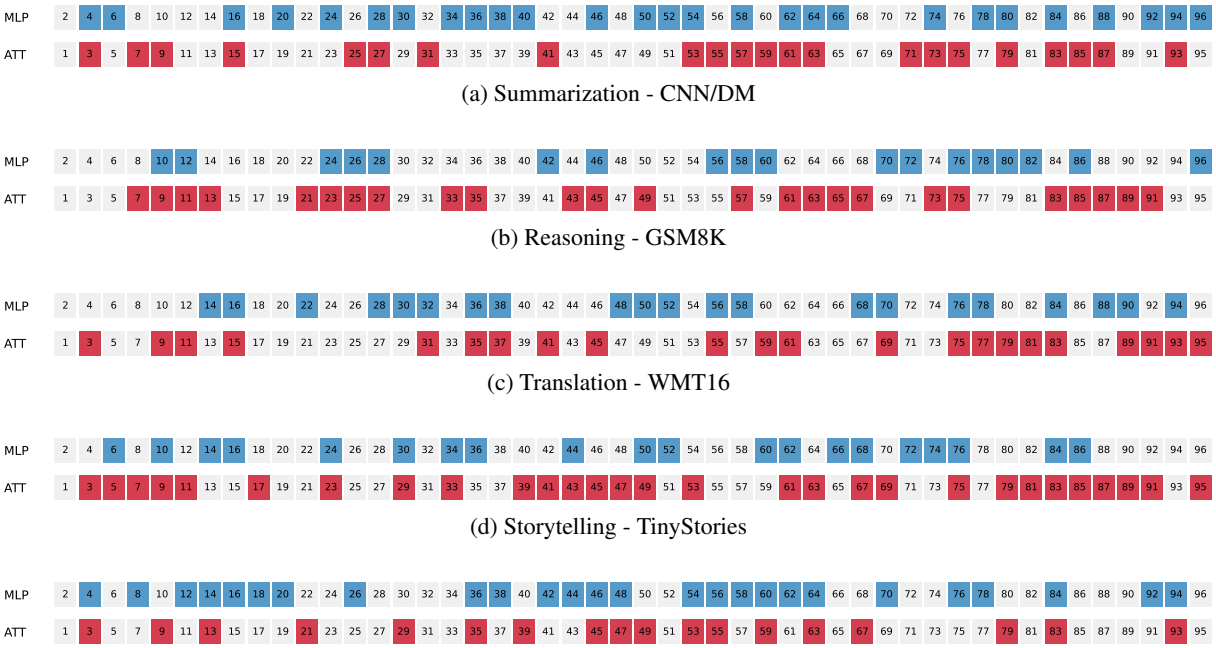
$$c = \frac{(1 - s_{\text{mlp}}) + (1 - s_{\text{attn}}) \times \beta}{1 + \beta} \tag{11}$$

where $s_{\text{mlp}}$ and $s_{\text{attn}}$ are the skipped layer ratios for MLP and Attention layers, respectively, while $\beta$ represents the latency ratio of Attention to MLP layers. When $\beta = 1$, Eq.11 reduces to the original $1 - s$ form.

In our experiments, using the LLaMA-2-13B model in our environment, we empirically measured the latency ratio $\beta$ to be approximately 2.3. For example, when the overall skipped layer ratio is $s = 0.45$, and the separate skipping ratios are $s_{\text{mlp}} = 0.42$ and $s_{\text{attn}} = 0.48$, the corresponding values of $c$ computed using the refined formula are 0.55 and 0.538, respectively. These yield expected speedups of 1.434 and 1.455, a difference of only 0.021, which is not statistically significant. Thus, we continue to use Eq.10 in our main experiment for simplicity.

### C.3 Details of Main Results

More details are provided in Table 7. All reported speedup results are averaged over 10 independent runs. Results show that our `KNN-SSD` outperforms




(a) Summarization - CNN/DM

(b) Reasoning - GSM8K

(c) Translation - WMT16

(d) Storytelling - TinyStories

(e) Text-to-SQL - Spider2

Figure 10: Visualization of skipped layer set configuration of Qwen-2.5-14B optimized by Self-SD (Zhang et al., 2024) on different task domains.

the two Self-SD methods on both metrics, indicating our method can handle a more diverse input stream with stable inference acceleration.

| **Models** | **Methods** | $r$=0.0 | | $r$=0.3 | | $r$=0.7 | | $r$=1.0 | | Overall | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $M$ | $\alpha$ | $M$ | $\alpha$ | $M$ | $\alpha$ | $M$ | $\alpha$ | $M$ | $\alpha$ |
| LLaMA-2-13B | `Vanilla` | 1.00 | - | 1.00 | - | 1.00 | - | 1.00 | - | 1.00 | - |
| | `SSD_F` | 2.22 | 0.65 | 2.19 | 0.63 | 2.14 | 0.59 | 2.12 | 0.61 | 2.17 | 0.62 |
| | `SSD_M` | 2.50 | 0.64 | 2.58 | 0.70 | 2.53 | 0.69 | 2.52 | 0.68 | 2.53 | 0.68 |
| | `KNN-SSD` | **3.10** | **0.86** | **3.14** | **0.88** | **3.11** | **0.89** | **3.12** | **0.88** | **3.12** | **0.88** |
| LLaMA-2-13B -Chat | `Vanilla` | 1.00 | - | 1.00 | - | 1.00 | - | 1.00 | - | 1.00 | - |
| | `SSD_F` | 2.03 | 0.60 | 1.99 | 0.56 | 1.92 | 0.55 | 1.97 | 0.57 | 1.97 | 0.57 |
| | `SSD_M` | 2.10 | 0.56 | 2.14 | 0.61 | 2.18 | 0.59 | 2.15 | 0.58 | 2.14 | 0.59 |
| | `KNN-SSD` | **2.84** | **0.84** | **2.85** | **0.86** | **2.90** | **0.85** | **2.90** | **0.86** | **2.87** | **0.85** |
| Qwen-2.5-14B | `Vanilla` | 1.00 | - | 1.00 | - | 1.00 | - | 1.00 | - | 1.00 | - |
| | `SSD_F` | 2.41 | 0.82 | 2.40 | 0.82 | 2.44 | 0.84 | 2.48 | 0.85 | 2.43 | 0.83 |
| | `SSD_M` | 3.02 | 0.89 | 2.94 | 0.89 | 2.99 | 0.90 | 2.97 | 0.90 | 2.98 | 0.90 |
| | `KNN-SSD` | **4.35** | **0.99** | **4.42** | **1.00** | **4.40** | **1.00** | **4.38** | **0.99** | **4.37** | **1.00** |
| Qwen-2.5-14B -Instruct | `Vanilla` | 1.00 | - | 1.00 | - | 1.00 | - | 1.00 | - | 1.00 | - |
| | `SSD_F` | 2.12 | 0.80 | 2.16 | 0.80 | 2.16 | 0.80 | 2.10 | 0.79 | 2.13 | 0.80 |
| | `SSD_M` | 2.32 | 0.83 | 2.25 | 0.84 | 2.35 | 0.87 | 2.34 | 0.87 | 2.32 | 0.85 |
| | `KNN-SSD` | **3.78** | **1.00** | **3.69** | **0.99** | **3.71** | **0.99** | **3.75** | **1.00** | **3.73** | **1.00** |

Table 7: Comparison between `KNN-SSD` and two Self-SD methods. $r$ indicates the mix ratio of sample streams. We report the mean accepted length and token acceptance rate, which are denoted as $M$ and $\alpha$, respectively.